GI-Edition

Lecture Notes in Informatics

Gero Mühl, Jan Richling,
Andreas Herkersdorf (Hrsg.)

ARCS 2012 Workshops

28. Februar – 2. März 2012
München

200

Proceedings

G. Mühl, J. Richling, A. Herkersdorf (Hrsg.) ARCS 2012 Workshops

Gero Mühl, Jan Richling, Andreas Herkersdorf (Hrsg.)

ARCS 2012 Workshops

28. Februar – 2. März 2012
München, Germany

Gesellschaft für Informatik e.V. (GI)

Lecture Notes in Informatics (LNI) - Proceedings
Series of the Gesellschaft für Informatik (GI)

Volume P-200

ISBN 978-3-88579-294-9
ISSN 1617-5468

Volume Editors
Prof. Dr.-Ing. Gero Mühl
Universität Rostock
Albert-Einstein-Straße 22
18051 Rostock, Germany
Email: gero.muehl@uni-rostock.de

Dr.-Ing. Jan Richling
Technische Universität Berlin
Einsteinufer 17
10587 Berlin, Germany
Email: richling@cs.tu-berlin.de

Prof. Dr. sc.techn. Andreas Herkersdorf
Technische Universität München
Arcisstraße 21
80290 München, Germany

Thematics
Andreas Oberweis, Karlsruher Institut für Technologie (KIT), Germany

printed by Köllen Druck+Verlag GmbH, Bonn

Message from the Chairs

The conference on Architecture of Computing Systems (ARCS) has a long and prestigious history in computing systems. Five workshops are co-located with the conference covering a large field of topics. The papers of the following workshops are included in these proceedings:

ASPRIT
: Workshop on Architectures for Self-Organizing Private IT-Spheres

CSECS
: Workshop on Complex Sciences in the Engineering of Computing Systems

PARMA
: 3rd Workshop on Parallel Programming and Run-Time Management Techniques for Many-core Architectures

PASA
: 10th Workshop on Parallel Systems and Algorithms

VERFE
: 8th Workshop on Dependability and Fault-Tolerance

For each of these workshops, the workshop committees selected high quality contributions within a competitive peer review process. Many thanks go to the organizers of the workshops: Detlef Krömker, Ralf Dörner, Uwe Brinkschulte, Ingo Scholtes, Claudio J. Tessone, Jacob Beal, Alex Bartzas, Giovanni Agosta, Jörg Keller, Rolf Wanka, Karl-Erwin Großpietsch, and Jörg Henkel.

In addition to the workshops, the conference also features tutorials on "Organic Computing: Status and Outlook" as well as on "Partial Reconfiguration of FPGAs in Practice - Tools and Applications". A paper related to the latter tutorial is included in this proceedings. We are grateful to the people that made these tutorials possible: Christian Beckhoff, Volker Breuer, Christopher Dennl, Michael Feilen, Dirk Koch, Christian Mueller-Schloer, Hartmut Schmeck, Walter Stechele, Juergen Teich, Jim Torresen, and Daniel Ziener.

The workshop and tutorial program would not have been possible without the help of many people: The program co-chairs Uwe Brinkschulte and Kay Roemer, the local and publication chair Walter Stechele and the financial chair Thomas Wild.
We would also like to thank Cornelia Winter (GI) and Jürgen Kuck (Köllen Verlag) for the uncomplicated and flexible handling of the proceedings publication process.
Our final thank goes to all the authors for their contributions that make this an attractive workshop program. We wish you all interesting discussions during the two days of workshops and tutorials.

Gero Mühl	ARCS 2012 Workshop Chair
Jan Richling	ARCS 2012 Tutorial Chair
Andreas Herkersdorf	ARCS 2012 General Chair

Table of Contents

Workshop on Architectures for Self-Organizing Private IT-Spheres

1st International Workshop on Complex Sciences in the Engineering of Computing Systems

Workshop on Dependability and Fault Tolerance

Tutorial on Reconfigurable Systems

3rd Workshop on Parallel Programming and Run-Time Management Techniques for Many-core Architectures

10th Workshop on Parallel Systems and Algorithms

ASPRIT 2012

Workshop on Architectures for Self-Organizing Private IT-Spheres

organized by

Detlef Krömker, Johann Wolfgang Goethe-Universität, Germany
Ralf Dörner, Hochschule RheinMain, Germany
Uwe Brinkschulte, Johann Wolfgang Goethe-Universität, Germany

QoS-based Testing and Selection of Semantic Services

Jan Schaefer Reinhold Kroeger
Design Computer Science Media Department
RheinMain University of Applied Sciences
Wiesbaden, Germany
{jan.schaefer|reinhold.kroeger}@hs-rm.de

Simone Meixler Uwe Brinkschulte
Department of Computer Science
Johann Wolfgang Goethe University
Frankfurt am Main, Germany
{meixler|brinks}@es.cs.uni-frankfurt.de

Abstract:
The importance of context-awareness is constantly increasing. Users want systems to react dynamically to their current context (e.g. their location). For emerging home service platforms, this represents a key element, as it allows systems to increase the users' comfort tremendously by acting on sensed and deducted situations. As such systems feature a lot of dynamic interaction between services, it has to be ensured that service selections and bindings simply *work*. This paper proposes an approach for QoS-based testing and selection of semantic services, which employs a service tester that uses genetic algorithms to check and monitor QoS properties. These properties are used to complete the semantic descriptions of services running on the platform, which are managed by a semantic service registry.

1 Introduction

The increasing amount of IT hardware and software (e.g. "apps") in personal living spaces allows users to experience more and more services both online via the Internet and offline in their homes: personal computers, smartphones, tablets, home theater devices and smart homes are becoming more and more intertwined with each other and online services (Internet and/or cloud services) forming an *Ambient Assisted Living* (AAL) home service platform. The term *Ambient Assisted Living* (AAL) was coined by the European Commission's Information Society in 2004, as AAL research activities were prepared in a special support action project that was part of the 6^{th} European Framework Programme [Gmb04]. AAL-related technologies introduce a complexity that users can or do not want to administrate themselves anymore. The increase of networked electronics and software can be compared to the increase of assistive and media technologies in cars during the last two decades. Unlike in cars, however, personal electronic devices and applications are much more open to (deliberate) modifications and data exchange, which makes securing their handling even more difficult. Finding and selecting the right service when using this home

service platform can become a time-consuming and even dangerous process (from an IT security perspective), if there are multiple service providers and a malicious or malfunctioning service is selected. In addition, usually multiple applications are active in parallel requiring binding decisions almost constantly.

Static binding between client consumer and provider, which usually takes place once at the application's compile/link time, is not a feasible approach here. Even dynamic binding with its implementation-dependent, tight coupling between consumer and provider is not sufficient. As smart homes consist of heterogeneous, distributed systems with varying computation capabilities, they represent an ideal environment for service-oriented computing with its loose coupling thus supporting the required runtime dynamics, which explains the advent and propagation of home service platforms in general. However, the service interface descriptions and, even more importantly, any existing service properties offered by service providers (typically name-value pairs) are still statically defined, as they do not reflect the runtime state of the provided implementations. This results in syntactic service look-ups, in which the binding only depends on the service interface and its statically defined properties. To support the definition of extended and dynamic service descriptions that are integrated with a common context model – an ontology – for the platform, the ability to define and process semantics is needed. The approach presented here allows clients to define semantic queries for services, which can be processed by the service platform. As these queries can use the semantics of both services and platform, it is possible to build adaptive, context-aware applications, in which clients can react to changing service and context properties, as service properties reflect the runtime state of the platform and its components.

With respect to the functional properties a service defines (e.g. regarding its input and output parameters), clients (i.e., users or applications) might require non-functional service capabilities such as *Quality of Service* (QoS) properties. In this case, the clients' needs can be formulated explicitly or as part of their preferences. Service properties, on the other hand, are typically either defined statically or dynamically gathered at runtime (e.g. by application instrumentation). The first approach is both hard to achieve and insufficient as it doesn't reflect the user experience: a service might be fast if accessed in one geographic region but slow in another. The second approach, however, might also be inapplicable as ongoing runtime instrumentation might lower the service's performance too much impairing end user experience. This paper proposes a third approach: The integration of semantic service descriptions with service testing and probing capabilities. Here, service implementations register a test interface, which is used to determine runtime QoS properties, before the service is used for the first time. The resulting QoS properties become part of the service's description and can be used either for service look-up or selection of the fittest service if multiple services fulfill the client's requirements. The subsequent service probing adapts automatically to the currently offered services as well as to the usage of these services. We also consider to keep the testing effort within reasonable bounds.

This paper is structured as follows: Section 2 presents required background information and is followed by Section 3, which presents related approaches. The subsequent Section 4 introduces the approach for QoS-based testing and selection of semantic services. Section 5 discusses the current state of and the next steps for the work presented here.

2 Background

2.1 The Web Ontology Language

The W3C specification for the *Web Ontology Language* (OWL) [Gro09] defines a family of languages to support the *Semantic Web* vision. While RDF already allows addressing resources and their properties via URIs thus creating a formal, triple-based representation of information, OWL adds machine-processable semantics enabling automated interpretation of this information by computers. Furthermore, OWL supports an inference mechanism – so-called *reasoning* – that allows the automated derivation of new information from existing descriptions of things and their relationships. To support reasoning and limit its complexity, which leads to increasing processing times, the OWL specification defines three OWL sublanguages with different levels of expressiveness: *OWL Lite*, *OWL DL* (includes Lite) and *OWL Full* (includes DL), where OWL DL was designed to provide maximum expressiveness while retaining computational completeness. Apart from these sublanguages, the OWL specification also supports several exchange syntaxes for specification and exchange purposes with varying degrees of human readability.

An OWL ontology contains statements consisting of *Resource-Property-Value* triples (e.g. *Father hasName 'John'*). A resource has a *Class* definition and features a number of *Individuals*, which represent the actual objects in a domain. Properties may feature *Domains* and *Ranges*: *Data Properties* associate resources with constants, whereas *Object Properties* associate resources with each other. In addition, properties may possess more detailed logical capabilities such as being functional, transitive, symmetric, reflexive, inverse and/or disjoint.

2.2 Genetic Algorithms

Genetic Algorithms are used for solving optimization problems and are based on the biological evolution theory. Terms that are often used for genetic algorithms are listed in table 1. Furthermore, the table shows the biological and IT specific translation of these terms.

Term	Biological Translation	IT Translation
Population	Set of individuals	Set of solution candidates
Parents	Mating subset of population	Selected individuals for generating new solution candidates
Fitness	Conformity of an individual	Quality of candidate solution
Chromosom	Properties of a individual	String
Gen	Part of a Chromosom	char
Allele	Characteristic of a gen	value of char
Locus	Location of a gen	Position of a char

Table 1: Genetic terms

From random variation new advantageous properties develop and will establish oneself in

the current environment. As the environment changes, properties can become disadvantageous. They may get replaced by properties that better fit to the current environment. For generating new solutions, genetic operators are used. Before using a genetic operator, individuals have to be selected. There exist many different types of selection methods. Fitness proportionate selection also called roulette-wheel selection is used as basis for our selection method, which is introduced in Section 4.4. After selecting two individuals, the genetic crossover operator can be used. Furthermore, random variation can be accomplished with the mutation operator. After creating the new population, the mutation operator can be used to choose randomly an allele and change its value. The reproduction operator is used to take individuals unchanged into the next generation. For the testing method, we have developed specific crossover, mutation and reproduction operators. A detailed explanation of these newly defined operators can be found in Section 4.4.

3 Related Work

With maturing tool support and increasing processing power, semantic (context-aware) services have gained enormous interest in recent years. Several approaches to semantics definition have been developed and compared in the last decade [CDM+04, ZKN06]. However, OWL-S, the *Semantic Markup for Web Services* extension for OWL, which provides additional concepts for specifying semantic processes and services, remains the one most widely used today, although it focuses on Web services environments. Due to its modularity, however, it allows the definition of *Service Groundings* for other service architectures.

Even before semantic services were examined in particular, several context modeling approaches for home service platforms emerged. Gu, Pung, Zhang [GPZ04] propose using layered ontologies for context models in the home domain. It covers widely-acknowledged concepts that also appear in more recent publications (e.g. activities, locations, persons, devices including a custom service concept). However, the interaction of formal service definitions with runtime systems (actual groundings) and how user preferences influence the platform's behavior are not discussed. Daz Redondo et al. [RVC+08] propose the combination of OWL-S with an OSGi grounding called *OWL-OS*, although neither QoS properties nor semantic queries for service selection are supported (only key-value pairs). However, the authors use OWL-S service categories as a means to group services into aspects (e.g. Lighting), which can be used by clients during service look-up. Romero et al. [RHT+11] propose a platform based on the *Service Component Architecture* (SCA), which provides similar service and component abstractions as OSGi. The paper focuses on device integration over heterogeneous communication protocols into an event processing architecture, which is able to process data from a wide collection of devices (including sensor networks). However, a semantic abstractions for implementation details and support for the deployment of additional (3rd party) services are not presented.

Apart from home service platforms, a broad collection of service discovery and binding approaches have been developed in the past. The most widely used specifications for this today, especially in living environments, is *Universal Plug and Play* (UPnP) [UPn11].

Apart from its convincing technical capabilities, it lacks OWL's modeling capabilities as well as support for rules, queries and reasoning. Thus, it rather can be seen as a potential grounding for OWL-S services.

4 Approach

The work presented here proposes enhancements to home service platforms, which allow extending, testing and integrating the collection of applications and services dynamically based on semantic descriptions of service properties.

4.1 A Platform for Semantic Services

The approach presented here proposes adding semantic interfaces and continuous testing to applications and services that integrate with common OWL ontologies provided by a *Semantic Service Registry*. This registry allows extending an existing service registry for non-semantic services by adding the capability to manage ontologies and supporting semantic service queries. This extension supports the installation of non-semantic and semantic services in parallel. An overview of the surrounding platform is given in [Sch10]. In addition, the *Service Tester* component uses the descriptions of registered services to gather runtime QoS properties once after service registration. After that, the Service Tester is able to probe services on demand. The *Knowledge Module* stores the context and application ontologies including related rules, which are accessed by the other components. With changing system context, the contained ontologies are updated by incoming events resulting in constantly changing facts and, thus, also changing service properties. This dynamic can be used by clients in service look-ups, as bound services can differ between two subsequent look-ups due to interim model changes (e.g. state, location or performance change). The interaction between the involved components is displayed in Figure 1.

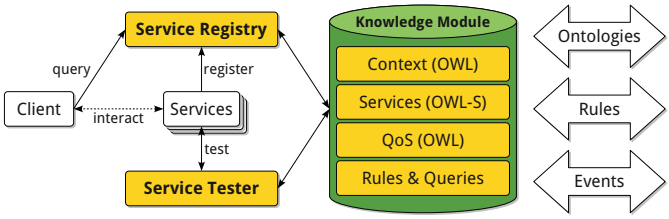


Figure 1: Service Registration and Testing

The context model has been constructed around the core concepts *Person*, *Computational Entitiy*, *Location* and *Activity* and is based on the context model from Gu et al. [GPZ04] introduced in Section 3, which has been extended with more detailed concepts for computational entities, services, activities and user preferences (among others) for this approach,

although the latter two are not covered here. Figure 2 presents a condensed version of the context model, in which the colored shapes represent some of the additions to the original model. The service-related ontologies contain service descriptions from service groundings (implementations) to formal service models and are defined using OWL-S. The *Service Registry* is responsible for registering and looking up services in the knowledge module and the platform's registry mechanism.

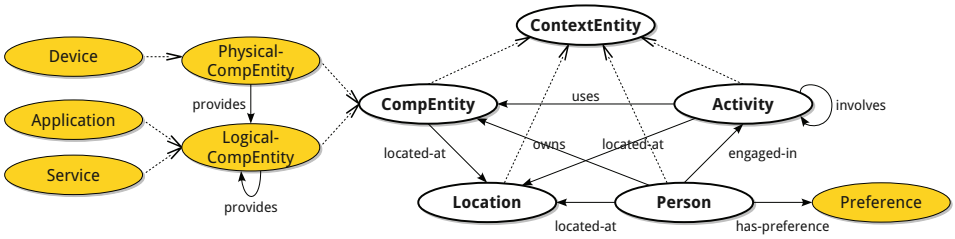


Figure 2: Context Model (condensed)

4.2 Service Registration

If a service implementation is installed by a service provider, it registers its implementation details – the service grounding – with the service registry assuming that the ontology describing the implementation integrates with the context and service ontologies. Subsequently, this integration allows the registry to find the service implementation, if the associated service is looked-up.

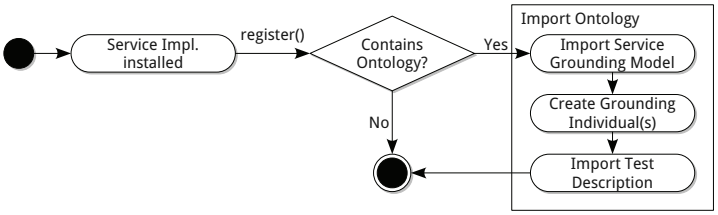


Figure 3: Service Registration Process

Apart from functional service properties (interfaces and parameters), services register additional context-related or QoS-related properties that can be specified as requirements by clients during service look-up. If these properties are dynamic, they are updated by the registry at runtime (e.g. by testing as described in Section 4.4). The registry is also responsible for removing service groundings, if service implementations are uninstalled or unavailable (e.g. caused by service failures). It is also possible for clients or services to register and update context or service ontologies without providing an implementation for them, as ontologies and implementations are decoupled.

4.3 Service Selection

If clients require a service, they can query the service registry for implementations of that service. This can be done using the platform's own service selection approach (if one exists), or by executing *SPARQL* queries [PE08] on the semantic service registry. The latter approach is required, if semantic properties are used for the service look-up. For the definition of look-ups, clients can use both functional and non-functional properties resulting in comprehensive queries. This can lead to situations, in which no service is able to meet these requirements. However, typically clients only define a limited set of requirements and end up with a selection of functionally equal services. In this case, the registry relies on the results of the service tests to select the fittest service. The service selection process is shown in Figure 4.

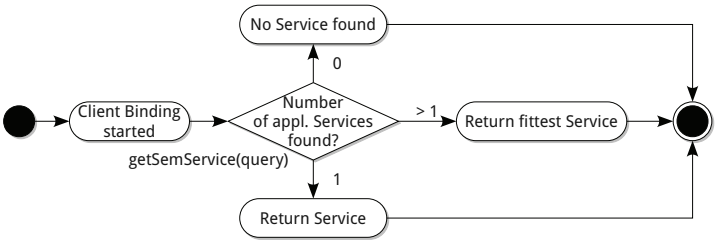


Figure 4: Service Selection Process

As an example, a client might look for a service providing off-site health monitoring. Here, services might feature more or less static properties such as the service provider's general trustability or service cost. Properties such as availability or response time, on the other hand, are dynamic properties, which would be collected by the service tester.

4.4 Service Testing

Given the new situation of being able to dynamically extend the service platform by services from different providers results in a great responsibility for testing these services. The need of testing is rooted in the fact that services can be added, deleted or substituted. Therefore no proven information exists how they perform on given client requests. As services of different service provider are supposed to interact, it is necessary to be able to rely on information given for these services (for example QoS Properties). Hence, a testing method should fulfill the requirements of adapting to currently offered services and client requests as well as being independent from the service provider offering a service. Therefore, we combine the approach of using a knowledge module with the approach of using genetic algorithms [MB10] for creating test sets.

QoS Property	Category	Valid Values
Response Time	1	$\leq 5ms$
	2	$6ms - 10ms$
	3	$11ms - 20ms$
	4	$\geq 21ms$

Table 2: Category Example

4.4.1 Test Case Adaption

In the following, a test set denotes a number of l different test cases. It should be taken care that a test set does not contain too many test cases because every test case results in a service-request and every service-request generates load for the service provider system. Therefore, it is necessary to find a suitable selection of test cases to compare and test services. As suggested in [Xia06], test cases are categorized by QoS properties and quality levels. Therefore, every QoS property is divided into different quality categories on the test system. These categories are used to distinguish between different QoS requirements. The test system manages m test sets for one category. A QoS property is e.g. the response time. Table 2 shows an example for such response time categories.

Crossover Operator	Modified Crossover Operator
Two offspring	One offspring
Locus of a gen is important	Locus of a gen is not important
All alleles are handed on to the offspring	A distinction is made between dominant and recessive allels

Table 3: Crossover Operator Modifications

If a client discovers a non-functional error, he has to send the ID of the service causing this error to the test system, the corresponding input, and the violated requirements. The test system uses these errors as test cases and generates test sets with this information. An error counter is attached to every test case. The value of the error counter reflects the up-to-dateness of a test case. When an error is reported, the corresponding error counter will be increased. In case of a new error, a test set will be chosen randomly and a test case to be substituted will be selected. For the selection, it has to be checked if the test set contains test cases having an error counter of value zero. In this case, one of them will be chosen randomly to be substituted. Otherwise, the selection of a test case will be based on a probability reverse proportional to its error counter. Test sets cannot be generated by errors only, but also by recombination of existing test sets. The advantage of recombinations is that good test cases can be unified to one test set. Hereby, is it possible to create better test sets from already existing ones. The approach presented in this paper is based on a genetic algorithm. This algorithm shows differences to standard genetic algorithms to reflect the considered application. Individuals are given by test sets of a QoS property category.

Test cases form the alleles and the fitness of a test set (individual). t_j is denoted as $f(t_j)$ and equals the sum of the error counters belonging to test cases of t_j . Let ec_i be the

error counter of test case i. Using the fitness, we can calculate the probabilities of the individuals. The probability p_j of an individual j depicts its possibility to survive, because an individual is chosen for the reproduction operator as well as for the crossover operator with probability $p_j = \frac{f(t_j)}{F}$, where $f(t_j) = \sum_{i=1}^{l} ec_i$ and $F = \sum_{j=1}^{l} f(t_j)$

For the crossover operator, two individuals have to be chosen. The crossover operator will produce one offspring from these two individuals. Not all alleles will be handed to the offspring, it will be distinguished between dominant and recessive alleles. The selection of alleles, which will be part of the subsequent generation, is affected by the probabilities calculated based on the error counters. The probability to include a test case (allele) in the next generation is proportional to the value of its error counter. First of all, the sum S of all error counters for both test sets has to be provided to calculate the probability $S = \sum_{j=1}^{2} \sum_{i=1}^{l} ec_{ij}$, where ec_{ij} is the error counter of test case i in test set j. Afterwards, the probability $p_{ij}^C = \frac{ec_{ij}}{S}$ can be calculated. Because we do not put back a selected allele, the values of S and p_{ij}^C change after every selection. Test cases handed on to the next generation by the crossover operator are called *dominant* alleles while the others are called *recessive*. Changes made to the crossover operator compared to the standard crossover operator are listed in Table 3.

The reproduction operator moves an individual to the next generation without any changes. The only modification of this operator is not to allow an individual to be chosen twice. Therefore the value of F as well as p_j change after every selection. A major problem of the crossover and reproduction operator is their seldom appliance to test sets having a low fitness value. Even with this low fitness value, these test sets might contain some test cases with high error counter values. Due to low error counter values of the majority of test cases, the overall sum nevertheless would be low. These high rated test cases would rarely have a chance to be handed on to the next generation without the mutation operator. For every mutation a test set has to be chosen from the current generation. The selection of test set j from all available test sets is done with probability $p_j^M = 1 - p_j$, where p_j is defined as described before. A test set is chosen for mutation with a probability reverse proportional to its fitness. This allows to extract test cases with high error counter values from test sets with a low fitness. From the chosen test set, a test case s_i will be taken with the probability p_i^{MST}. This probability is proportional to the value of the corresponding error counter ec_i

Mutation Operator	**Modified Mutation Operator**
Random selection of an allele which is to be mutated	The probability of selecting an allele to be mutated depends on its error counter
New allele is chosen randomly	New allele is chosen from parent generation with the help of the Fitness and error counters
Mutation is not cancelled	Mutation will be cancelled if it causes a duplicate allele

Table 4: Mutation Operator Modifications

of test case $s_i : p_i^{MST} = \frac{ec_i}{\sum_{j=1}^{l} ec_j}$. A test case e_k from the next generation has to be chosen to be substituted by test case s_i. The test set in which the substitution takes place is chosen randomly. If the test case s_i is already an element of this test set, the mutation will be cancelled. This avoids the presence of duplicate test cases in a test set. Test case e_k, which has to be substituted, will be chosen with probability $p_k^{MET} = 1 - \frac{ec_k}{\sum_{j=1}^{l} ec_j}$. This results in test cases with a low error counter having a higher probability to be substituted. The modifications of the mutation operator are summarized in Table 4.

Genetics provide a solution for adaption of individuals to a changing environment. This is one reason for taking genetic algorithms to address the issue of test case adaption. Another important issue is that genetics only takes the sum of all properties of an individual to calculate its fitness. This means an individual having a high fitness may even have low rated properties. The importance of handing on properties with a low rating is given by the changing environment. The rating of a property can improve as the system context changes. This reflects the changing ability of test cases to detect faults as the offered services and the requests change. We are confronted with the problem of an always changing search space and the consequence of never reaching a final optimum. The goal is to keep a variety of test cases and also to consider their rating changes over time. This implies giving newly registered test cases a chance to improve. Therefore, test cases with a low error counter should also get a chance to be integrated into the used test set.

4.4.2 Initial Test Setup

Initially, no test sets exist. However, services have to be tested from the start to find out which requirements they fulfill. Therefore, this phase will be conducted similarly to [TPC+03]. When a service is registered to the test system, the service provider has to deliver test cases for this service. Furthermore, the service provider has to deliver information about the service's QoS properties. Optionally, the service provider can specify, for which QoS property a test case is especially applicable. If no information is given, the test case will be seen as relevant for all QoS properties. The QoS property category of a test case is given by the QoS properties of the corresponding service. These initial test cases will be put into a pool belonging to the corresponding QoS property and category. There is a dedicated pool for every type of service and QoS property category. Furthermore, the pool can have test cases sent by a client. These test cases might not have caused any errors yet, but the client favors them to be tested. In this case, the client has to act like a service provider. The client has to send the type of service to be tested, the QoS property and the QoS property category. This enables testing a service not only with its own provided test cases, but also with other test cases provided by services of the same category or by clients.

The m test sets are generated with randomly chosen test cases from the corresponding pool. Test cases are removed from the pool as soon as they are used in a test set. As an improvement to the work of [TPC+03], services are also tested with inputs, which are currently used by clients and already have caused errors.

The step of generating a pool and m test sets is only necessary, if a service of a not existing functional domain is registered. Errors and inputs can be reported continuously to the service registry. Furthermore, the adaption and testing is a continuously repeating process of our test system.

5 Current State and Outlook

The approach presented in this paper is part of ongoing research, which focuses on enhancing the (self-)management capabilities of emerging home service platforms. Although the context model and the semantic service extensions presented here are fundamental elements of context-aware systems, they are just the basics for the creation of these self-management capabilities. As the introduction of semantics, just like service-orientation, decouple concepts from implementation technologies, the semantic service registry concept presented here is not limited to a specific implementation technology. For the prototypical implementation, however, the OSGi service platform [OSG09] was selected in combination with the *Jena Semantic Web Framework* [jen08] and the *Pellet OWL 2 Reasoner* [SPG+07] for ontology and SPARQL query processing. Here, the semantic service registry extends the default OSGi service registry and offers an additional client API for semantic service selection. Service registrations use either distinct or default OSGi API methods. In the case of the latter, OSGi service registration calls are intercepted and their associated bundles inspected for ontologies using OSGi's *Event Hook* mechanism that allows the inspection of OSGi services, whenever their runtime state changes (i.e., they are registered or unregistered). As clients have to formulate specific queries for semantic service look-ups, the semantic API offers an additional query-based method for this. At the current stage, however, the client still has to address the OSGi service interface of the targeted semantic service as part of the invocation, as abstract service queries including parameters cannot be mapped to arbitrary OSGi services currently, yet. For goal- or intention-driven semantic services, the service semantics have to be decoupled from service interfaces as well.

Part of this research focuses on the development of a self-management module for service platforms, which relies on dedicated management ontologies and associated rules to manage the platform – applications, services and devices – autonomically. As monitoring and state assessment are critical for this task, this management module relies on sensors and tests (as described in Section 4.4) to gather runtime information required to compute an appropriate management action to improve the managed system's state. Here, the approach for service testing with test sets plays an important role, as the effectiveness of management actions could be tested in this setup as well. Apart from the self-management module, test sets are generated by occurring faults and with a genetic algorithm. This results in test sets, which adapt to the current faults and offered services. The advantages of this testing method, therefore, lie in dynamic service environments. Future work will implement, test, and compare this approach with other solutions. Furthermore we think about expanding the fitness function to include other parameters besides the error counter.

References

[CDM+04] Liliana Cabral, John Domingue, Enrico Motta, Terry R. Payne, and Farshad Hakimpour. Approaches to Semantic Web Services: An Overview and Comparison. In *European Semantic Web Conference*, May 2004.

[Gmb04] VDI/VDE Innovation + Technik GmbH. Ambient Assisted Living - Preparation of an Article 169 Initiative. `http://http://www.aal169.org`, September 2004.

[GPZ04] T. Gu, H.K. Pung, and D.Q. Zhang. Toward an OSGi-based infrastructure for context-aware applications. *Pervasive Computing, IEEE*, 3(4):66 – 74, Oct.-Dec. 2004.

[Gro09] W3C OWL Working Group. OWL 2 Web Ontology Language Document Overview (W3C Recommendation). `http://www.w3.org/TR/owl2-overview/`, October 2009.

[jen08] Jena - Semantic Web Framework for Java. `http://openjena.org`, January 2008.

[MB10] Simone Meixler and Uwe Brinkschulte. Test Case Generation for Non-functional and Functional Testing of Services. *Object-Oriented Real-Time Distributed Computing, IEEE International Symposium on*, pages 245–249, 2010.

[OSG09] OSGi Alliance. OSGi Service Platform Release 4 (Version 4.2) Core Specification. `http://www.osgi.org/Download/Release4V42/`, May 2009.

[PE08] Eric Prud'hommeaux and Andy Seaborne (Editors). SPARQL Query Language for RDF (W3C Recommendation). `http://www.w3.org/TR/rdf-sparql-query/`, January 2008.

[RHT+11] Daniel Romero, Gabriel Hermosillo, Amirhosein Taherkordi, Russel Nzekwa, Romain Rouvoy, and Frank Eliassen. The DigiHome Service-Oriented Platform. *Software: Practice and Experience*, 2011.

[RVC+08] R.P. Diaz Redondo, A.F. Vilas, M.R. Cabrer, J.J.P. Arias, J.G. Duque, and A.G. Solla. Enhancing Residential Gateways: A Semantic OSGi Platform. *Intelligent Systems, IEEE*, 23(1):32 –40, Jan.-Feb. 2008.

[Sch10] Jan Schaefer. A Middleware for Self-Organising Distributed Ambient Assisted Living Applications. In *Workshop Selbstorganisierende, Adaptive, Kontextsensitive verteilte Systeme (SAKS)*, March 2010.

[SPG+07] Evren Sirin, Bijan Parsia, Bernardo Cuenca Grau, Aditya Kalyanpur, and Yarden Katz. Pellet: A Practical OWL-DL Reasoner. *Web Semant.*, 5:51–53, June 2007.

[TPC+03] W. T. Tsai, R. Paul, Z. Cao, L. Yu, A. Saimi, and B. Xiao. Verification of web services using an enhanced uddi server. *Object-Oriented Real-Time Dependable Systems, IEEE International Workshop on*, 0:131, 2003.

[UPn11] UPnP Forum. About UPnP Forum. `http://upnp.org/about/`, August 2011.

[Xia06] Jinchun Xia. Qos-based service composition. In *COMPSAC '06: Proceedings of the 30th Annual International Computer Software and Applications Conference*, pages 359–361, Washington, DC, USA, 2006. IEEE Computer Society.

[ZKN06] Jiehan Zhou, J.-P. Koivisto, and E. Niemela. A Survey on Semantic Web Services and a Case Study. In *Computer Supported Cooperative Work in Design, 2006. CSCWD '06. 10th International Conference on*, pages 1 –7, May 2006.

Adaptive Content Distribution Network for Live and On-Demand Streaming

Yuta Miyauchi, Noriko Matsumoto, Norihiko Yoshida

Graduate School of Science and Engineering
Saitama University
Saitama 338-8570, Japan
{yuta, noriko, yoshida}@ss.ics.saitama-u.ac.jp

Yuko Kamiya, Toshihiko Shimokawa

Graduate School of Information Science
Kyushu Sangyo University
Fukuoka 813-8503, Japan
{kamiya, toshi}@nw.is.kyusan-u.ac.jp

Abstract: We have proposed an adaptive content distribution network (CDN), FCAN (Flash Crowds Alleviation Network), which changes its structure dynamically against *a flash crowd*, that is a rapid increase in server load caused by a sudden access concentration. FCAN in our preceding studies responds only to static content delivery. In this paper, we extend FCAN to alleviate flash crowds in video streaming. Through some experiments, we confirmed that FCAN for video streaming is effective to alleviate flash crowds.

1 Introduction

When a Web site catches the attention of a large number of people, it gets an unexpected and overwhelming surge in traffic, usually causing network saturation and server malfunction, and consequently making the site temporarily unreachable. This is the "flash crowd" phenomenon on the Internet. An example of a flash crowd is Figure 1, which shows the traffic volume of Web site during the solar eclipse based on a real access log provided from "LIVE! ECLIPSE 2006" [LE06]. During the solar eclipse, the accesses from clients increased several times higher than the normal condition by flash crowds.

We have proposed an adaptive content distribution network (CDN), FCAN (Flash Crowds Alleviation Network), which changes its network structure adaptively depending on a load fluctuation against flash crowds [Pan06, Yos08]. FCAN only focused on static content delivery in our preceding studies, thus in this study, we extend FCAN to video streaming.

A large amount of clients receive a sequential stream from a streaming server, therefore network traffic is concentrated on a specific site on the Internet. To assure resilience in

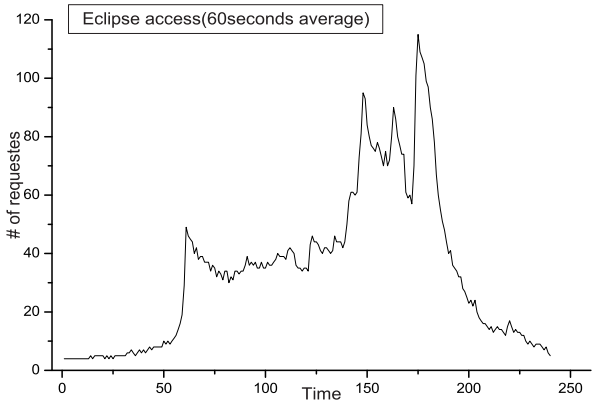


Figure 1: Flash Crowds on “LIVE! ECLIPSE 2006” site

P2P video streaming, the FCAN framework is thought to be promising. There is a survey paper on resilience in P2P video streaming [Abb11], however, there is no system which changes its network structure dynamically according to load fluctuation.

In this paper, we describe FCAN’s extension. It uses Apple’s HTTP Live Streaming [AD11a] for stream segmentation and distributed delivery. This paper is organized as follows: Section 2 provides a brief overview of HTTP Live Streaming. Section 3 presents an overview of our previous FCAN for static content delivery. Section 4 gives FCAN’s extending design for video streaming. Section 5 describes preliminary experiments with a simple prototype. Section 6 contains some concluding remarks.

2 HTTP Live Streaming

Conventional standard streaming protocols such as progressive download and real-time streaming do not allow switching of stream source servers on the client side dynamically. Therefore, massive accesses from clients concentrates on a particular site on the Internet. Accordingly, the server and its surrounding network choke up, and a flash crowd occurs.

In order to resolve this problem of conventional protocols, Apple has introduced a new protocol for video streaming, HTTP Live Streaming (also known as “HLS”). It has been proposed as a standard draft for the Internet Engineering Task Force [Pan11]. Figure 2 shows an overview of this protocol.

The server starts providing a video stream with the following procedure: (1) Encodes an audio/video inputs; (2) Divides the encoded stream into a set of media segments (“.ts” files), and makes an index (“.M3U8” file) which refers them; (3) Delivers them to clients using HTTP on the Internet.

This protocol enable a client to switch the source server dynamically as opposed to the

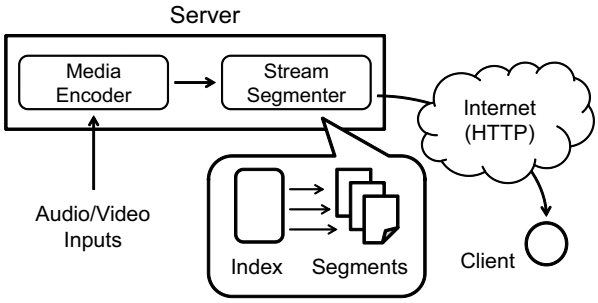


Figure 2: HTTP Live Streaming Overview

conventional streaming protocols. The delivery system archives load distribution easily with additional servers.

As another key feature, HTTP Live Streaming supports "adaptive bitrate." The server provides alternative streams with different quality levels of bandwidths, so as to enable a client to optimize the video quality according to the network situation, as the load on the network and CPU, both on the server side and the client side, fluctuates on a frequent basis.

3 FCAN

FCAN is an adaptive CDN which takes the form of C/S or CDN depending on the amount of accesses from clients. Specifically, in the C/S mode, a server provides contents to clients as in a traditional C/S. In the CDN mode, when the server detects a coming of a flash crowd, volunteer cache proxies in the Internet construct a temporary P2P network and provide the content on behalf of the server. These volunteer proxies are recruited in advance out of providers and organizations. In case servers in such providers and organizations suffer from flash crowds, they will be helped by other volunteer proxies. FCAN is built upon this mutually-aiding policy. Figure 3 shows an overview of FCAN.

In our preceding studies, we summarized some researches to alleviate flash crowds [Yos08]. These researches are divided into three categories: server-layer, inter-mediatelayer and client-layer solutions, according to typical architectures of networks. FCAN is an intermediate-layer solution, which employs an Internet infrastructure of cache proxies to organize a temporal P2P-based proxy cloud for load balancing. However, FCAN has some extensions with some dynamic and adaptive features. Our FCAN studies achieved very promising results regarding static content delivery on the real Internet [Miy11].

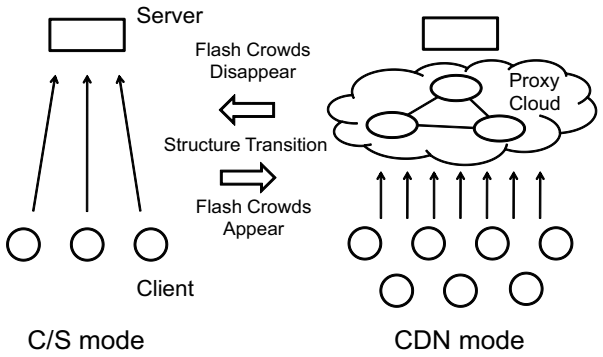


Figure 3: FCAN Overview

4 FCAN for Streaming

4.1 Structure Transition

First, We changed the behavior of structure transition in the previous FCAN design.

The server and the cache proxies in the proxy network always monitor the amount of accesses they receive from clients and evaluate the load on the network. The system switches to the CDN mode if all nodes' loads are higher than a certain threshold, and switches back to the C/S mode if lower. Each cache proxy sends its own load information to the server periodically, and the server determines whether to perform structure transition.

We use two thresholds to prevent "thrashing" between the two mode. The threshold for transition from the C/S mode to the CDN mode is set to higher than the one for transition from the CDN to C/S.

In peaceful times, the conventional C/S architecture satisfies most of the client requests. A server and cache proxies, both of which comprise FCAN, do little more than what normal ones do. When a flash crowd comes, the server detects the increase in traffic load. It triggers a subset of the proxies to form an overlay, through which all requests are conducted. All subsequent client requests are routed to this overlay.

The server-side procedure is outlined as follows: (1) Selects a subset of proxies to form a CDN-like overlay of surrogates, and builds a distribution tree; (2) Pushes the index file and stream segments to the node of the distribution tree, so as to meet the real-time constraint of the video streaming; (3) Prepares to collect and evaluate statistics for the object from the involved proxies, so as to determine dynamic reorganization and release of the overlay.

The proxy-side procedure is outlined as follows: (1) Changes its mode from a proxy to a surrogate (or, in the strict sense, a mixed mode of a forward proxy and a surrogate); (2) Stores flash-crowd objects (except the index file) permanently, which should not expire until the flash crowd is over; (3) Begins monitoring the statistics of request rate and load,

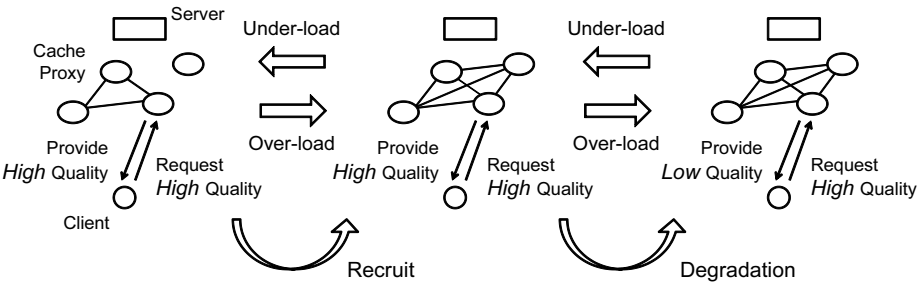


Figure 4: Handling of load increase in CDN mode

and reporting them to the server periodically.

In the live streaming, the index file is updated periodically, therefore the server monitors its composition, and pushes the segments at an appropriate time. Meanwhile, when the proxy is released by the server, it discards the index so as to keep the consistency among the nodes.

When the member server detects the leaving of the flash crowd, the involved proxies are dismissed one by one with the following procedure: (1) The server notifies the proxy to be dismissed; (2) The server requests the related proxies to modify the relation of connection; (3) The proxy changes its mode from a surrogate to a proxy.

The CDN-like overlay transits back to the normal C/S mode when all the proxies are dismissed. They are not all dismissed at once, since the low load may be just temporary, and the system should therefore remain in the anti-flash-crowd mode for a while.

4.2 Dynamic Resizing and Quality Restriction

The proxy network is a pure P2P network. Therefore, it is highly fault-tolerant and scalable. Unlike traditional P2P systems, it does not include clients into the network itself in order to assure reliability and security.

FCAN resizes a scale of the proxy network depending on a load fluctuation adaptively in order to avoid troubles such as server down by massive access concentration. Figure 4 shows how the system works in the CDN mode.

When the server detects a coming of flash crowds, it forms a temporary proxy network as shown in the lefthand side of Figure 4. If the initial network cannot handle increasing an amount of accesses, the server recruits a new member proxy one by one as shown in the middle of Figure 4.

If the server cannot recruit temporary proxies any more, it degrades the quality of the video stream as shown in the righthand side of Figure 4. For example, in the situation that the

system provides video streams of two different qualities, high and low, it delivers the low quality content as substitute for the high quality one under this quality restriction. The network occupancy per client decreases so that the server can alleviate the load of whole delivery network.

If the proxy network can easily handle all the incoming loads, the server may lift the restriction of the stream quality at first. After the derestriction, it releases temporarily-recruited proxies one by one until all proxies are dismissed. Finally, the system all turns back to the normal condition.

4.3 Access Redirection

In our preceding studies, FCAN uses DNS-based redirection, i.e. the authoritative DNS server redirects an access to an appropriate node depending on the network structure. We use TENBIN [Shi00] for the authoritative DNS server. It is a high-performance DNS which allows server selection policies and DNS lookup entries to be changed dynamically. DNS-based redirection works transparently to users, however, we confirmed "cache effect" problem which is caused by some DNS servers somewhere in the world which make caches of the address resolution at their own discretion.

In this study, we make the client access redirect to an appropriate server through the *mediator*. The mediator works on the same machine as the client software and relays requests from the client to the servers. It handles client requests with the following procedure: (1) Receives a list of working servers from the origin server; (2) Receives the content from a certain server in the list, and provides it to the client; (3) When the mediator get a request from the client next time, it receives new list from a certain server; (4) Return to (2).

Using the mediator, we eliminate the cache effect problem, and even utilize geographical information-based redirection for example. While the mediator works non-transparently to users, we expect that the function of the mediator can be implemented in browser cookies in the future.

5 Preliminary Experiments

We conducted some preliminary experiments on a real network with a prototype of the system. Figure 5 shows an overview of the experiments.

In our experiments, we use some hosts in Saitama University and Kyushu Sangyo University for a server, proxies, a pseudo client, and a client node. We use Apple's stream segmenter (mediastreamsegmenter) for the segmenter, and QuickTime Player for QuickTime X in the client.

The pseudo client is to trigger the FCAN's functions against flash crowds. It submits requests for randomly chosen segments to the server following the pattern shown in the Figure 6. In the rest of this section, CP1, CP2 and CP3 are the proxies shown in the Figure

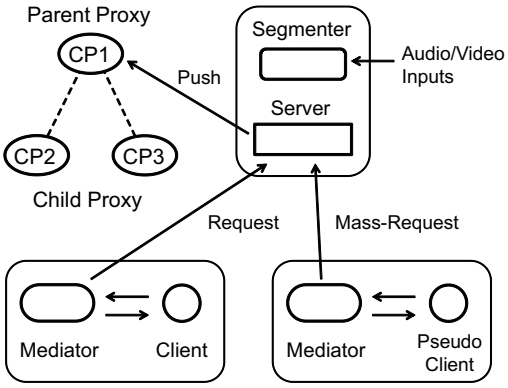


Figure 5: Experiment Environment

Table 1: Segments for On-Demand Experiment

Quality	Resolution	Average size	Duration
High	480×360	960 [KB]	10 [sec]
Low	320×240	410 [KB]	10 [sec]

5.

We made two experiments with two delivery methods, live and on-demand. In the on-demand streaming experiment, the server provides high and low quality contents with adaptive bitrates. We use segments shown in Table 1, which are samples of HTTP Live Streaming in the Apple Developer's site [AD11b]. The client software, the client has a master index file indicating these two quality contents, and the QuickTime Player requests segments of an adequate quality depending on load fluctuation following the master index. On the other hand, in the live streaming experiment, the server provides contents in a single quality in real time.

The server computes a load value regarding the size of requested segments. In the experiments, thresholds for load detection are defined beforehand based on some experiences. Workloads on the real Internet varies, and automatic and dynamic configuration of the thresholds is difficult. We suppose they may be configured based on the server capacity and the network bandwidth around the server.

Table 2 shows the time-line of the live experiment. We confirmed that the structure transition and dynamic resizing were performed depending on load fluctuation.

Table 3 shows the time-line of the on-demand experiment. In addition to the result of the live streaming, the stream quality was limited during server load growth.

Table 4 shows the sequence of the stream segments which the client played in the on-demand experiment. The client consistently requested the high quality contents until the

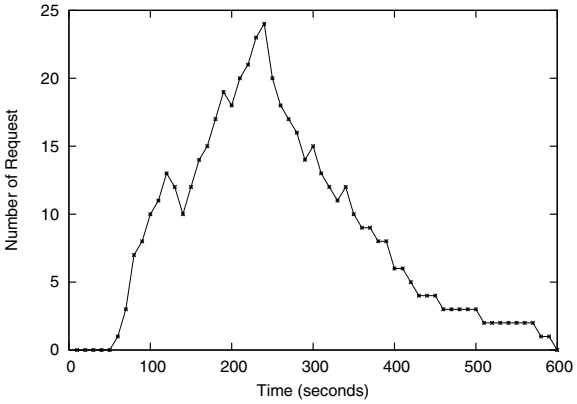


Figure 6: Request Pattern of Pseudo Client

Table 2: Time-line of Live Experiment

Time	Actions
0 [sec]	Start experiment Pseudo client request start
30 [sec]	Client request start
130 [sec]	Structure transition to CDN Recruit the core proxies (CP1, CP2)
220 [sec]	Recruit the additional proxy (CP3)
490 [sec]	Dismiss the additional proxy (CP3)
550 [sec]	Structure transition to C/S Dismiss the core proxies (CP1, CP2)
600 [sec]	End experiment

end of the experiment. In the on-demand streaming, the segmentation is done before the beginning of the experiment and the composition of the index does not change, therefore, clients received some number of segments before playback. After the number 27, the server degraded the stream quality for load alleviation, so the client received low quality segments as substitutes for the high quality ones. As the server load decreased, the server lifted the restrictions on the quality. After the number 45, the client herewith received high quality segments as required again. During the experiment, no malfunctioning, such as interrupt in the playback, was observed when the quality changed.

Figure 7 shows the load transitions of the server with FCAN and without FCAN in the on-demand experiment. The case of the server with FCAN shows the average loads of member nodes in the distribution network. The first peak at the 40th second shows that "buffering" was done when the client started the playback as mentioned above. The load on the server exceeded the higher threshold at the 130th second, then structure transition

Table 3: Time-line of On-Demand Experiment

Time	Actions
0 [sec]	Start experiment Pseudo client request start
30 [sec]	Client request start
130 [sec]	Structure transition to CDN Recruit the core proxies (CP1, CP2)
220 [sec]	Recruit the additional proxy (CP3)
250 [sec]	Quality degradation
430 [sec]	Quality improvement
490 [sec]	Dismiss the additional proxy (CP3)
550 [sec]	Structure transition to C/S Dismiss the core proxies (CP1, CP2)
600 [sec]	End experiment

Table 4: Playback Time-line of Client

Segment Name	Size	Quality
fileSequence0.ts	926 [KB]	High
fileSequence1.ts	946 [KB]	High
fileSequence2.ts	950 [KB]	High
...	...	...
fileSequence26.ts	958 [KB]	High
fileSequence27.ts	414 [KB]	Low
fileSequence28.ts	410 [KB]	Low
...	...	...
fileSequence44.ts	414 [KB]	Low
fileSequence45.ts	958 [KB]	High
fileSequence46.ts	958 [KB]	High
...	...	...
fileSequence60.ts	967 [KB]	High
fileSequence61.ts	958 [KB]	High
fileSequence62.ts	963 [KB]	High

to the CDN mode occurred so as to alleviate load concentration. However, the load on the member nodes continued to increase even after the transition, a new member proxy was recruited at the 220th second, and additionally the quality of segments was degraded at the 250th second, and consequently the server withstood the heavy load condition.

On the contrary, in the case of the server without FCAN, we observed that load values were far exceeding ones of the server with FCAN consistently. We, therefore, confirmed that FCAN's features against flash crowds were performed as a result of the increase of client

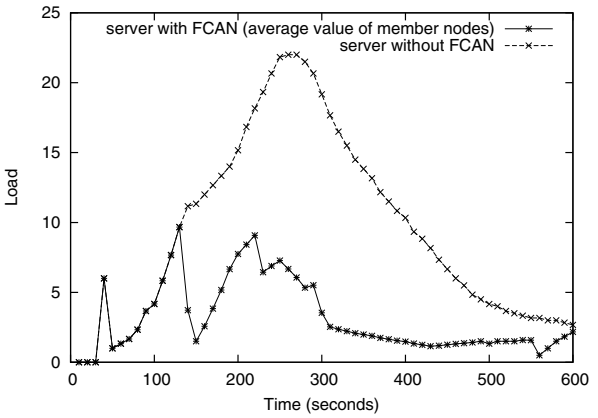


Figure 7: Comparison of load transitions in On-Demand Experiment

requests, and FCAN archived dynamic load balancing. We obtained equivalent results also in the live streaming experiment.

6 Conclusion

In our preceding studies, FCAN only focused on the static content delivery, however, flash crowds occur also in the video streaming. In order to alleviate flash crowds in the video streaming, both live and on-demand, FCAN adopts a new features such as *dynamic resizing* and *quality restriction* so as to raise a resilience of the system. In this paper, we proposed FCAN's extension for video streaming and demonstrated a prototype of the system on the real Internet. Through some experiments, we confirmed that FCAN's extension works effectively to alleviate flash crowds.

We are still at a starting point toward practical implementation and promotion of FCAN. Future research directions include: (1) quality guarantee in the CDN mode in the situation that multiple proxies deliver the same stream content, (2) appropriate thresholds assignments, and (3) implementation of dynamic access redirection using browser cookies.

References

[Abb11] O. Abboud, et. al. Enabling Resilient P2P Video Streaming: Survey and Analysis. *Multimedia Systems*, Vol:17, Springer, pp.177–197, 2011.

[AD11a] Apple Developer. HTTP Live Streaming. http://developer.apple.com/resources/http-streaming/, 2011.

[AD11b] Apple Developer. Bip Bop All. http://devimages.apple.com/iphone/samples/bipbopall.html, 2011.

[LE06] LIVE! ECLIPSE. http://www.live-eclipse.org/, 2006.

[Miy11] Y. Miyauchi, et. al. Preliminary Study on World-Wide Impremention of Adaptive Content Distribution Network. *Proc. Workshop on Self-Organising, Adaptive, Context-Sensitive Distributed Systems* , 11 pages, 2011.

[Pan06] C. Pan, et. al. FCAN: Flash Crowds Alleviation Network Using Adaptive P2P Overlay of Cache Proxies. *IEICE Tr. Comm.*, E89-B(4), pp.1119–1126, 2006.

[Pan11] R. Pantos. IETF Internet Draft: HTTP Live Streaming. http://tools.ietf.org/html/draft-pantos-http-live-streaming, 2011.

[Shi00] T. Shimokawa, et al. Flexible Server Selection Using DNS. *Proc. Int. Workshop on Internet 2000 (in IEEE-CS 20th Int. Conf. on Distributed Computing Systems)*, pp.A76–A81, 2000.

[Yos08] N. Yoshida. Dynamic CDN against Flash Crowds. *Content Delivery Networks* (R. Buyya, et al., eds.), Springer, pp.277–298, 2008.

An Anonymous Efficient Private Set Intersection Protocol for Wireless Sensor Networks

George Moldovan, Anda Ignat

Department of Design, Computer Science and Media
RheinMain University of Applied Sciences
Unter den Eichen 5
65195 Wiesbaden
george.moldovan@hs-rm.de
anda.ignat@hs-rm.de

Abstract: We present an efficient protocol which, under certain assumptions, provides a suitable level of security and anonymity in the ideal cipher model when computing the intersection of two private data-sets containing lists of elements from a large domain. The assumptions are that each node is pre-loaded with a set of pseudonyms, signed by the network's trusted authority; that the cardinality of each data-set is globally known. Our protocol first establishes a secure, trusted connection between two partners, then uses lightweight, symmetrical key operations for encoding and privately comparing the elements of two sets. Given a cryptographically secure symmetric encryption scheme, our protocol is safe for both semi-honest and malicious adversaries. The primary target platform for this protocol are Wireless Sensor Networks (WSNs), specifically those used in Ambient Assisted Living (AAL) scenarios, which almost entirely consist of a heterogeneous mix of devices, providers and manufacturers.

1 Introduction

Self-organizing WSNs consist of a multitude of small-scale devices with sensing properties and with wireless communication capabilities. The size and cost of such nodes makes them easy to be manufactured and distributed, but an inherent implication is the fact that they are resource restricted devices: low energy, low CPU processing power, small memory and storage space. WSNs are used to collect, process and distribute data from an environment. This makes them usable in projects related to military or surveillance tasks, vehicular networks communication, body and environment monitoring with AAL applications.

The AAL technologies are meant to help create and maintain a safe environment (both medical and socially) that addresses the needs of elderly or impaired people. The scenario is even more relevant considering that the current trend in Europe sees a rise in the percentage of elderly people [Ste05]. A cost-effective, unintrusive care, that would allow them to retain their autonomy is an appropriate, welcomed solution.

The use of WSNs in the AAL context implies that important security and privacy re-

quirements, that address both legal and personal needs, have to be fulfilled for them to be commercially and legally accepted (i.e. patience confidentiality needs maintaining). The public's acceptance, as they are social projects, is also a key aspect.

The WSN security is a broad research domain with specific challenges: privacy protocols and algorithms have to suit their hardware restrictions, but this has to be done without sacrificing the security level and without hindering other applications from performing their tasks, as the security should be a discrete, intrusive layer. Even more, in case of AAL projects, there are some additional, notable requirements:

- For certain AAL use case scenarios, the trade-off between performance and security needs to not surpass certain metrics, like the response time needed for a system to respond to the input of a user. Such interactions between the users and the system should be felt as instantaneous, i.e. users shouldn't have to wait in the seconds or minutes range if they want to remotely trigger a light-switch.
- Concerning groups of sensors and devices, an AAL scenario is implicitly heterogeneous, as several users and their devices might join or leave the network: visits from friends and technical or medical personnel.
- The anonymity of the user, of his processes and requests is another important aspect - a TV, light switch or energy measurement system should be oblivious of the identity of its users, but at the same time it should retain the ability to validate their authorization of using its services.

Problem Statement and Our Contribution Suppose node A is a data aggregator or end device (i.e. a display) that queries node B. Because of the multi-hop, ad hoc nature of the network, one could consider two relevant scenarios: (i) node A cannot reach B directly, and (ii) node A has to make sure (i.e. by checking a list of permissions) that it only accepts data readings from a node affiliated with the network. In the scenario (i), node B would have to initiate a route through trusted nodes only, while in the latter (ii) A would have to check whether B's permissions match its own access list. In both scenarios, this is ideally done without any of the involved parties revealing any additional information except a list of common group memberships in the data routing scenario, or the list of common permissions in the access control scenario. This is known as the *Private Set Intersection* [FNP04] problem .

The goal of this paper is to offer an efficient private set intersection algorithm, usable on resource restrained sensor nodes. We make the following assumptions: that there exists a trusted authority (TA), that the number of elements in each node's data-set is relatively small, and that there is a global domain from which the elements were extracted and which is large enough. It is beyond the scope of this paper to provide suggestions regarding changes to the sets on the nodes in the WSN after deployment, like the revocation of an element.

Returning to the previous AAL scenario, let node A and B each have k elements from a common domain whose size is n. These elements could stand for groups inside the

network or different access permissions. It is required to securely and privately compute the intersection of the two groups belonging to *A* and *B*. Given $A = \{a_1, ..., a_k\}$ and $B = \{b_1, ..., b_k\}$, compute $A \cap B$ without disclosing any other information about the identity of the other elements.

We propose a resource-efficient solution which preserves the anonymity of the involved nodes, the anonymity of the elements that are not part of the their sets intersection, and which uses lightweight cryptographic operations for its methods. Briefly described, the core of the protocol uses the value of each element in a set, it's signature generated by the overall TA, and a secure, shared session key. Both parties will encrypt each of their set elements by computing, from the element's corresponding signature and the shared session key, a one-time symmetric key.

The role of the signature associated to each element, and it's actual form, is to prevent the creation or unauthorized association of elements by a node. A detailed explanation is not in the scope of this paper and will be explained in a future publication

An important note is that, since there is not real need for decrypting the generated, encrypted values, the use of a message authentication code like HMAC [19802] or AES-CMAC [TIETF06] would yield an even faster protocol, with fewer resource requirements.

Paper Organization The rest of the paper is organized as follows. Section 2 briefly describes the related work and the general functionality of alternative solutions. Section 3 describes the cryptographic blocks used in the current form of the protocols. Section 4 contains the detailed description of our protocol, together with its prerequisites and explicit assumptions. Section 3.3 lists the behaviour of different possible malicious nodes; Section 5 offers an overview of future plans and changes to the protocol, security wise.

2 Related Work

Freedman, Nissim and Pinkas (FNP) proposed in [FNP04] a *Private Disjointness Test*, achieved by using a scheme that preserves the group homomorphism of addition, as well as of multiplication with a constant.

An additive homomorphic, public key cryposystem (Paillier's cryptosystem [Pai99]), allows a party to oblivious compute $E_{pk}(x) \cdot E_{pk}(y) = E_{pk}(x+y)$ or to compute $(E_{pk}(y))^x = E_{pk}(x \cdot y)$ being given only $E_{pk}(x)$ and $E_{pk}(y)$, where *x*, *y* are some plain-text messages and E_{pk} is the additive homomorphic function.

The basic structure of the FNP protocol consists of defining a polynomial *P* whose roots are the elements of private set $X = \{x_1, \ldots, x_n\}$ of size k_c:

$P(y) = (x_1 - y) \cdot (x_2 - y) \cdot \ldots \cdot (x_{k_c} - y) = \sum_{u=0}^{k_c} \alpha_u y^u$.

The α_u coefficients are then encrypted using a Paillier's cryptosystem and sent by the verifier *A* to the prover *B*, whose role is to evaluate the polynomial for each of his private elements. Each result is randomized by multiplying it with a non-zero constant *r*. An

optional final step is to also add the element to the randomized polynomial result. The resulting cipher-text will thus have the form $E_{pk}(r \cdot P(y))$ or $E_{pk}(r \cdot P(y) + y)$ for each element *y* of *B*. Once the verifier *A* receives the encryptions back from the *B*, he only needs to check if any of the decrypted values are zero, since any common element would be a root of *P*. If the latter encryption was used, *A* will need to check if the decryption matches an element from his own set.

Agrawal, Evfimievski and Srikant (AgES) proposed a different approach in [AES03], for information sharing accross private database. For it to work, a commutative encryption scheme is needed [Gam85], [MVO96]. Informally, a commutative encryption schemes uses to a pair of cryptographic function E_1 and E_2 with the property that $E_A(E_B(m) = E_B(E_A(m))$. If just $E_A(E_B(m))$ is known, neither *A* or *B* will be able to retrieve *m*.

AgES works as follows. Both partners *A* and *B* apply a hash on their private sets. Each then generates a secret key and encrypts each hashed element. The resulting sets of encryptions $\{E_A(H(x_1^A)), \ldots, E_A(H(x_n^A))\}$ and $\{E_B(H(x_1^B)), \ldots, E_B(H(x_k^B))\}$ are exchanged, where H is a cryptographic hash function, E is the encryption function using *A*'s and *B*'s secret keys, *n* and *k* are the cardinalities of the two sets. Each party then encrypts all the elements it received from the communication partner. In *A*'s case, this means computing $E_A(E_B(H(x_j^B))$, for all $1 \leq j \leq k$. For *B*, $E_B(E_A(H(x_i^A))$, for all $1 \leq i \leq n$. *A* then proceeds to send pairs of the form $P_j = \langle E_B(H(x_j^B), E_A(E_B(H(x_j^B))\rangle$ back to *B*. By intersecting the commutative encrypted set received from *A* with his own, *B* will get a subset of elements for which $E_A(E_B(H(x_i^B)) = E_B(E_A(H(x_j^A))$. x_j^B is then found by using P_j.

Both the FNP and the AgES protocols and their variants imply the use of public-key operations, whose number is directly proportional with the size of the elements in the groups of the involved parties. This is problematic especially for sensor networks.

3 Preliminaries

3.1 Bilinear Pairings

Let $(\mathbb{G}_1, +)$ and $(\mathbb{G}_2, \cdot)$ denote two cyclic groups of order *q* (some large prime), in which the Discrete Logarithm Problem (DLP) is considered to be hard.

An admissible bilinear pairing $\hat{e} : \mathbb{G}_1 \times \mathbb{G}_1 \rightarrow \mathbb{G}_2$ has the following properties:

1. *Bilinearity*: $\forall g_1, g_2, g_3 \in \mathbb{G}_1,\ \hat{e}(g_1 + g_2, g_3) = \hat{e}(g_1, g_3) \cdot \hat{e}(g_2, g_3)$ and $\hat{e}(g_1, g_2 + g_3) = \hat{e}(g_1, g_2) \cdot \hat{e}(g_1, g_3)$.
2. *Non-degeneracy*: $\exists g_1, g_2 \in \mathbb{G}_1 :\ \hat{e}(g_1, g_2) \neq 1$.
3. *Computability*: $\forall g_1, g_2 \in \mathbb{G}_1,\ \hat{e}(g_1, g_2)$ is efficiently computable.

In practice, the known implementations of these pairings - the Weil ([Jou00]) and the Tate pairings - prove that such constructions exist and involve fairly complex mathematics.

Typically, $\mathbb{G}_1$ is an elliptic-curve group and $\mathbb{G}_2$ is a finite field.

3.2 Identity-based Encryption

Following the definition given by Boneh and Franklin [BF03], an identity-based encryption scheme (IBE) is specified by the following four algorithms:

Setup Given a security parameter $k \in \mathbb{Z}^+$ as input, the algorithm returns the system parameters $params$ and the $master - key$.

Step 1: A randomized algorithm (known as a Bilinear Diffie-Hellman parameter generator), running on input k in polynomial time, generates: a prime q, the description of two groups $\mathbb{G}_1$ and $\mathbb{G}_2$ of order q, and the description of an admissible bilinear map $\hat{e} : \mathbb{G}_1 \times \mathbb{G}_1 \rightarrow \mathbb{G}_2$. A random generator $P \in \mathbb{G}_1$ is also chosen.

Step 2: A random $s \in \mathbb{Z}_q^*$ is picked and $P_{pub} = sP$ is set.

Step 3: The cryptographic hash functions $H_1 : \{0,1\}^* \rightarrow \mathbb{G}_1^*$ and $H_2 : \mathbb{G}_2 \rightarrow \{0,1\}^n$ for some n are chosen.

Then, the message space is defined as $\mathcal{M} = \{0,1\}^n$ and the cipher-text space as $\mathcal{C} = \mathbb{G}_1^* \times \{0,1\}^n$. The system parameters $params = \{q, \mathbb{G}_1, \mathbb{G}_2, \hat{e}, n, P, P_{pub}, H_1, H_2\}$ are published, and the $master - key$ $s \in \mathbb{Z}_q^*$ is kept secret.

Extract Using $params,\ master - key$ and an arbitrary $ID \in \{0,1\}^*$ (the public key) as input, a private key d_{ID} is returned by:

1. computing $Q_{ID} = H_1(ID) \in \mathbb{G}_1^*$, and
2. setting $d_{ID} = sQ_{ID},\ s$ being the $master - key$.

Encrypt With $params,\ ID$ and $M \in \mathcal{M}$ as input, the cipher-text $C \in \mathcal{C}$ is returned by doing the following:

1. compute $Q_{ID} = H_1(ID) \in \mathbb{G}_1^*$,
2. pick a random $r \in \mathbb{Z}_q^*$, and
3. set $C = \langle rP, M \oplus H_2(g_{ID}^r) \rangle$ where $g_{ID} = \hat{e}(Q_{ID}, P_{pub}) \in \mathbb{G}_2^*$.

Decrypt For $params,\ C = \langle U, V \rangle \in \mathcal{C}$ a cipher-text encrypted using the public key ID and a private key $d_{ID} \in \mathbb{G}_1^*$, $M = V \oplus H_2(\hat{e}(d_{ID}, U)) \in \mathcal{M}$ is returned.

To note is that the masks used during encryption and decryption are the same since:

$$\hat{e}(d_{ID}, U) = \hat{e}(sQ_{ID}, rP) = \hat{e}(Q_{ID}, P)^{sr} = \hat{e}(Q_{ID}, P_{pub})^r = g_{ID}^r$$

Each of these four algorithms must satisfy the standard consistency constraint: when d_{ID} is a private key generated by the *Extract* algorithm for a given ID as public key, then

$$\forall M \in \mathcal{M} : \ Decrypt(params, C, d_{ID}) = M \ where \ C = Encrypt(params, ID, M)$$

3.3 Adversary Models

The definition of the adversary models, as defined in [Gol96], follows.

The Semi-honest Adversary Model The *semi-honest* model of an adversary assumes that both parties act according to their defined action in a protocol, but they may store all intermediate results and computations, and use this to learn more than what they would know after working in an ideal model. In this case, only the initiator *A* of the protocol receives an input from the responder *B*. Thus, *A*'s anonymity means that the responder *B* is not able to determine if the sender has different inputs. *B*'s anonymity means that *A* will not get more information from its output than that defined by the protocol.

The Malicious Adversary Model A *malicious* model permits an adversary to not follow the protocol's description. In this case, this would mean the responder *B* not replying to *A*'s request, sending random generated elements or aborting the protocol at any point. Security in the malicious adversary model means not revealing more information than in an ideal model.

4 Protocol Description

The set for each entity that takes part in the protocol is assigned to it by a central secure and TA. The initialization of each party consists of the following steps:

4.1 Initialization Stage

Both the network owner (the TA) and each of the nodes have to perform an initial setup procedure. The two entities are described as follows:

Trusted Authority *Bilinear Maps Setup* The TA starts by generating the system parameters $\langle q, \mathbb{G}_1, \mathbb{G}_2, \hat{e}, n, P, P_{pub}, H_1, H_2 \rangle$ and retaining the private master-key g_{TA} as previously described in Section 3.2. Additionally, a cryptographically safe symmetric encryption scheme E is included into the system parameters.

Element List Setup The TA defines a large enough domain, from which the elements are randomly chosen. The TA also sets the global constant value k_c, which represents the maximal cardinality of each set loaded on a node.

Nodes *Pseudonyms Generation for Each Node* On every node, the TA deploys the public system parameters. It then generates and loads a set of pairs of the form $\{PS_i^{A_i}, SP_i^{A_i}\}$, where $PS_i^{A_i}$ is a collision-resistant pseudonym associated to node A_i, and $SP_i^{A_i} = g_{TA}H_1(PS_i^{A_i})$ is the *secret point* associated to it. [ZLLF06] and [BDS+03] have a more elaborate description of such a key agreement; since we have a similar implementation, the same notation is used.

Element List Loading The TA loads, on every node A_i, a set of pairs of the form $\{x_i^{A_i}, CS_i^{A_i}\}$, where $CS_i^{A_i}$ represents the commitment signature (CS) for the value $x_i^{A_i}$, with the property that for any two nodes, only identical elements will also have a common CS. Formally, for any two nodes A_i and A_j, $CS_i^{A_i} = CS_j^{A_j}$ iff $x_i^{A_i} = x_j^{A_j}$. The commitment signature has two roles: first, to prove that an element was created and distributed by the TA. Second, it is used to prove that a certain set of elements was created and assigned to a specific node by the TA.

Because the cardinality of each set is set to size k_c, the TA will also generate unique random values and corresponding CSs to fill all the sets to the requested size, in those cases when the representation of a node's functionality would require less than k_c different elements.

4.2 Message Exchange

There are two distinct phases (see Figure 1) needed by the protocol: the first stage creates *trust* between the two communicating parties. The second stage handles the private set intersection and elements obfuscation.

Network Affiliation Proof and Shared Key Creation The network affiliation proof and session key agreement both work by using an adapted IBE scheme, like the similar one used in [ZLLF06] and [BDS+03]: let there be two nodes, *A* and *B*. The messages exchange takes place as follows:

1. *A* initiates the protocol by selecting an unused pseudonym of his PS_i^A , together with a random generated number n_A, and sending both to *B*:
   $A \rightarrow B: \ PS_i^A, \ n_A$

2. Upon receiving the message, *B* generates a random number n_B, selects an unused pseudonym PS_j^B, and computes the key $K_{B \rightarrow A} = \hat{e}(H_1(PS_i^A), SP_j^B)$,
   which forms the value
   $V_{B \rightarrow A} = H_2(n_A \,||\, n_B \,||\, 0 \,||\, K_{B \rightarrow A})$.

The message is then:
$A \leftarrow B : \ PS_j^B, n_B, V_{B \rightarrow A}$

3. *A* generates his own key $K_{A \rightarrow B} = \hat{e}(H_1(PS_j^B), SP_i^A)$,
then checks if
$H_2(n_A \ || \ n_B \ || \ 0 \ || \ K_{A \rightarrow B}) \stackrel{?}{=} V_{B \rightarrow A}$.

4. The equation holds, according to Section 3.1, only if both nodes, *A* and *B*, had their pseudonym and corresponding secret key issued by the same TA, owner of g_{TA}:

$$\begin{aligned} K_{B \rightarrow A} &= \hat{e}(H_1(PS_i^A), SP_j^B) \\ &= \hat{e}(H_1(PS_i^A), g_{TA} H_1(PS_j^B)) \\ &= \hat{e}(g_{TA} H_1(PS_j^A), PS_i^B) \\ &= \hat{e}(SP_j^A), PS_i^B) \\ &= K_{A \rightarrow B} \end{aligned}$$

If the verification succeeds, *A* knows that *B* is under the same TA as him, and sends:
$V_{A \rightarrow B} = H_2(n_A \ || \ n_B \ || \ 1 \ || \ K_{A \rightarrow B})$.
$A \rightarrow B : \ V_{A \rightarrow B}$

5. *B* performs the same test as in the previous step and, if it succeeds, it will trust *A*.

Private Set Intersection As a result, not only did *A* and *B* authenticate themselves as *trusted* (in a certain degree) nodes, they also established a shared key $K_{A \rightarrow B} = K_{B \rightarrow A} = K$, that is used for establishing the private set intersection as follows:

1. Using the public symmetric encryption scheme *E*, *B* computes for every element of his set the following output: $E_{CK_j^B}(x_j^B)$, where $CK_j^B = CS_j^B \oplus K$ is the commitment key.
It then sends the resulting set $set(B)^K = \{E_{CK_j^B}(x_j^B), \ \forall x_j^B \in set(B)\}$ to *A*, ordered lexicographically.

2. Upon receiving $set(B)_s^K$, *A* creates its corresponding set $set(A)^K$ by computing: $E_{CK_i^A}(x_i^A)$, where $CK_i^A = CS_i^A \oplus K$ for each of its elements. The intersection of $set(A)^K$ and $set(B)^K$ will then contain only their common elements.

4.3 Security

The security requirements are stated for the private set intersection part of the protocol - regarding the anonymous authentication and key agreement stage, the proof is given by the fact that given the difficulty of solving DLP in $\mathbb{G}_1$, given any pair $\{PS_i^{A_i}, SP_i^{A_i}\}$, it is computationally unfeasible to deduce g_{TA}. Detailed proofs on bilinear maps can be found in [BF03].

Network Affiliation Proof and Shared Key Creation

1. $A \rightarrow B:\ PS_i^A,\ n_A$
2. $B: K_{B\rightarrow A} = \hat{e}(H_1(PS_i^A), SP_j^B)$
   $V_{B\rightarrow A} = H_2(n_A \mid\mid n_B \mid\mid 0 \mid\mid K_{B\rightarrow A})$
   $A \leftarrow B:\ PS_j^B, n_B, V_{B\rightarrow A}$
3. $A: K_{A\rightarrow B} = \hat{e}(H_1(PS_j^B), SP_i^A)$
   $H_2(n_A \mid\mid n_B \mid\mid 0 \mid\mid K_{A\rightarrow B}) \stackrel{?}{=} V_{B\rightarrow A}$
   $V_{A\rightarrow B} = H_2(n_A \mid\mid n_B \mid\mid 1 \mid\mid K_{A\rightarrow B})$
   $A \rightarrow B:\ V_{A\rightarrow B}$
4. $B: V_{A\rightarrow B} \stackrel{?}{=} H_2(n_A \mid\mid n_B \mid\mid 1 \mid\mid K_{B\rightarrow A})$

Private Set Intersection
A and B share the common key $K_{A\rightarrow B} = K_{B\rightarrow A} = K$
E is a symmetric encryption scheme

1. $B:\ set(B)^K = \{\forall x_j^B \in set(B),\ E_{CK_j^B}(x_j^B),\ with\ CK_j^B = CS_j^B \oplus K\}$
   $B \rightarrow A:\ set(B)^K$
2. $A: set(A)^K = \{\forall x_i^A \in set(A),\ E_{CK_i^A}(x_i^A),\ with\ CK_i^A = CS_i^A \oplus K\}$
   $set(A)^K \cap set(B)^K$ will contain A's and B's common elements.

Figure 1: Message Exchange

The Semi-honest Case The protocol satisfies the *correctness* requirement in the semi-honest model, as *A* receives an encryption of an element x_i as long as it is part of $A_s \cap B_s$, where A_s is the set of elements in *A* and B_s is the set of elements in B. For all the rest *A* receives, in the ideal cipher model, what looks like random values .

A's *privacy* requirement is automatically preserved, as it doesn't expose any of its data-set elements to *B* in the current scenario.

B's *privacy*, given an A^* that operates in an ideal model and given *A* that operates in a semi-honest model, means that their view of the protocol is indistinguishable. Informal, the case is represented by a curious adversary in the following example: by issuing a valid request to another node, the attacker receives the encrypted list of its elements, then exhaustively goes through the entire domain from which the values x and CSs were generated, an computes all possible results. If the domains are small enough, the attacker succeeds in finding all of the senders elements. As the assumption is that the TA uses a sufficiently large domain, the proof is direct and holds as long as the chosen *E* is a secure symmetric key scheme.

The Malicious-Adversaries Case The meaningful attack of a malicious node means trying generating random or carefully crafted elements instead of using the ones assigned by the certification authority. More specifically, in the current context, without knowing the signature key used to generate CS for a value x, it means having to find two correct random numbers s, t with the property that for a i, $(x_i, EC_i) = (s, t)$. The same proof as in the *semi-honest case* applies.

5 Conclusions and Future Work

We presented an resource efficient, secure private set interaction protocol, while assuming certain conditions, meant to be used in WSNs, more specifically in the context of AAL environments. The protocol also authenticates trusted nodes and performs all the communication anonymously (sender and receiver anonymity). It is secure against malicious attacks by using an efficient, symmetric encryption schemes where the actual value, as well as they key (commitment signature) are both unknown to a malicious attacker.

The protocol is not suitable for *disjointness tests*, as the cardinality of the intersection set, as well as the elements that form it, are known. We hope to offer a viable, WSN solution to this this problem, too.

A desirable feature is the ability of a party to detect the cases when the partner does not contain a valid set of elements prior to performing the private set intersection routine. By our definition, valid means assigned by the TA. This would prevent nodes from trying to assume a false list of elements or to request a list without having a certified one of their own. We will present our proposed solution in the near future.

References

[19802] Federal Information Processing Standards Publication 198. The Keyed-Hash Message Authentication Code (HMAC), 2002.

[AES03] Rakesh Agrawal, Alexandre Evfimievski, and Ramakrishnan Srikant. Information Sharing Across Private Databases. In *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data*, pages 86–97. ACM, 2003.

[BDS+03] Dirk Balfanz, Glenn Durfee, Narendar Shankar, Diana Smetters, Jessica Staddon, and Hao-chi Wong. Secret Handshakes from Pairing-based Key Agreements. In *IEEE Symposium on Security and Privacy*, pages 180–196. IEEE, 2003.

[BF03] Dan Boneh and Matt Franklin. Identity-Based Encryption from the Weil Pairing. In *SIAM J. of Computing*, pages 586–615, 2003. Extended abstract in Crypto'01.

[FNP04] Michael J. Freedman, Kobbi Nissim, and Benny Pinkas. Efficient Private Matching and Set Intersection. In *Advances in Cryptology*, EUROCRYPT '04, pages 1–19. Springer-Verlag, 2004.

[Gam85] Taher El Gamal. A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms. In *IEEE Transactions on Information Theory*, pages 469–472, 1985.

[Gol96] Oded Goldreich. Adaptively Secure Multi-party Computation. In *Proceedings of the Twenty-eighth Annual ACM Symposium on Theory of Computing*, pages 639–648. ACM, 1996.

[Jou00] Antoine Joux. A One Round Protocol for Tripartite Diffie-Hellman. In *Proceedings of the 4th International Symposium on Algorithmic Number Theory*, pages 385–394. Springer-Verlag, 2000.

[MVO96] Alfred J. Menezes, Scott A. Vanstone, and Paul C. Van Oorschot. *Handbook of Applied Cryptography*. CRC Press, Inc., Boca Raton, FL, USA, 1st edition, 1996.

[Pai99] Pascal Paillier. Public-key Cryptosystems based on Composite Degree Residuosity Classes. In *Proceedings of the 17th International Conference on Theory and Application of Cryptographic Techniques*, EUROCRYPT '99, pages 223–238. Springer-Verlag, 1999.

[Ste05] H. Steg. Ambient Assisted Living - European Overview Report, September 2005.

[TIETF06] RFC 4493 The Internet Engineering Task Force, IETF. The AES-CMAC Algorithm, 2006.

[ZLLF06] Yanchao Zhang, Wei Liu, Wenjing Lou, and Yuguang Fang. MASK: Anonymous On-Demand Routing in Mobile Ad Hoc Networks. In *Transactions on Wireless Communications*, pages 2376–2385. IEEE, 2006.

A Mobile Sink-initiated Proactive Routing Protocol for Deadline-Aware Data Aggregation Method in Energy-Efficient Wireless Sensor Networks

Tatsuya Abe, Yutaka Arakawa, Shigeaki Tagashira, Akira Fukuda

Graduate School/Faculty of Information Science and Electrical Engineering
Kyushu University
744, motooka, nishi-ku, Fukuoka-city, Fukuoka, Japan
819-0395
{abet, arakawa, shigeaki, fukuda}@f.ait.kyushu-u.ac.jp

Abstract: In this paper, we focus on monitoring environments with wireless sensor networks in which mobile sink nodes traverse sensing fields in a specific spatial-temporal manner and aggregate various types of environmental data with different deadline constraints distributed over sensor nodes. For such environments, we propose an energy-efficient data aggregation method that reduces intermediate transmission in multi-hop communication while guaranteeing predetermined deadlines. The basic approach of the proposed method is to temporarily gather (or buffer) the observed data into several sensor nodes around the moving path of the mobile sink that would meet their deadlines at the next visit. Then, the buffered data is transferred to the mobile sink node when it visits the buffering nodes. We also propose a mobile sink-initiated proactive routing protocol with low cost (MIPR-LC) that efficiently constructs routes to the buffering nodes on each sensor node. Moreover, we simulate the proposed aggregation method and routing protocol to show their effectiveness. Our results confirm that the MIPR-LC method can reduce energy consumption by up to 23% when compared with a simple routing protocol. In addition, the mobile sink nodes can gather almost all of the observed data within the deadline.

1 INTRODUCTION

Based on recent advancements in micro-electro-mechanical systems (MEMS) and wireless communication technologies, wireless sensor networks (WSNs) have emerged as a promising tool for monitoring environments in a wide range of applications [1]. A WSN is generally composed of sensor nodes for observing environment data and sink nodes for aggregating the data distributed over the sensor nodes. In WSNs, aggregation mechanisms are often very dependent on multi-hop communication, i.e., because of limited radio ranges of sensor nodes, data transmission between sensor-sink pairs are routed through several intermediate sensor nodes. An increase of such intermediate transmission leads to obvious power consumption concerns, especially in large-scale WSNs, since it is widely recognized that data transmission is responsible for a large part of the total power consumption in sensor nodes [2]. Thus, as a means to reduce intermediate transmission, a mobile sink

approach has attracted considerable attention over the last decade [3, 4, 5, 6, 7, 8].

In the mobile sink approach, a mobile sink node traverses a given sensing field and aggregates data observed in sensor nodes when it moves close to them. This approach can achieve an energy-efficient aggregation of data since it does not always use multi-hop communication, i.e., the mobile sink node can gather data directly from the sensor nodes without intermediate nodes. However, this approach increases the time delay that is required for the mobile sink node to visit the sensor nodes. To overcome the delayed aggregation, it is necessary for the sensor nodes to frequently use multi-hop communication in order to reach the mobile sink node, which also contributes to their large power consumption, as described above. Hence, the two objectives of realizing low power consumption and shortening the delay time appeared to be mutually exclusive during the aggregation of the observation data.

In this paper, we focus on monitoring environments with WSNs that require source-to-sink delay bounds according to the observation data. More specifically, in our system model, multiple mobile sink nodes exist to traverse a sensing field in a specific spatial-temporal manner and aggregate various kinds of environment data with different deadline constraints. For such environments, we propose an energy-efficient data aggregation method that reduces intermediate transmission in multi-hop communication while guaranteeing the delay bounds.

The basic approach of the proposed method is to temporarily gather (or buffer) the observed data into several sensor nodes that exist around the moving path of the mobile sink node. The buffered data is then transferred to the mobile sink node when it visits the buffering nodes. For these buffering nodes, the proposed method uses sensor nodes that would meet the deadline at the next visit of their mobile sink node. In addition, we also propose a mobile sink-initiated proactive routing protocol with low cost (MIPR-LC) that efficiently constructs routes to the buffering nodes on each sensor node, i.e., the routing table contains routes from the sensor node to the shortest buffering nodes for all the mobile sink nodes. Moreover, we evaluate the proposed aggregation method and routing protocol by performing simulation to show their effectiveness. As a result, we confirm that the MIPR-LC method can reduce energy consumption by up to 23% when compared with a simple routing protocol, and the mobile sink nodes can gather almost all of the observation data within the required deadline.

The paper is organized as follows. Section 2 describes the system model. Section 3 discusses related work. Section 4 presents the proposed aggregation method considering the data deadline and the energy-efficient routing protocols. Section 5 presents the simulation results. Section 6 concludes this paper.

2 SYSTEM MODEL

In this section, we show the system model used in this paper. In our model, multiple mobile sink nodes traverse a given sensing field in a certain pattern and gather various kinds of observation data with different deadline constraints. For example, typical applications

include environment monitoring systems in a farm, for which various data such as temperature, humidity, and sunlight are collected for the primary purpose of monitoring crop growth. In this model, we assume farmers and farm machines to be mobile sink nodes. Next, we explain the details of sensor nodes, mobile sink nodes, environmental data, and performance metrics.

In this model, many homogeneous sensor nodes are deployed in a sensing field. A sensor node is static and battery powered. In addition, a sensor node periodically generates observation data that are then stored into its own local buffer that is sufficiently large (in terms of capacity) to store data until the next visit of a mobile sink node. Furthermore, two sensor nodes can directly communicate if they are within each other's radio range. The wireless communication between sensor nodes is generally stable, although at times, sensor nodes may fail to receive packets owing to packet collision and radio noise.

In this model, existing mobile elements in a sensing field, such as farmers, farm machines, and so on, are diverted to mobile sink nodes, i.e., mobile sink nodes are uncontrollable and act with a specific spatial-temporal pattern. Furthermore, we define their patterns as periodic with a given period for each mobile sink node. Mobile sink nodes move while broadcasting beacons at fixed intervals and gather data from the beacon-received sensor nodes.

A sensor node measures different kinds of environmental data. These data are gathered by the mobile sink nodes within specific deadlines according to their type. Then, the mobile sink nodes immediately transmit the collected data to a control center over a mobile phone network such as 3G or WiMAX. In this paper, the delay time is the time that elapses from the instant a sensor node measures data to the instant at which a mobile sink node receives the data. For example, in the environment monitoring system, we consider that there are no problems even if the delay time of the aggregation is approximately one day. However, in the case of mechanical controls (e.g., the opening or closing of the windows of a greenhouse based on results of the gathered data), the deadline of the data must be set to approximately one hour.

Finally, we describe two performance metrics for aggregation methods in our system model:

Energy consumption: Energy consumption is an important performance metric in data aggregation. The network lifetime of the WSNs can be drastically extended by realizing an energy-efficient aggregation method. The energy consumption of a sensor node is mainly dependent on the number of data transmission, including intermediate transmission, which are required for the multi-hop communication and message propagation for routing construction. Thus, increasing the energy efficiency of a data aggregation method leads to fewer data transmission for the collection of data.

Delay time: Another performance metric is the delay time for data aggregation. The delay time occurs owing to the nature of the mobile sink approach. The delay time is dependent on the cycle period of a mobile sink node. There is no problem if the delay time for gathering the data is within the deadline.

3 RELATED WORK

Over the last decade, a number of approaches have been proposed for exploiting mobile sink nodes for data aggregation in WSNs. From the perspective of data aggreagation architecture, these approaches can be broadly classified into three types: mobile base station (MBS)-based approach, mobile data collector (MDC)-based approach, and rendezvous-based approach. In this section, we introduce the three approaches.

3.1 MBS-based and MDC-based approaches

In the MBS-based approach, a mobile sink node gathers observation data directly from sensor nodes using multi-hop communication. In [3], the authors address the problem of determining the sojourn times on the moving path for the mobile sink node using the linear programming (LP) method in order to maximize the network lifetime, i.e., to balance the energy consumption of all of the sensor nodes required for the intermediate transmission. In [4], the authors propose a two-tier data dissemination mechanism for large-scale WSNs in which multiple mobile sink nodes are deployed in the sensing field. With this approach, sensor nodes transmit data to the nearest mobile sink node. For this MBS-based approach, the delay time is short because the data are directly delivered from the sensor nodes to the mobile sink node. However, many sensor nodes require more frequent intermediate transmission for the multi-hop communication. In addition, with this approach, the sensor nodes must update the route information to the mobile sink nodes by frequently propagating control messages.

With the MDC-based approach, a sensor node stores data into its own local buffer and waits for a mobile sink node to arrive within its transmission range. When the mobile sink node arrives, the sensor node transmits the stored data to the mobile sink node in a single-hop communication. In [5], the mobile sink nodes randomly traverse the sensing field to gather the data from the sensor nodes. Moreover, to minimize the energy consumed by the entire network, the authors in [6] solve a path selection problem in delay-guaranteed sensor networks by exploiting path-constrained mobile sink nodes. With the MDC-based approach, the sensor nodes can transmit data to the mobile sink nodes without the multi-hop communication. However, the delay time increases because the sensor nodes need to store the data in local buffers until visited by the mobile sink node. In addition, it cannot gather data that have been generated by sensor nodes that do not have contact with any mobile sink node. Although a controllable mobile sink node may solve this problem, the installation of such a controllable node would incur additional costs.

3.2 Rendezvous-based approach

The rendezvous-based approach is a hybrid approach that combines the MBS-based and MDC-based approaches. This approach introduces several rendezvous points (nodes) for a

mobile sink node at which data are gathered from sensor nodes through multi-hop communication. These points then transmit the buffered data using the single-hop communication to the mobile sink node that visits them. In [7], the mobile sink nodes pass through predetermined anchor nodes (i.e., rendezvous points) while collecting data. To gather data at the anchor nodes, a tree structure-based aggregation method has been proposed. This method organizes a tree structure from a cluster node to its child sensor nodes using a routing protocol MintRoute [9]. The MintRoute protocol establishes the shortest route from the cluster node to each child. In addition, in [8], the authors assume that the mobile sink nodes can change their moving paths over time. Hence, they have proposed a data aggregation method that selects as the rendezvous nodes the sensor nodes that are to be frequently contacted by the mobile sink nodes. The rendezvous-based approach improves the energy efficiency relative to that of the MBS-based approach, in the sense that it reduces the number of intermediate transmission for multi-hop communication. Furthermore, it can also reduce the delay time relative to that of the MDC-based approach.

Our proposed approach can be considered to be a rendezvous-based approach. In this paper, we assume that observation data has a deadline time according to its type, and multiple mobile sink nodes with different cycle periods exist in the sensing field. For such a model, we need an energy-efficient aggregation method to satisfy the deadline for gathering the sensing data.

4 PROPOSED AGGREGATION METHOD

In this section, we propose an energy-efficient data aggregation method that reduces intermediate transmission in multi-hop communication while guaranteeing a maximum delay time for the observation data. Moreover, we propose the MIPR-LC protocol used in the proposed aggregation method to efficiently construct the routing paths for the aggregation on each sensor node.

4.1 Data aggregation

The basic approach of the proposed method is to buffer the observed data into several sensor nodes that exist around the moving path of the mobile sink node. The buffered data is then transferred to the mobile sink node when it visits the buffering nodes. The proposed method identifies sensor nodes that would meet the deadline at the next visit of their mobile sink node, and uses these sensor nodes as the buffering nodes. The identification is based on the predicted cycle periods recorded in its routing table. To reduce power consumption, it also uses the nearest buffering node out of all discovered ones. Then, the sensor node transfers the observation data to the shortest buffering node using the multi-hop communication. In this paper, the buffering nodes that exist around the moving path of a mobile sink are called mobile sink-path neighbor (MN) nodes. In addition, an MN node is said to be the shortest if it is the nearest one among all the MN nodes.

Fig. 1 shows an example of the proposed aggregation method. In this example, we assume that there are two mobile sink nodes: mobile sink node $M1$ with cycle period $T1$ and mobile sink node $M2$ with cycle period $T2$, where $T2 < T1$. Furthermore, a sensor node S transmits data with a $deadline$. In Fig. 1, if the $deadline$ is longer than $T1$, S transmits data to A. The destination node of the transmission is $D1$, which is reachable in a small number of hops ($H1$). If the $deadline$ is shorter than $T1$ and longer than $T2$, S transmits data to B bound for $D2$, which needs a large number of hops ($H2$).

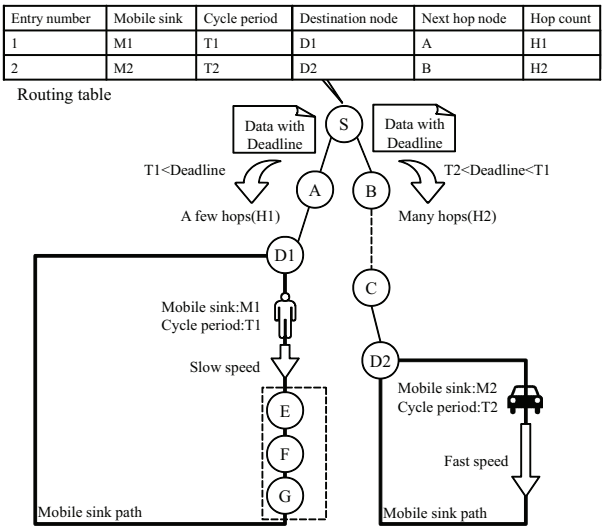


Figure 1: An overview of the proposed method

4.2 Route construction

To achieve the proposed aggregation method described in the previous section, each sensor node needs to construct routes to the shortest MN nodes for all mobile sink nodes. To construct routes, in this study, we propose two routing protocols: MIPR and MIPR-LC. The MIPR-LC method improves the MIPR method by reducing the routing cost required for constructing routing tables used in the MIPR method.

4.2.1 MIPR

The MIPR method is a proactive routing protocol that is initiated by a mobile sink node, i.e., the traversing mobile sink node periodically transmits trigger messages to one-hop neighbor sensor nodes (i.e., MN nodes). The received MN nodes broadcast control messages to the whole sensor network by employing flooding-based communication. More

detailed steps in the route construction are shown below.

1. A mobile sink node sends trigger messages to neighbor sensor nodes while traversing the given sensing field, i.e., the message can be received by the sensor nodes that exist within its single-hop communication range.
2. The received sensor nodes recognize themselves as the MN nodes and broadcast a control message to all the sensor nodes.
3. Upon receiving the control message, each sensor node constructs the route to the source node of the message using the distance vector algorithm.
4. Each sensor node can construct routes to all MN nodes because all MN nodes broadcast the control messages. The sensor node therefore selects the nearest MN node from all of the MN ones.
5. Each sensor node can construct routes to the shortest MN nodes for all of the mobile sink nodes because each mobile sink node transmits trigger messages while moving.

The proposed method adopts a mobile sink-initiated protocol in order to realize the high maintainability of routing tables and to quickly construct routing tables even in large-scale WSNs.

4.2.2 MIPR-LC

In the MIPR method, a routing table is constructed for each sensor node by employing flooding-based broadcasts. The broadcasts increase the energy consumption of the sensor nodes owing to repeated retransmission of control messages in the sensor nodes. The goal of the MIPR-LC method is to reduce these retransmission. In the MIPR method, all MN nodes broadcast control messages and the received sensor nodes retransmit the messages (on the left side in Fig. 2). However, in the proposed aggregation method, it is sufficient that each sensor node constructs routes to the shortest MN node. Therefore, in the MIPR-LC method, a sensor node retransmits the message only when the coming path length for the received control message is the shortest (on the right side in Fig. 2).

We now consider the trigger timing when a mobile sink node sends a trigger message. Fig. 3 shows the transition of the propagation region in the MIPR-LC method in chronological order. In this figure, sensor nodes A, B, and C are MN nodes for the same mobile sink node. A, B, and C receive a trigger message at $t = t_1$, $t = t_2$ and $t = t_3$, respectively. In this figure, sensor nodes that have retransmitted a control message are marked as p, while sensor nodes that have not retransmitted any message are marked as f. It can be seen that the propagation region changes depending on the sending order, i.e., the propagation region is the narrowest when C sends the last control message. Thus, we consider the trigger timing of control messages to minimize the propagation region.

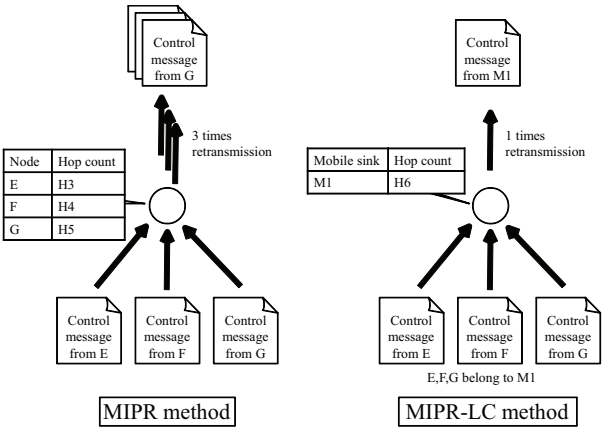


Figure 2: Propagation for control messages

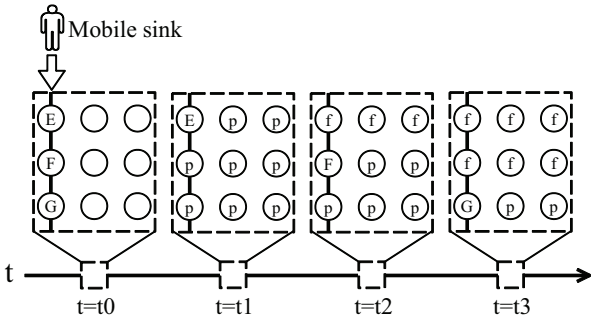


Figure 3: Propagation region

5 PERFORMANCE EVALUATION

In this section, we evaluate the proposed aggregation method by performing simulation. The routing protocol and aggregation method proposed in this paper are implemented on the network simulator platform QualNet version 5.0.1[10].

5.1 Routing protocol

First, we evaluate the average energy consumption for sensor nodes, i.e., the number of control message retransmission that is required to construct routes to the MN nodes that are the shortest of all mobile sink nodes. We describe our results and compare them to

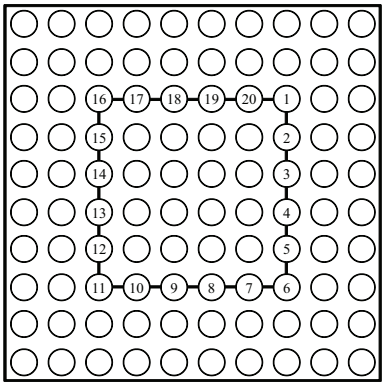


Figure 4: Simulation environment for evaluating routing protocols

Table 1: Evaluation results for the proposed routing protocols

	(1)			(2)
	(1-1)	(1-2)	(1-3)	
Transmission energy consumption[nJ]	0.58	0.43	0.36	2.52
Received energy consumption[nJ]	3.08	2.30	1.97	13.44
Total energy consumption[nJ]	3.66	2.73	2.33	15.96

those obtained using the MIPR-LC method (called (1) hereafter) and the MIPR method (called (2) hereafter). As shown in Fig. 4, in this simulation, 10×10 sensor nodes and 20 MN nodes numbered from 1 to 20 are regularly placed at intervals of 150 m. Each sensor node can communicate with its neighbor nodes. In addition, the transmission rate, transmission power, and received power for transmitting one control message are 1 Mbps, 100 mW, and 130 mW, respectively. The MN nodes transmit control messages once every second. Furthermore, in (1), we evaluate the impact of the change of the trigger timing on the results. More specifically, we implement the three trigger timings, (1-1), (1-2), and (1-3) and represent the control messages sent by each node (1, 2, 3, $\cdots$, 19, 20), as well as by all three nodes (1, 4, 7, $\cdots$, 15, 18) in a diagonal manner (1, 11, 6, 16, 2, 12, 7, 17, $\cdots$, 10, 20).

Table 1 shows the result. From the table, the energy consumption of (1) is 23% lower than that of (2). This is because the sensor nodes do not retransmit unnecessary control messages in (1), i.e., they retransmit control messages only when the next path length is shorter than previous path lengths, as described in Section 4.2.2. Furthermore, we confirm the effect of the trigger timing on the energy consumption. From the table, the timing (1-3) has the lowest energy consumption of all the timings. This is because in (1-3), the sensor nodes can transmit control messages in the most spatially distributed manner.

Table 2: Simulation parameters

Parameter	Meanings	Value
$N_{sensors}$	number of sensor nodes	200
$N_{mobilesinks}$	number of mobile sink nodes	2
$T_{mobilesinkA}$	cycle period of mobile sink node A	200[s]
$T_{mobilesinkB}$	cycle period of mobile sink node B	600[s]
λ	data arrival rate	12[packets/hour]
P	packet size	100[byte]
D_{dataA}	deadline of data A	300[s]
D_{dataB}	deadline of data B	700[s]

5.2 Aggregation method

Next, we evaluate the proposed aggregation methods. In this simulation, we measure the number of data that has been stored within the deadline and the energy consumption of the sensor nodes. We compare the results of three methods: our proposed data aggregation method (called (3) herafter), an MDC-based aggregation method that always uses the shortest mobile sink nodes regardless of the delay bound constraints (called (4) herafter), and an MDC-based aggregation method that always uses the fastest mobile sink nodes regardless of the energy consumption (called (5) herafter). In this simulation, the route for each sensor node is constructed by the MIPR-LC method.

The simulation parameters used in our experiments are summarized in Table 2. Furthermore, 10×20 sensor nodes are placed at fixed intervals of 150 m. Each sensor node can communicate with its neighbor nodes. In addition, the sensor nodes generate two sets of data (A, B) with different deadlines. The deadlines of the data are longer than the cycle periods for mobile sink nodes. In this simulation, two mobile sink nodes are positioned: mobile sink node A with a short cycle period is placed on the left side of the field, and mobile sink node B with a long cycle period is placed on the right side. The mobile sink node traverses the field at a constant speed. In addition, the transmission rate, transmission power, and received power for transmitting one control message are 1 Mbps, 100 mW, and 130 mW, respectively. The simulation time is one hour.

Table 3 shows the summarized results of the measurements. In this table, the data aggregation ratio is the number of data generated divided by the number of data aggregated, and the data aggregation ratio within the deadline is the number of data generated divided by the number of data aggregated within the deadline. All of the aggregation methods achieve aggregation ratios of over 94%. However, for (4), the data aggregation ratio within the deadline is about 80%. This is the worst performance of all the aggregation methods, since (4) does not consider the delay bounds. On the other hand, (3) and (5) achieve good performance. Moreover, the energy consumption of (3) is lower than that of (5). Thus, we conclude that the proposed method (3) improves the energy consumption of (5) while guaranteeing the deadline, as in (5).

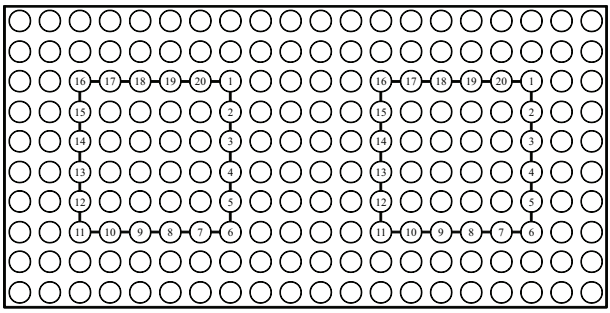


Figure 5: Simulation environment for evaluating aggregation methods

Table 3: Evaluation results for the proposed aggregation methods

	(3)	(4)	(5)
Number of data generated	2400	2400	2400
Number of data aggregated	2326	2257	2380
Data aggregation rate [%]	96.9	94.0	99.1
Number of data aggregated within the deadline	2326	1926	2380
Data aggregation rate within the deadline [%]	96.9	80.2	99.1
Total energy consumption in transmission [nJ]	191.8	113.5	280.0

6 CONCLUSIONS

In this paper, we proposed a data aggregation method that reduces intermediate transmission in multi-hop communication while guaranteeing a bounded delay. In our proposed approach, the observed data with deadlines are gathered for a set of MN nodes having a mobile sink node that can satisfy the deadline at the next visit. More specifically, each sensor node selects a mobile sink node that meets the deadline of the observed data, which is based on a prediction of arrival interval. The observed data is then transmitted to the shortest MN nodes that correspond to the selected mobile sink node. In addition, we propose a MIPR-LC protocol that is required for the proposed aggregation method that efficiently constructs a routing table on each sensor node. As a result, we confirm that the MIPR-LC method can reduce energy consumption by up to 23% when compared with a simple routing protocol, and the mobile sink nodes can also collect almost all observed data within the data deadline.

In future, we will study a data aggregation method that can reduce intermediate transmission even if the deadline is longer than the shortest cycle period of mobile sink nodes. In addition, we will examine in more detail the trigger timing for control messages. Furthermore, we will evaluate the proposed aggregation method and routing protocol in more generated situations.

ACKNOWLEDGMENT

The work described in this paper was partially supported by the Strategic Information and Communications R&D Promotion Programme (SCOPE) and by KAKENHI (22300025 and 22700076).

References

[1] I. Akyildiz,W. Su, Y. Sankarasubramniam, and E. Cayirci, "A Survey on Sensor Networks," *IEEE Communication Magazine*, No. 8, pp. 102–114, 2002.

[2] D. Estrin, A. Sayeed, and M. Srivastava, "Wireless Sensor Networks," Tutorial at the Eighth ACM International Conference on Mobile Computing and Networking, 2002.

[3] Z. M. Wang, S. Basagni, E. Melachrinoudis, and C. Petrioli, "Exploiting Sink Mobility for Maximizing Sensor Networks Lifetime," *Proceedings of the 38th Annual Hawaii International Conference on System Sciences*, pp. 287, 2005.

[4] H. Luo, F. Ye, J. Cheng, S. Lu, and L. Zhang, "TTDD: Two-Tier Data Dissemination in Large-Scale Wireless Sensor Networks," *Elsevier/ACM Wireless Networks*, Vol. 11, No. 1–2, pp. 161–175, 2005.

[5] S. Jain, R.C. Shah, S. Roy, and W. Brunette, "Exploiting Mobility for Energy Efficient Data Collection in Wireless Sensor Networks," *IEEE Workshop on Modeling and Optimization in Mobile Ad hoc and Wireless Networks*, Vol. 11, No. 3, pp. 327–339, 2006.

[6] S. Gao, and H. Zhang, "Energy Efficient Path-constrained Sink Navigation in Delay-guaranteed Wireless Sensor Networks," *Journal of Networks*, Vol. 5, No. 6, pp. 658–665, 2010.

[7] J. Luo, J. Panchard, M. Piorkowski, M. Grossglauser, and J. P. Hubaux, "MobiRoute: Routing towards a Mobile Sink for Improving Lifetime in Sensor Networks," *Proc. IEEE Int. Conf. on Distributed Conmputing in Sensor Systems*, pp. 10–11, 2006.

[8] Ayana Yamamoto, Shinya Kondo, Akimitsu Kanzaki, Takahiro Hara, and Shojiro Nishio, "On Selection of Data Forwarding Destinations Based on Status of Connection with Mobile Sinks in Wireless Sensor Networks," *DICOMO 2011*, Vol. 2011, No. 1, pp. 878–885, 2011.

[9] A. WOO, T. Tong, and D. Culler, "Taming the Underlying Challenges of Reliable Multihop Routing in Sensor Networks," *Proc. Int. Conf. on Embedded Networked Sensor Systems*, pp. 14–27, 2003.

[10] Scalable Network Technologies, Inc.: *QualNet Simulator.* http://www.scalablenetworks.com/.

Cellular Location Determination - Reliability and Trustworthiness of GSM Location Data

Richard M. Zahoransky, Klaus Rechert, Konrad Meier,
Dennis Wehrle, Dirk von Suchodoletz

Department of Computer Science
University of Freiburg
Hermann-Herder Str. 10
79104 Freiburg

Abstract: While using mobile telephony networks, the serving network infrastructure is able to determine the mobile station's location. Until now, cellular telephony has been built on self-contained infrastructure, i.e. all network components have been certified and especially users have been unable to take over control over their mobile equipment's behavior. With the rising awareness on privacy issues, software-based mobile phone network stacks became available and thereby a new freedom degree for mobile subscribers is introduced.

While slight modification to the mobile phones behavior will not impair with the general functionality of the network, cellular location determination becomes less reliable and trustworthy. We discuss user imposed measures to detect external location determination attempts and to obfuscate generated location information. With a dedicated testbed setup, the effects of location obfuscation were evaluated.

1 Introduction

Digital wireless telephony networks have become a core communication infrastructure within the past 15 years. GSM and its successors have significantly changed the communication landscape both in developed and, with only a slight delay, in developing market economies, by far outnumbering landline connections (e.g. in Germany [Ger10]). Mobile telephony and data are a crucial part of today's communication infrastructure; moreover, they can contribute to security and safety. The mobile telephony network and its physical characteristics help to locate mobile phone users in cases of emergency [1] and may be a valuable tool for search and rescue (SAR) [CLR10]. For instance, Bengtsson et al. analyzed post-disaster population's displacement using SIM-card movements in order to improve allocation of relief supplies [BLT+11]. Due to regulatory requirements but also driven by commercial opportunities, locating mobile phones gained the attention of research and industry. Furthermore, location information gathered through mobile telephony networks is now a standard tool for crime prosecution and it is enforced by the EC

[1] US Regulation on location determination in case of a emergency call: FCC Enhanced 911 Wireless Service, `http://www.fcc.gov/pshs/services/911-services/enhanced911`, [12/15/2011].

Data Retention Directive with the aim of reducing the risk of terror and organized crime [EP06]. Additionally, commercial services are based on the availability of live mobility patterns of large groups [2] or location-aware advertising [Kru10].

In general, law enforcement and commercial agencies exploiting location information have two options for utilizing location determination in mobile telephony networks: an active and a passive method. While active positioning yields immediate and more accurate results (e.g. through Uplink Time of Arrival [rGPPG02]), there are additional costs involved (e.g. network utilization) and thus, an incentive and dedicated target is required. This method is usually used to track identified individuals in criminal investigations. For instance, the police of North Rhine-Westphalia issued 225.784 location determinations on 2644 different subjects in 778 preliminary proceedings in 2010 [Min11]. Germany's federal police forces initiated 440.783 so called silent text messages. [3] On the other hand, with passive location determination techniques, all required information is generated from normal communication with the subscriber's mobile station, thereby causing no additional costs.

In this paper we investigate the user's possibilities to detect active location attempts and we lay out a scenario in which a user takes measures to provide a false position. Furthermore, measures for passive positioning methods are proposed which are capable to reduce location determination accuracy and potentially obfuscate position information in case of passive location monitoring. Finally, we evaluate and verify the location obfuscation method. For this purpose a test environment reflecting all components of a mobile telephony network was developed and deployed. The resulting mobile network infrastructure is based on real-life hardware and open-source software in order to create a realistic and defined environment which includes all aspects of the air interface in mobile telephony networks. The network is fully functional and thus provides a defined and fully controlled environment for analyzing all aspects of subscriber-provider interaction.

2 Localization Determination in Cellular Communication Networks

As an example we discuss the GSM infrastructure, because it is widely deployed and recently software and analysis tools have become available. Its successors UMTS (3G) and LTE (4G) still share most of its principal characteristics.

There is a variety of possibilities for determining a mobile station's location from the view point of the infrastructure, e.g., by Cell Origin with timing advance (TA) and Uplink Time Difference of Arrival (U-TDOA) for GSM [rGPPG09]. [4] While the latter method requires sophisticated network infrastructure, Cell Origin and TA are available in any network setup. However, both methods work without special requirements for the mo-

[2] Commercial traffic monitoring service, `http://www.vodafone.com/content/index/press/local_press_releases/germany/2008/tomtom_and_vodafone.html`, [12/15/2011].

[3] Letter of the Federal Ministry of the Interior by request of a parliamentarian, `http://www.andrej-hunko.de/start/downloads/doc_download/185-stille-sms-bei-bundesbehoerden`, [12/15/2011].

[4] For location determination options for UTRAN cf. [rGPPG10]

bile station and achieve a positioning accuracy of up to 50 m for U-TDOA in urban areas [SCGL05].

Another (non-standard) method to determine a mobile stations's (MS) location makes use of measurement results. Usually based on databases built from signal propagation models used during the planning phase of the infrastructure, this data can be used to create a look-up table for signal measurements to determine the MS's location. Based on the cell, TA and received signal strength of the serving cell as well as the six neighboring cells, Zimmermann et al. achieved positioning accuracy of below 80 m in 67% and 200 m in 95% in an urban scenario [ZBL+04]. With a similar method but more generic setup, Peschke et al. report a positioning accuracy of 124 m in 67% [HUP07].

While the mobile phone is in idle mode, network-assisted positioning is not possible. The network either has to wait for the next active period of the MS (e.g. phone call, location update) or has to initiate the MS's activity. This can be done by transmitting a so called *silent message* to the MS in order to force an active communication without raising the user's awareness. The procedure is used for instance by law enforcement authorities or by location-based services based on cellular positioning [Min11].

2.1 Interference by User Controlled Mobile Network Stack

With the development of *OsmocomBB*[5] GSM baseband implementation, the basic work has been done for a fully user controlled mobile phone. For instance, such a mobile station could be modified to log and expose the location data to its user that has been gathered by the mobile communication infrastructure [RMB+11].

2.1.1 Active Location Determination

A fully user controlled mobile device requires software interfaces with a network stack which controls and exposes signaling attempts (e.g. by detecting silent text messages). However, such a signaling attempt does not provide information on the purpose of paging the mobile station. Hence, it is difficult for a subscriber to decide whether the paging attempt is legitimate (i.e. incoming call or text message) or a (hidden) location determination attempt was triggered. Only after the device has reacted to the signaling the originator and the purpose of the paging becomes visible. However, by answering to the signaling, the mobile phone is getting active (i.e. sending network packages) and therefore a location measurement unit is able to determine the MS's position (e.g. through TDOA).

While active positioning requires a dedicated target and some costs, concealing the mobile station's location is also possible with some effort. Due to the usage of a full software network stack, lower network layers could be decoupled from the mobile phone. By leveraging a second communication channel, the user and his mobile station can be at a different place than the device running the physical layer and antenna, communicating directly with

[5] Open Source GSM Baseband implementation, `http://bb.osmocom.org`, [12/15/2011].

the mobile network infrastructure. This way the location of an individual SIM-card can be forged.

2.1.2 Passive Location Determination

In order to take over control on passive location monitoring, access to measurement results and the occurrence of location updates is required. Especially the density of periodic location samples makes a significant difference in the provider's possible knowledge on the user's movement pattern and thus on the user's present and future privacy risks. Such a monitor feature enables the users to select a mobile telephony provider that requests location updates less frequently or the user demands compensation for his or her loss in location privacy.

A second step to improve a user's location privacy is to reduce the observer's observation accuracy (obfuscation). A possible way to blur the exact location is to send empty or significantly altered measurement reports. Normally, the measurement reports include signal strength measurements of the surrounding BTS to support handover decision-making during active connections. Since a periodic location update requires only a very brief communication with the network, a handover between different cells is very unlikely. Thus, sending measurements of neighboring stations is technically not always required. By reducing the number of transmitted measurements the accuracy of the network's position estimation is significantly decreased. In the best case scenario (if no or false measurements were transmitted) the accuracy is decreased to the cell of origin combined with the timing advance parameter. To further decrease the accuracy of the estimated position, the MS may send with a slight timing offset. Such offsets have a direct impact on the timing advance calculation of the BTS. Consequently, this leads to an incorrect distance estimation between MS and BTS. It is also possible to report a wrong MS transmission power to the network. This influences any estimation the network draws based on the received signal strength of the MS.

The combination of manipulating measurement results, timing advance and reported transmission power makes it possible to conceal the actual position of the MS. Nevertheless, the rough location of the MS is still available through the coverage area of the serving BTS.

3 Evaluation Setup

To evaluate the proposed measures and their effects, a full mobile telephony network testbed is required. Different scenarios can be tested without interfering with the public network infrastructure.

The testbed consists of three basic components: the *Mobile Network*, the *Testbed Serving Mobile Location Center* (TB-SMLC) and the *Mobile Stations*. Figure 1 provides a schematic overview on the structure of the testbed. In combination, these components allow us to analyze all aspects of the communication between network and mobile station in

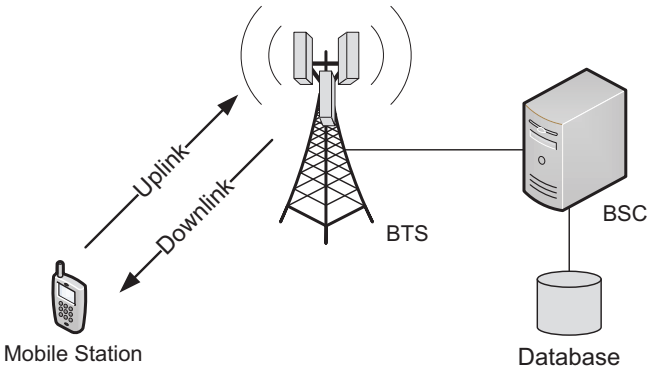


Figure 1: Overview of testbed network and transmission of measurement results.

a realistic scenario. Since the testbed implements a standard GSM network, it can simply be extended with standard GSM network components, for instance by the addition of any arbitrary mobile station. In contrast to a software simulation, such a setup allows for direct interaction with the network as a subscriber in order to get immediate feedback on status and events within the network. Since complete control over all components in the network is achieved, the subscriber's behavior as well as the impact on the infrastructure can be evaluated within the testbed. Special hardware and software is needed for the practical implementation of the testbed. A detailed description of hardware components used and the software implementation is given in earlier work [MWRv11].

3.1 Training Phase

A training phase is needed before localization can take place. For the training phase, a person equipped with a GPS receiver and a mobile phone was continually walking within the testbed's covered area. While walking, the MS was working in dedicated mode, continuously generating measurement reports that have been stored by the BSC, respectively the associated logging component. Each measurement report was assigned to its GPS coordinates. In a second step, the resolution of the measured coordinates is set. Measurements have been aggregated into tiles, with the size of the tile is the resolution of the map. However, the tile size can be set arbitrarily to a certain degree [ELM04]. An average is commuted among measurements with coordinates within the the same tile. Finally, outliners have been removed using Grubb's test [Gru69]. For our experiments the tile size of 8.52 m x 6 m have been chosen, which results in 6200 tiles for the covered testbed area. In total 171654 measurements were recorded and analyzed.

Measured received signal levels are considered as gaussian distributed. During an experiment with a stationary mobile phone 1500 MRs were created and analyzed. Results in Figure 2 show the histogram of the experiment and a fitted normal distribution. A standard

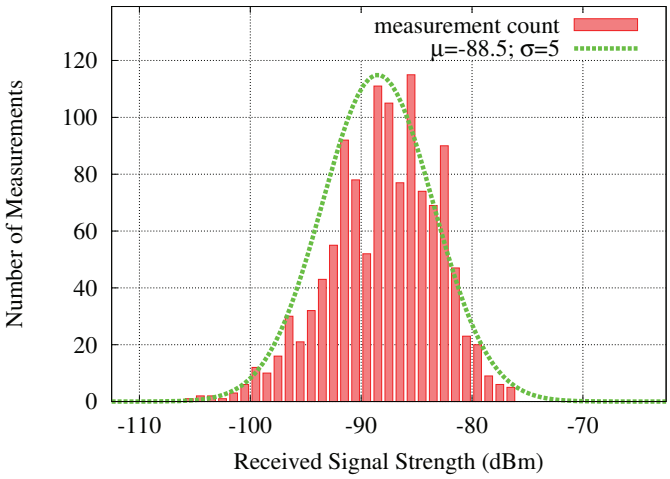


Figure 2: Histogram of observed received signal strength (RSS) of a stationary MS measuring a single BTS versus scaled Gaussian distribution with μ =-88.5 and σ =5.

deviation of $\sigma = 5$ dBm has been assumed throughout the localization calculation.

3.2 Interpolation

Since it is not feasible to take measurements for any place within the testbed coverage area, interpolation is used to approximate the RSS for places with no measurements. For this step, a Voronoi (or natural neighbor) interpolation was chosen because it shows the lowest error margin and results in a smooth interpolation (except at data points) [Suk01, LPA10]. The coordinate for which interpolation is required (denoted as point N) gets inserted into the Voronoi diagram. The resulting Voronoi region surrounding N "steals" some area from neighboring points. The stolen area size is expressed as fraction of the size of the Voronoi region of point N and treated as weighting factor. For every measurement point from which an area is stolen, the corresponding measurement is multiplied with the weighting factor. Hence, the interpolated value for point N is the sum of individual weighted measurements. Let a, b, c, d be the area size of the stolen areas by the Voronoi region of point N, with n denoting the size of N's Voronoi region, the interpolated signal strength for point N yields to $N_{RSS} = \frac{a}{n} \cdot A_{RSS} + \frac{b}{n} \cdot B_{RSS} + \frac{c}{n} \cdot C_{RSS} + \frac{d}{n} \cdot D_{RSS}$. An interpolated map is depicted in Figure 3.

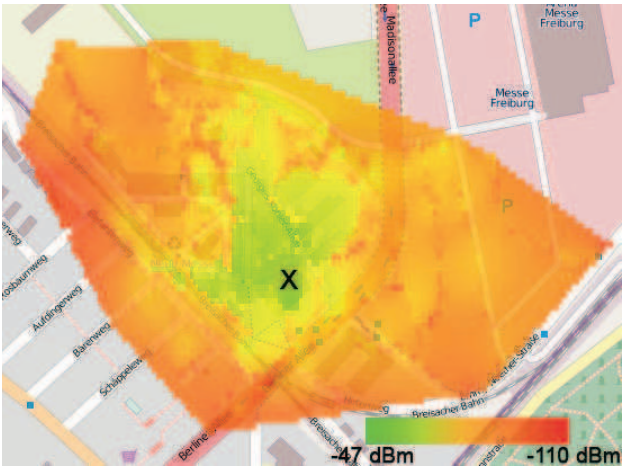


Figure 3: Interpolated GSM-map. The color denotes the receivable signal strength at that coordinate.

3.3 Location Determination

Localization calculation used is based on the Bayes' theorem. For every tile $l_{x,y}$ of the GSM map and for every receivable BTS BTS_i with $i \in 1...n$ and n denoting the number of receivable base stations, the available information is a vector of the received level of n base stations as $RSS := (BTS_1, \ldots, BTS_n)$ for the location (tile) of the MS. The probability distribution for the received signal strength RSS_i is estimated using every measurement observed within a tile: $P(RSS_i|l_{x,y}) = \mathcal{N}_i^{x,y}(\mu_i^{x,y}, \sigma_i^{x,y})$. We assume a Gaussian distribution of the signal strength measured in dBm. Figure 2 shows a histogram of observed RSS values of a stationary MS. Signal strengths from different BTSs are considered to be independent. The mean $\mu_i^{x,y}$ was already computed for every tile during training phase by averaging and outliners' removal. The variance $\sigma_i^{x,y}$ was chosen as 5 dBm, based on experiments (as shown in Figure 2).Usually, the network provides a list of the neighboring cells to the mobile phone to be monitored during an active connection in order to support a communication handover between two BTSs. In our testbed setup only one additional BTS is located on the campus, leading to limited localization possibilities. To cope with this shortcoming, we extended the neighbor list by adding additional public GSM cells receivable on the campus. By this, the mobile phones measure signal strengths of those other cells and sends the additional readings to the testbed network.

In order to locate a phone, the corresponding measurement entries are used from a prerecorded GSM signal map. An area based probability algorithm ($ABP - \alpha$) is used for location lookup [ELM04, YAUS03]. This group of algorithms return a set of the most likely map tiles, matching the actual and predetermined RSS fingerprints controlled by a confidence value α. The summed probability of the resulting set of tiles matches the required confidence value. Hence, the α-value controls the trade-off between positioning accuracy and methodical precision.

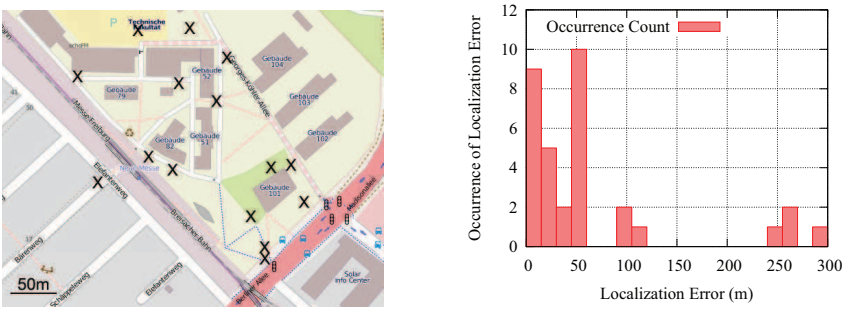
Given a received signal fingerprint vector (RSS), first we compute the probability at being at each tile's location $l_{x,y}$ using Bayes' equation

$$P(l_{x,y}|RSS) = \frac{P(RSS|l_{x,y}) \cdot P(l_{x,y})}{P(RSS)}, \tag{1}$$

with $P(RSS|l_{x,y})$ computed as multiplication over the probability distribution of BTS_i as

$$P(RSS|l_{x,y}) = \prod_{i \in 1...n} \int \mathcal{N}_i^{RSS} \cap \mathcal{N}_i^{x,y} dRSS\,,$$

and $\mathcal{N}_i^{RSS}$ as the derived Gaussian distribution of the MS's received signal strength of BTS_i.



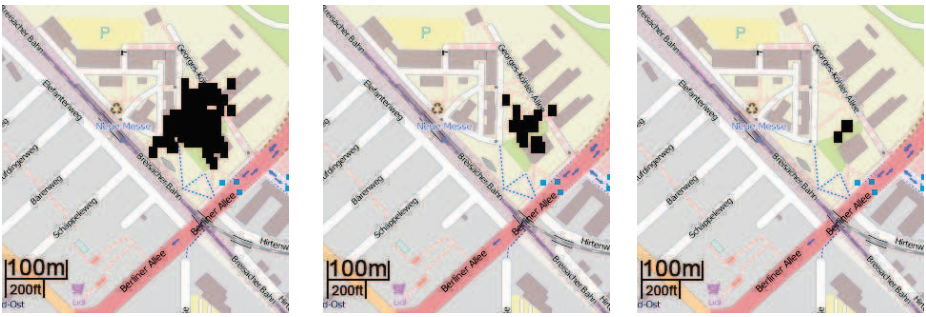
(a) (b)

Figure 4: Results of localization experiments with different mobile phones. (a) "X" mark the locations on which localization were carried out. At some places three phones were used for testing while on other places only a subset of the available phones were tested. (b) shows the resulting localization error versus its occurrence.

A priori $P(l_{x,y})$ is considered to be equally distributed. Its value is the reciprocal of the number of tiles within the map. The probability of the fingerprint vector RSS being measured within the GSM-map is calculated as

$$P(RSS) = \sum_{x \in X, y \in Y} P(RSS|l_{x,y}) \cdot P(l_{x,y})\,.$$

Equation 1 yields the probability of being at tile $l_{x,y}$ given the fingerprint vector RSS. Since we want to return an area with a given confidence-value α, the algorithm outputs the top probability locations $l_{x,y}$ until they sum up to α. For our purposes a dedicated LMU is not necessary since the required measurement reports are generated during normal operation.



(a) (b) (c)

Figure 5: Comparison of a localization results with different number of reported base stations. The marked areas depict possible whereabouts of the mobile station. Fig. (a) shows the localization with a set of two measurement, (b) shows a reduced set of four measurements and (c) a shows the result of a full set of six measurements.

3.4 Evaluation

Localization accuracy of was measured based on thirteen different locations and a set different mobile phones. In total, 33 experiments were done. A short phone call was made with every phone and location. The actual coordinates were derived from a GPS receiver. Localization error is considered as the distance between the GPS coordinate and the most probable calculated location by the ABP-α algorithm. As shown in Figure 4, the average position error is 67 m and a median error of 47 m. With this accuracy and the possibility to locate any phone call originated in the past it is feasible to extract movement patterns of the network's users.

3.4.1 Effectiveness of Location Obfuscation

From the user's perspective it is not possible to recognize if a phone call is or will be localized. With common normal phones, a user cannot influence the creation and data hold in a MR and is therefore incapable of regaining his location privacy.

With mobile phones running a user controlled GSM network stack it is possible to fabricate and send false MRs. A strategy to regain location privacy would be trying to decrease the obtainable localization accuracy. This goal can be achieved by sending only a subset of the measurements of surrounding BTSs. Uncertainty in the position calculation rises with With less information available for the ABP-α to process, the uncertainty in localization rises. The results are shown in Figure 5.

4 Conclusion

Until now, cellular telephony was built on self-contained infrastructure, i.e. all network components were certified and especially users were unable to take over control over their mobile equipments behavior. With the rising awareness of privacy issues, software based mobile phone network stacks became available and thereby a new freedom degree for mobile subscribers is introduced.

While slight modification on the mobile phone's behavior will not impair with the general functionality of the network. However, the network based location determination becomes less reliable and trustworthy. First, attempts of law enforcement agencies are observable by using an open and user controlled mobile station. Second, by modifying the mobile phone's behavior, reliability of location information is reduced to cell size or worse, since explicitly false positions may have been generated.

References

[BLT+11] Linus Bengtsson, Xin Lu, Anna Thorson, Richard Garfield, and Johan von Schreeb. Improved Response to Disasters and Outbreaks by Tracking Population Movements with Mobile Phone Network Data: A Post-Earthquake Geospatial Study in Haiti. *PLoS Med*, 8(8):e1001083, 08 2011.

[CLR10] Ling Chen, M. Loschonsky, and L.M. Reindl. Characterization of delay spread for mobile radio communications under collapsed buildings. In *IEEE 21st International Symposium on Personal Indoor and Mobile Radio Communications (PIMRC)*, pages 329–334, sept 2010.

[ELM04] E. Elnahrawy, Xiaoyan Li, and R.P. Martin. Using area-based presentations and metrics for localization systems in wireless LANs. In *Local Computer Networks, 2004. 29th Annual IEEE International Conference on*, pages 650 – 657, 2004.

[EP06] Council European Parliament. Directive 2006/24/EC of the European Parliament and of the Council of 15 March 2006 on the retention of data generated or processed in connection with the provision of publicly available electronic communications services or of public communications networks and amending Directive 2002/58/EC. *Official Journal of the European Union*, L 105:54 – 63, 2006.

[Ger10] German Federal Network Agency (Bundesnetzagentur). Jahresbericht 2010. `http://www.bundesnetzagentur.de/cae/servlet/contentblob/195950/publicationFile/10486/Jahresbericht2010pdf.pdf`, 2010.

[Gru69] Frank E. Grubbs. Procedures for Detecting Outlying Observations in Samples. *Technometrics*, 11(1):1–21, 1969. Available, `http://www.jstor.org/stable/1266761` [last Access 10/30/2011].

[HUP07] R. Haeb-Umbach and S. Peschke. A Novel Similarity Measure for Positioning Cellular Phones by a Comparison With a Database of Signal Power Levels. In *Vehicular Technology, IEEE Transactions on*, volume 56, pages 368 –372, Jan. 2007.

[Kru10] John Krumm. Ubiquitous Advertising: The Killer Application for the 21st Century. *IEEE Pervasive Computing*, 99(PrePrints), 2010.

[LPA10] J. P. Lewis, Frédéric Pighin, and Ken Anjyo. Scattered data interpolation and approximation for computer graphics. In *ACM SIGGRAPH ASIA 2010 Courses*, SA '10, pages 2:1–2:73, New York, NY, USA, 2010. ACM.

[Min11] Ministerium für Inneres und Kommunales NRW. Drucksache 15/3300 Funkzellenauswertung (FZA) und Versenden "Stiller SMS" zur Kriminalitätsbekämpfung. `http://www.landtag.nrw.de/portal/WWW/dokumentenarchiv/Dokument?Id=MMD15/3300(11/23/2011)`, 2011.

[MWRv11] Konrad Meier, Dennis Wehrle, Klaus Rechert, and Dirk von Suchodoletz. Testbed for mobile telephony networks. In *Resilience and IT-Risk in Social Infrastructures (RISI 2011)*, pages 661–666, 2011.

[rGPPG02] 3rd Generation Partnership Project (3GPP). TS 45.811 Technical Specification Group GSM/EDGE Radio Access Network; Feasibility Study on Uplink TDOA in GSM and GPRS (Release 6). `http://www.3gpp.org/FTP/Specs/html-info/45811.htm`, 06 2002.

[rGPPG09] 3rd Generation Partnership Project (3GPP). TS 43.059 Technical Specification Group GSM/EDGE Radio Access Network; Functional stage 2 description of Location Services (LCS) in GERAN (Release 9). `http://www.3gpp.org/ftp/Specs/html-info/43059.htm`, 11 2009.

[rGPPG10] 3rd Generation Partnership Project (3GPP). TS 25.305 Technical Specification Group Radio Access Network; Stage 2 functional specification of User Equipment (UE) positioning in UTRAN (Release 10). `http://www.3gpp.org/ftp/Specs/html-info/25305.htm`, 9 2010.

[RMB+11] Klaus Rechert, Konrad Meier, Greschbach Benjamin, Dennis Wehrle, and Dirk von Suchodoletz. Assessing Location Privacy in Mobile Communication Networks. In J. Zhou X. Lai and H. Li, editors, *ISC 11*, LNCS 2001, pages 309–324. Springer, Heidelberg, 2011.

[SCGL05] Guolin Sun, Jie Chen, Wei Guo, and K.J.R. Liu. Signal processing techniques in network-aided positioning: a survey of state-of-the-art positioning designs. *Signal Processing Magazine, IEEE*, 22(4):12 – 23, July 2005.

[Suk01] B. Semenov Sukumar, N. Moran. Natural neighbour galerkin methods. *International Journal for Numerical Methods In Engineering*, Volume 50; Part 1:1–28, 2001.

[YAUS03] M.A. Youssef, A. Agrawala, and A. Udaya Shankar. WLAN location determination via clustering and probability distributions. In *Pervasive Computing and Communications, 2003. (PerCom 2003). Proceedings of the First IEEE International Conference on*, pages 143 – 150, 2003.

[ZBL+04] D. Zimmermann, J. Baumann, A. Layh, F. Landstorfer, R. Hoppe, and G. Wolfle. Database correlation for positioning of mobile terminals in cellular networks using wave propagation models. In *Vehicular Technology Conference, 2004. VTC2004-Fall. 2004 IEEE 60th*, volume 7, pages 4682 – 4686 Vol. 7, sept 2004.

Physiological Data in Living Environments

Matthias Pfeiffer, Claudia Stockhausen, Katharina Reitz, Detlef Krömker

Graphische Datenverarbeitung
Goethe Universität Frankfurt
Robert-Mayer-Strasse 10
60325 Frankfurt

{pfeiffer, stockhausen, kreitz, kroemker}@gdv.cs.uni-frankfurt.de

Abstract: In this paper we develop three ideas for possible scenarios in future living environments. These ideas are based on technology which is present at the moment but may be deficient or not used in living environments. This paper focuses on showing possibilities, while knowing but neglecting actual practical problems. After introducing the potentials of physiological data, we present ideas for different scenarios. The first is concerned with games, trying to neglect the age differences of users and even up the difficulties in game-play. In the second scenario we show a concept for a supportive kitchen utilizing a brain computer interface and gesture detection. In the last scenario we present a draft for smart notification based on mental load. The ideas in this paper are not limited to elders or handicapped people, but most of them can be used to improve their lives.

1 Introduction

In future living environments many different users have to be addressed, when realising a user interface. In addition a new complexity of controlling all the devices in a living environment arises. Such a fusion of devices may lead to synergetic effects, but will also induce higher demands on the user. In the following we explore the potentials of using physiological data in the context of living environments. We explain several physiological measures and show how these can be used for different ways of interaction. This paper is structured as follows: After introducing the technology used and setting some fundamentals, we present different more or less futuristic scenarios based on actual technology. In the first scenario, differences in age and skills while playing computer games get neglected using in-game adaptations based on physiological data. In the second we show an automated kitchen and how table top, gesture detection and brain computer interface (BCI) technology can be combined and integrated. In the third scenario a concept for smart notifications based on the user state will be presented. Finally we draw conclusions implied by this paper.

2 Background

Different physiological signals of a user can be obtained. In our work we focus on cardiovascular activity, electrodermal activity and BCIs. In this section we present related methods and devices.

(a) Chest strap with radio transmitter

(b) ECG with 3 electrodes

(c) Portable EDA device with 2 electrodes

(d) The Emotiv EPOC headset (left rear view)

Figure 1: The different devices used in this work for measuring the physiological data.

2.1 Cardiovascular Activity

The cardiovascular activity of a user can be measured with an electrocardiogram (ECG). Several components of an ECG can be analyzed to determine the user state. The most common used are heart rate and heart rate variability.

2.1.1 Signals and Measurements

Heart Rate The heart rate (HR) is defined as the number of heart beats per minute. The HR can be affected by different factors like age, illness, physical training and breathing [Man08]. It can also be influenced intentionally by the user.

Heart Rate Variability To measure the heart rate variability (HRV), the time between two R-peaks in an ECG is measured. The variability of this measured time intervals is defined as the HRV. The HRV is one of the most used measures for determination of a user's mental load [SFM+01]. Several methods exist to conclude on this status [MBC+96, RSI98, Man08]).

2.1.2 Devices

To measure an ECG several devices exist; for example, stationary ECG devices, with 3 electrodes or more (figure 1b). The shown device needs a wired connection to a computer, for submitting data online. The 3 electrodes have to be attached to the chest of the user according to specific positions, depending on the chosen method of measurement. For the purpose of measuring HRV and HR, heart rate monitors, commonly used in sports, are applicable (figure 1a). An example setup, calculating the low frequency power spectrum, is described in [SK10]. In comparison to stationary devices, heart rate monitors used in sports are very comfortable to wear and allow wireless data transmission.

2.2 Electrodermal Activity

2.2.1 Signals and Measurements

Beside the cardiovascular measurements electrodermal activity (EDA) is one of the most commonly used physiological measures. The signal arises from electrical changes on the skin surface and can be split into tonic and phasic values [Bou88]. Tonic values are represented by the skin conductance level (SCL). The phasic component is called skin conductance response (SCR) and relates to certain stimuli. Every person shows a certain amount of non specific electrodermal reactions (NS.SCRs) per time. EDA is linearly correlated with a persons arousal [Lan95] and stands for emotional reactions and mental activity [Bou88].

2.2.2 Devices

The most common way of measuring EDA is by placing 2 electrodes on the palm. Devices can be stationary or portable (figure 1c). EDA measuring in general is unobtrusive. Recent work towards more comfortable devices led to the design of the Q-Sensor [PSP10]. This device is wearable at the wrist like a watch. Additionally the next generation will support wireless data transfer. This creates new opportunities for long term measurement in a living environment while people can follow their daily activities.

2.3 Electroencephalography

In this work we use electroencephalography (EEG) as base for a BCI. A BCI provides a direct way of communication between a user and a computer. In this case, direct communication means without the detour of using any muscular movement. This direct link is for some people (e.g. locked-in-patients) at this time the only possible way of communication with their environment [Moo03]. There are different approaches for obtaining signals from the brain. At the moment the EEG-based approach is the most promising, not only based on the fact, that it is the only portable one. A BCI can be used as an active and passive interface. This differentiation is based on how signals can be influenced by the user. An active signal needs a direct mental action produced by the user, such as focusing an object or thinking of something special. A passive signal can not be influenced directly, an example is the emotional state.

2.3.1 Signals and Measurements

P300 A P300 is an event related potential (ERP) and describes a wave of positive-going scalp-recorded brain potentials. In detail a P300 is a complex of waves and consists at least of a P3a and a P3b wave [Pol03, Pol07]. The first is best measured with frontal and central positioned electrodes and has its peak in a range of 250 to 280 ms after the stimulus. The second is best measured over parietal brain areas and has a task depending latency of 250 to 500 ms [CMC+09]. In case of this work, the P300 is a response to an infrequent, task-related stimulus, often evoked by using the oddball paradigm [SSH75]. The basic principle of an oddball paradigm is presenting a composition of high-probability non-target elements mixed with low-probability target elements. The latter will invoke the P300 response, which most likely can be detected after some repetitions.

Steady State Visually Evoked Potentials The steady state visually evoked potentials (SSVEPs) are the brain's response to a visual stimulation given at a specific frequency [WZGG04, CGGX02]. The response to this specific frequency is measurable as electric activity in the same or a harmonic frequency [BPS+03].

2.3.2 Devices

At the moment there are a variety of different electroencephalography (EEG) devices used for BCIs. A useful subdivision for this paper is probably a division in portable and not portable devices. All medical EEG devices are for stationary use and not portable while recording data. Medical EEGs also disqualify in terms of usability, because of their complicated set-up procedure, even if they have a better resolution. Non-medical BCIs are primary used for gaming and marketing research. These BCIs lack in accuracy, but can be used more easily. At the moment the Emotiv EPOC BCI (figure 1d) describes some sort of compromise. The Emotiv BCI can be mounted in a short time by a single user and is usable for more complex tasks, though some restrictions apply [PK10].

3 Possible Interactions

Physiological data leads to new ways of interaction. In this chapter we show several possible interactions in the context of living environments.

3.1 Raise/Drop Heart Rate

A possible interaction is raising and dropping of the HR. Raising the HR can for example be achieved by doing physical exercises. The raise of the HR could be used bound to a specific time interval, so that the user has to raise the HR to a certain level, holding it for a given time. Then the user has to calm down and relax, to achieve a dropping of the HR. To ensure that a user does a certain amount of sports, a possible interaction for, example for games or health applications, would be to raise the own HR to a certain level for a given time. On the other hand the interaction to drop the HR could be used for relaxation.

3.2 Adaptation to Mental Load

Based on the HRV the mental load can be used as a passive interaction. The HRV can not be influenced directly by the user. The mental load could be used as a part of the user state to indicate if a user is under mental stress. Depending on the level of mental stress, the user interface can adapt to this state. In contrast to many other possibilities of measuring the mental load, measuring the HRV does not require a direct user interaction.

3.3 Adaptation to Arousal

An increasing number of specific electrodermal reactions per time frame indicates a change in the user's arousal. As succeeding SCRs add up, the overall EDA signal is increasing if a user is exposed to continuous stimuli. While relaxing the signal drops back to a certain baseline.

3.4 Selection

When using an EEG-based BCI a selection mechanism can be implemented at least in two different ways. The first is utilizing the P300 waves and the oddball paradigm, the second is by using SSVEPs. If using the first way the principle is the same used in a P300-Speller [FD88, PK10]. The focused object gets selected by flashing empty spaces or spaces surrounding objects in contrast to flashing the surrounding space of the chosen object, which evokes a P300 response. This works for physical and virtual objects [YDTS10].

The approach based on SSVEPs highlights all objects of the selection at once, but uses different flashing frequencies for each highlight. The flashing frequency of the object focused by the user produces a measurable response in the brain. The given response consists of brain activity in the same or a harmonic frequency as used by the stimulus [BPS+03]. Both approaches can be used to implement a selection.

4 Possible Scenarios

In this chapter we introduce some scenarios based on physiological data. These scenarios are a selection based on our previous and recent work.

4.1 Adaptive Age Neglecting Games

People in living environments are of different age and have different knowledge and preferences regarding entertainment technologies. Therefore playing together or using game based learning, targets different audiences, sometimes even at the same time. Physiological measures can be used to cope with the differences. They provide input for creating more adequate and constantly modulating experiences, specifically tailored to the users needs and interests. The task is to detect and interpret the user state in order to create a supporting or challenging response of the system.

4.1.1 Adaptation to Player Status

The usual way of adapting a game to its player is by calculating the game performance based on game statistic values. According to this approach the game can react to the fact that a player is rather successful in achieving certain goals. However this method does not take into account the player's status. A high performance level can either be a result of constant stress (the player is nearly at his skill limits) or mental underload (the player is not challenged sufficiently). In contrast physiological feedback can adapt the game more specifically and can therefore be way more effective than performance feedback [RSL05]. Additionally the quality in game adaption through physiological feedback is independant of the amount and quality of statistic values available through the game context. Findings of our recent work [Rei11] show that the implementation of a smartphone game based on physiological data results in an individually adapted gameplay. A significant difference in EDA measurement was detected by comparing the adaptive game to a version without integrated physiological feedback. In case of the non adaptive version the plotted EDA value of each player tends to rise. The adaptive version shows a more flatened and in clonclusion adapted EDA curve.

4.1.2 Scenario

In this scenario, two people with different gaming experience and of different age play a multiplayer game on separate devices. One player has no experience with the game device or with gaming at all. The other player is an experienced player, enjoys playing games on different platforms and uses the device on a daily basis. Normally this leads to frustration on the side of the inexperienced player and boredom on the side of the gaming enthusiast. Therefore the game has to adapt to the divergent characteristics of the players. To achieve this, physiological parameters of both players are used as additional input modalities. Based on these measurements a feedback controller in the game can interpret and react to the current user state. The resulting reaction of the system is generated through visual, acoustical or haptic feedback. This feedback influences in turn the physiological reactions of both players. A so called closed feedback loop is generated [PBB95]. The game can support the inexperienced player while exposing the gaming enthusiast to greater challenges. It compensates the differences in learning curves and adapts to constantly changing demands.

4.2 A BCI extended On and Around the Surface-Tabletop Kitchen

At this point, we create a kitchen scenario, which is possible today. In this kitchen we combine elements from more than four different disciplines, which are normally separated. The main elements are borrowed from classical table top, gesture recognition, automated (self organized) storage and transport systems and BCI.

4.2.1 The Main Kitchen Elements and their Advanced Functionality

The Counter Top In this scenario the counter top is a large table top. We assume the counter top as a large display, where information can be shown on every free, uncovered spot. This can be realized through hidden projectors mounted under the kitchen cabins or by integration of displays into the counter top.

The Kitchen Cabinets The cabinets are much alike normal kitchen cabinets, except all cabinets are connected and have continuous shelves. Dividing walls on the inside have been removed. Somewhat special is an elevator system, which connects the shelves of the cabinets to the counter top. This elevator system can be realized as some sort of gripper system or as a traditional elevator system. The first is more complex, but can be useful for other purposes. The refrigerator should be constructed and integrated in the same manner.

The Storage and Transport Units Everything that is stored in the kitchen cabinets is stored in mobile storage units. These storage units consist of two elements. The upper element is a nearly normal kitchen storage container, maybe like the famous Tupperware containers. The bottom element is basically an autonomous transport unit. The main

function of this unit is to transport the storage unit on a given path through the system. To avoid getting stuck or collision problems, the whole unit should be as rounded as possible. As an extension, every transport unit is equipped with a kitchen scale to gather weight information. Each transport unit is connected to a central control unit, which organizes and controls the movement of the units and all other actions taking place in the kitchen.

The Central Control Unit All control, organisation and communication tasks concerning the kitchen are handled by the central control unit. This unit delegates as much work as possible to autonomous self-organized units, like the storage and transport units. Some tasks are handled directly by the central unit or are only supported by other units. An example for a central unit task is keeping track of the supplies, based on the kitchen scales data transmitted by the transport units and the best-before date gathered when supplies enter the system.

The User The most important part of the system is the user. This scenario can be seen as a support system for users with constrained movement, but it can also be seen as a system of comfort. However, the user in this scenario has a special feature, the user is wearing a portable (EEG-based) BCI.

4.2.2 User Perspective: Support for a Rather Normal Cooking Task

At first the user chooses a recipe. A selection of possible dishes will be presented by the central system at the nearest free space on the counter top near the user. The selection is based on the availability and weight information provided by the storage and transport units. After completing the selection, the chosen recipe, including a timeline, will be projected next to the hob. When the recipe is chosen the system starts arranging the needed supplies at the counter top. The directly needed supplies are arranged within reach around the hob, while the later needed supplies and possible alternatives are arranged within sight. If there are more than one eligible supplies within sight, a selection can be made based on a combination of on and around the table gesture recognition and a BCI. The selection is divided into two parts. In the first part a group of supplies is identified by a long distance pointing gesture [HLL+12]. In the second step the user picks the required unit using a BCI based selection. Depending on the circumstances a P300-based [YDTS10, PK10, FD88] selection or a SSVEP [WZGG04, CGGX02] approach can be used.

The selected unit will be moved within reach immediately. It is possible to distinct between a selection for usage, transporting it to the cooking area and just gathering some information about the ingredients without moving the transport and storage unit. This can be done by using a pull-gesture (gathering information) or a pick-gesture (selecting as ingredient) after completing the selection [HLL+12]. After usage, the storage units move out of direct reach, but stay within sight. When the cooking is finished the storage and transport units move back to their storage positions in the kitchen cabinets.

4.2.3 Central System Perspective: Hidden Background Tasks

The first task for the central system is to analyse the available supplies and present possible recipes based on this data. After a recipe is selected, the system has to coordinate the movement and the ordering of the transport and storage units from the storage spaces to the counter top. This can be done easily with reserved fairways and parking lots used as shunting routes. Storage systems of this kind can be found nowadays in pharmacies or warehouses. Depending on the realisation, the central system can pass some tasks to the autonomous or self organized transport and storage units. All the ingredients of the recipe, which are not for direct use, taken as possible alternatives or for spicing up, are placed within sight but not within range of the cooking area. The cleaning, especially after the cooking, can be supported by autonomous hoovering and wipe robots.

Special Cases and Side Effects In this scenario we concentrated on transport and storage units for ingredients, but they can also be used for dishes, cutlery, pots and pans. The system works similarly with respect to the weight and size of the object. A positive effect could be an automatic ordering system if some supplies are depleted or expired the best before date. It is also easily possible for the system to make order suggestions based on the actual offerings of the preferred supplier.

4.3 Smart Notification

Notifications can have a different level of importance or can occur in different situations. In some situations it might not be appropriate to notify the user about an unimportant message. In our scenario we describe how the state of the user for a notification system could be determined. Furthermore we will describe what ways of notification could be used in a living environment.

4.3.1 User State

Different possibilities for determination of the user state exist. As described in 3.2 the mental load can be determined by HRV and used for adaptations. Other possibilities to determine mental load are pupil dilatation [IZB04], questionnaires or performance measurements. Physiological measures have the advantage, that they are unobtrusive and do not need interaction by the user. Questionnaires have the disadvantage that they interrupt the user during the task. Performance measurements need interaction on a task [SFM00]. Today many HRV sensor devices are comfortable to wear, like the chest strap used in [SK10]. Most likely they will get even more comfortable in the future. For example it could be possible, that measures like temperature or HR are collected by a patch on the skin or a tiny chip under the skin. To improve the determination of user state, context information, e.g. if the user is moving, is needed. For this context information the sensors of mobile phones or a motion tracking of a person could be used.

4.3.2 Ways of Notification

In living environments notifications are not limited to one device. In our previous work [SSK11] we describe an email notification icon changing its highlighting and visual appearance. In this case the icon is in the system tray of a Windows desktop computer and HRV is measured. A first study was conducted and showed a correlation between time needed to recognize the icon and mental load based on HRV. Another realisation of a notification system based on the interruption level of the user was realized for mobile phones [CV04]. These examples and ideas can be carried on for living environments. In our following scenario, we describe how the notification of an important email or any other message could be done under different levels of mental load in a living environment. In this scenario we distinguish between high and low mental load.

High Mental Load Based on the position of the user in the living environment the way of notification could be chosen. With the help of person tracking or eye tracking, a notification could be displayed at the current position and it could be determined which way the user was looking. For example, instead of showing a notification icon for a very important email on the system tray of the desktop computer, the notification could be shown on the display of the refrigerator in the kitchen, if the user is in the kitchen. In the same way, the notification could be in the form of a voice notification, if the user has a high mental load, and would not recognize the notification in any other way or is not near any device that could display the notification. The intensity of the notification can be adapted to the level of mental load.

Low Mental Load When the mental load of the user is low, a simple notification is appropriate. Similar to the scenario for high mental load, the position of the user can be used to choose the device for displaying the notification. In contrast to the high mental load scenario a visualization with a low intensity should be sufficient.

5 Conclusion

Physiological data has great potential in living environments, even if there are some limitations. We proposed several possible interactions based on physiological data in the context of living environments and introduced different scenarios where they prove to be beneficial. Our previous work concerning physiological data in games [Rei11] and for notification [SSK11] support this statement. In the context of physiologically enhanced gaming it will be possible to compensate the differences of age and experience in gaming. This can open the field of gaming to new groups of users, which could not participate because of their lack of skills. As research in adaptive gaming evolves, physiological data as additional or even as main game control can help to integrate disabled people who are nowadays unable to use standard controls. We proposed a system for notifications which adapts to the user state, e.g. mental load. This addresses the heterogeneous user group in

living environments. Reaching the targeted user without distracting the surrounding users stays an open problem. By fusing physiological data and especially BCIs into the living environment the environment gets more sensitive to its user. Additionally, uncomfortable situations can get more comfortable, not only for handicapped persons. A current usability problem is the mounting of the measurement devices. We strongly believe, that the stand alone systems, as they currently exist, will not survive in future. Most of the nowadays separate systems will fuse into ubiquitous support systems. In this matter, important elements concerning the acceptance of such systems will be usability, especially a simple configuration and a trustworthy handling of sensible user data. This will result in better system usability and more capable computer systems which will result in enhanced quality of life.

References

[Bou88] W. Boucsein. *Elektrodermale Aktivität*. Springer, 1988.

[BPS+03] F. Beverina, G. Palmas, S. Silvoni, F. Piccione, and S. Giove. User adaptive BCIs: SSVEP and P300 based interfaces. *PsychNology Journal*, pages 331–354, 2003.

[CGGX02] M. Cheng, X. Gao, S. Gao, and D. Xu. Design and implementation of a brain-computer interface with high transfer rates. *Biomedical Engineering, IEEE Transactions on*, 49(10):1181–1186, 2002.

[CMC+09] A. Combaz, N. V. Manyakov, N. Chumerin, J. A. K. Suykens, and M. M. Hulle. Feature Extraction and Classification of EEG Signals for Rapid P300 Mind Spelling. In *Proceedings of the 2009 International Conference on Machine Learning and Applications*, ICMLA '09, pages 386–391, Washington, DC, USA, 2009. IEEE Computer Society.

[CV04] D. Chen and R. Vertegaal. Using mental load for managing interruptions in physiologically attentive user interfaces. In *CHI '04 extended abstracts on Human factors in computing systems*, pages 1513–1516. ACM, 2004.

[FD88] L. A. Farwell and E. Donchin. Talking off the top of your head: toward a mental prosthesis utilizing event-related brain potentials. *Electroencephalography and Clinical Neurophysiology*, 70(6):510–523, December 1988.

[HLL+12] N. Haubner, J. Luderschmidt, S. Lehmann, U. Schwanecke, and R. Dörner. An Unobtrusive System to Realize Interaction On and Around a Digital Surface. *(to be published)*, 2012.

[IZB04] S. T. Iqbal, X. S. Zheng, and B. P. Bailey. Task-evoked pupillary response to mental workload in human-computer interaction. In *CHI'04 extended abstracts on Human factors in computing systems*, pages 1477–1480. ACM, 2004.

[Lan95] P. J. Lang. The emotion probe. Studies of motivation and attention. *The American psychologist*, 50(5):372–385, May 1995.

[Man08] R. Mandryk. *Game Usability*, chapter Physiological Measures for Game Evaluation, pages 207–235. Morgan Kaufmann, 2008.

[MBC+96] M. Malik, J. T. Bigger, A. J. Cramm, R. E. Kleiger, A. Malliani, A. J. Moss, and P. J. Schwarz. Heart rate variability: Standards of measurement, physiological interpretation and clinical use. *European Heart Journal*, (3):354–381, 1996.

[Moo03] M. M. Moore. Real-world applications for brain-computer interface technology. *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, 11(2):162–165, 2003.

[PBB95] A. Pope, E. Bogart, and D. Bartolome. Biocybernetic system evaluates indices of operator engagement in automated task. *Biological Psychology*, 40(1-2):187–195, 1995.

[PK10] M. Pfeiffer and D. Krömker. The Emotiv EPOC BCI as Inexpensive Solution for the P300 Spelling Task - Pros and cons when using the Emotiv EPOC BCI. In R. Dörner and D. Krömker, editors, *Self Integrating Systems for Better Living Environments*, pages 29–35. Shaker Verlag, 2010.

[Pol03] J. Polich. *Detection of Change:Event-Related Potential and fMRI Findings*. Kluwer Academic Press, Boston, 2003.

[Pol07] J. Polich. Updating P300: an integrative theory of P3a and P3b. *Clinical neurophysiology*, 118(10):2128–2148, October 2007.

[PSP10] M.-Z. Poh, N. C. Swenson, and R. W. Picard. A wearable sensor for unobtrusive, long-term assessment of electrodermal activity. *IEEE transaction on bio-medical engineering*, (5):1243–1252, May 2010.

[Rei11] K. Reitz. Smartphone Games Based on Physiological Data, 2011.

[RSI98] D. W. Rowe, J. Sibert, and D. Irwin. Heart rate variability: indicator of user state as an aid to human-computer interaction. In *Proc. of the SIGCHI conf. on Human factors in computing systems*, pages 480–487. ACM Press/Addison-Wesley Publishing Co., 1998.

[RSL05] P. Rani, N. Sarkar, and C. Liu. Maintaining Optimal Challenge in Computer Games through Real-Time Physiological Feedback. *Pulse*, pages 1–7, 2005.

[SFM00] M. W. Scerbo, F. G. Freeman, and P. J. Mikulka. *Engineering Psychophysiology*, chapter A Biocybernetic System for Adaptive Automation, pages 241–253. Lawrence Erlbaum Associates, 2000.

[SFM+01] M. W. Scerbo, F. G. Freeman, P. J. Mikulka, R. Parasuraman, F. Di Nocero, and L. J. Prinzel III. The efficacy of psychophysiological measures for implementing adaptive technology. Technical report, NASA Langley, 2001.

[SK10] C. Stockhausen and D. Krömker. Measuring Mental Load with a Polar Heart Rate Monitor. In R. Dörner and D. Krömker, editors, *Proc. of the first Sensyble Workshop 2010*, pages 45–49. Shaker Verlag, 2010.

[SSH75] N. K. Squires, K. C. Squires, and S. A. Hillyard. Two varieties of long-latency positive waves evoked by unpredictable auditory stimuli in man. *Electroencephalography and Clinical Neurophysiology*, 38(4):387–401, 1975.

[SSK11] C. Stockhausen, D. Schiffner, and D. Krömker. Adaptation of User Interfaces based on Physiological Data. In *Reflexionen und Visionen der Mensch-Maschine-Interaktion - Aus der Vergangenheit lernen, Zukunft gestalten 9. Berliner Werkstatt Mensch-Maschine-Systeme*, pages 78–79. VDI Verlag, 2011.

[WZGG04] Y. Wang, Z. Zhang, X. Gao, and S. Gao. Lead selection for SSVEP-based brain-computer interface. *Engineering in Medicine and Biology Society, 2004. IEMBS '04. 26th Annual International Conference of the IEEE*, 2:4507–4510, 2004.

[YDTS10] B. F. Yuksel, M. Donnerer, J. Tompkin, and A. Steed. A Novel Brain-Computer Interface Using a Multi-Touch Surface. In *CHI '10: 28th International Conference on Human Factors in Computing Systems*, pages 855–858, New York, NY, USA, April 2010. ACM.

Goal-Snapping: An Empirical Evaluation of Object Snapping in Tangible and Multi-Touch Interfaces

Sebastian Schmitt, Johannes Luderschmidt, Nadia Haubner, Simon Lehmann
Ralf Dörner, Ulrich Schwanecke

RheinMain University of Applied Sciences
Department of Design, Computer Science & Media
Unter den Eichen 5
65195 Wiesbaden
Germany
mail@s-schmitt.de, {johannes.luderschmidt, nadia.haubner, simon.lehmann,
ralf.doerner, ulrich.schwanecke}@hs-rm.de

Abstract: We present "Goal-snapping", a novel approach for applying snapping techniques to tangible and multi-touch interfaces. It can be used to support users in accomplishing basic tasks such as aligning, sorting or grouping of virtual objects. As using snapping on large surfaces poses challenges in interaction design, we identify and discuss according parameters in Goal-snapping. For sorting and aligning, we propose to use snappers that attract objects within a target zone and visually arrange them to present an overview. For exchanging objects among users, we propose that each user has a target snapper that acts as a goal to which objects can be flicked. A user study has shown that although participants embrace the use of snapping to automatically group objects in a sorting task, snapping does not accelerate the completion time and increases the error rate by accidently snapped objects. In a long distance positioning task, the use of snapping significantly increases task completion.

1 Introduction

Tangible user interfaces (TUI) make use of real-world objects to interlink between a virtual and the physical world by allowing to detect physical artifacts (so-called props). TUIs can utilize a person's interaction capabilities with real-world objects in order to manipulate digital information. Multi-touch interfaces allow to detect multiple simultaneous touches on a display. This enables to use specific multi-finger gestures and interaction techniques [LPS+06]. The combination of TUIs and multi-touch interfaces constitute the field of hybrid surfaces (for instance, in [TKR+08]).

Hybrid surfaces have a large potential for being employed in future living environments as they are able to seamlessly enrich peoples surroundings with ICT. For instance, elderly people can use this technology in order to communicate and stay in contact with family members as these user interfaces allow for natural communication and sharing of artefacts in remote collaboration ([MHPW06] gives an example). Families can gather around an

interactive surface integrated in a table in order to manipulate documents such as photos or play games ([SCH+06] provides an example). As these user interfaces possess the potential to make interactions more intuitive, more natural and easier to grasp, hybrid surfaces can provide in particular an easy access to complex information systems, their configuration and administration. Users could, for example, facilitate these interfaces to specify the security policies for their private IT-sphere through sorting priorities that are visually represented on a hybrid surface.

Sorting or grouping are basic tasks that need to be supported on such hybrid surfaces. Since these surfaces might become large or could be operated from a distance, an additional basic task will be long-range positioning that enables users to comfortably use the complete surface for sorting and grouping. In this paper, we introduce a new user interface (UI) technique we call Goal-snapping that can be applied with hybrid surfaces and provides a novel realization alternative for the three basic tasks mentioned. It contributes not only to the easy and safe usage of the private communication and information infrastructure; it mirrors also the abilities of self-organizing and self-adapting IT-systems as it supports users in organization and adaptation tasks.

Goal-snapping builds on the commonly used method of snapping. Snapping refers to different techniques in various application fields. In general, it has been introduced in [BS86] as application-based assistance to place and align virtual objects or cursors on a display and has since then been thoroughly investigated for mouse and keyboard-based user interfaces. Goal-snapping employs snapping in target zones that automatically attract virtual objects that remain within the boundaries of that zone.

We show how to employ Goal-snapping to assist users in sorting and grouping tasks on interactive surfaces. Target zones can attract objects that have been dragged or flicked to it and arrange them clearly in piles in order to provide an overview of the documents. We also show how to use Goal-snapping to facilitate long range positioning tasks. If several users work together with shared documents on the same surface, users might want to exchange documents with each other. Due to the size of larger interactive surfaces, users may not be able to reach all regions from their location. Thus, for instance, each user can have their own Goal-snapper to which other users can flick documents.

This paper contributes an in-depth consideration and evaluation of design and interaction issues of Goal-snapping. For evaluation purposes, we conducted a user study with 20 participants allowing us to give qualified statements about the impact of alternative design aspects on Goal-snapping.

2 Related Work

In the area of multi-touch and tangible interface computing, snapping can be used for different purposes. [NBBW09] empirically evaluates interaction techniques for translating, rotating and scaling single objects on a multi-touch surface. Here, snapping is used for a more accurate alignment of objects. Hancock et al. also present techniques for the rotation and translation of virtual objects on a tabletop system [HCV+06]. They discuss the role

of snapping and state that snapping could simplify the alignment of objects to one another. Still, they only discuss the possibility and suggest that it could be used in systems that employ any of their techniques. [SCH+06] introduces a technique to move virtual objects to related groups. Here, snapping is realized in combination with arrows that appear close to objects that can be added to a group. When tapping an arrow, the related object is moved to the according group. This concept is based on existing textual metadata, describing the photos and videos. Therefore, it assists the users by offering only certain target groups for an object. However, it does not support grouping and sorting tasks where no prior metadata has been defined. In [AB06] Agarawala et al. describe the use of snapping in a virtual desktop environment that can be manipulated by pen-based interaction. Objects on the virtual desktop can be grouped together by tossing them towards piles. Still, they neither describe their snapping approach in further detail nor do they evaluate it empirically. [RGS+06] introduces flicking as a long-distance positioning technique for a pen-based interactive surface. However, they do not investigate how flicking can be employed on multi-touch surfaces in combination with snapping.

3 Goal-Snapping

We now introduce the properties of our Goal-snapping concept. First, we give a definition of Goal-snapping and introduce the related terminology. In section 3.2, we discuss various design issues for a concrete realization of Goal-snapping.

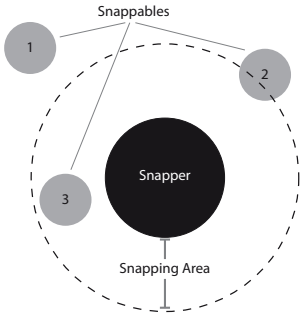


Figure 1: Basic Goal-snapping concept including a snapper object with it's related snapping-area and snappable objects which can be snapped torwards the Snapper

3.1 Basic Concept

For the basic concept of Goal-snapping we regard two different kinds of UI elements, namely "snapper" and "snappable". Figure 1 illustrates this concept: Similar to the functionality of a magnet that attracts a piece of metal, a snapper attracts snappable objects.

If a snappable object comes within range of the "snapping area", the snappable is being snapped by the related snapper. An essential aspect of Goal-snapping is that an object is either snapped by a snapper or not at all. This means that once a snapping is detected there is no way to interrupt this process. Furthermore, the snapping scope is locally restricted to the snapping area only. In the scenario in Figure 1, snappables 2 and 3 would be snapped by the snapper while snappable 1 would not be snapped.

3.2 Design Issues of Goal-Snapping

Our basic definition makes neither suggestions about how a snapper object is represented on a hybrid surface nor what further properties it may have. As a result, the introduced concept for Goal-snapping leaves space for various design decisions when considering a concrete realization. In this section, we identify several design parameters that have to be considered when implementing Goal-snapping and give design advices where appropriate.

3.2.1 Snapper Design

The snapper itself can be represented graphically or by a physical object. For instance, the snapper in Figure 1 is a graphical object that has a circular shape. Alternatively, a physical prop of an arbitrary shape (for instance, in the shape of a magnet) could be employed. In order to create snappers, there are several possible approaches. For instance, they could be created automatically when an application starts or manually by a user. A user could perform a gesture to create a snapper under their fingers. If a prop is employed, it can be simply put on top of the interactive surface to activate its snapping behavior. After the creation of a snapper, it could also be possible to move a snapper around or to change its size.

3.2.2 Designing the Snapping Area

Similar to the snapper, the snapping area can have different properties regarding size, shape and behavior. As a snapper resembles a magnet that attracts virtual objects, it is recommendable to use a circular snapping area to represent the snapper's magnetic field that usually has spherical characteristics. The snapping area size defines the strength of the snapper's magnetic field. Hence, the snapping area needs to exceed the snapper's boundaries. At runtime a user could be allowed to enlarge or shrink the area. The snapping area needs not necessarily to be visualized. Basically, a visible snapping area could be useful as to ascertain users when they have moved an object into the boundaries. Alternatively, snapping areas could only be displayed at a certain moment. For instance, if a snappable approaches a snapper, the according area could be faded in. Furthermore, the area can be visualized with different aspects concerning alpha, color or texture. If an object enters overlapping snapping areas, the conflict to which snapper it will be moved needs to be settled. Normally, the object should be attracted to the snapper to which center it is nearer or which has the stronger "magnetic field". Alternatively, snappers with overlapping areas

could reject each other like they had the same magnetic poles in order to prevent areas from overlapping. However, this will be rather complicated to establish if the snappers are physical props.

3.2.3 Designing the Snapping Process

This section discusses aspects of the actual snapping process that can comprise up to three phases. In the first phase, a snappable object is recognized, in the second phase the snapper attracts it and finally in the third phase – the after phase – the object may be automatically arranged and grouped within or manually removed from the snapper.

In the first phase – the recognition phase – of the snapping process the snapper will decide if an object that resides within the snapping area's boundaries will be attracted to the snapper. In order that a snappable object will not be pulled from a user's hand while dragging it, interaction with the object must be finished. Basically, each snapper attracts all kinds of snappables. For instance, if snappables are geometric objects like squares and circles with different colors, a snapper does not discern between shapes and colors it recognizes. However, it is conceivable that snappers attract only certain snappables. For instance, in a tabletop application in which different kinds of documents like photos and videos are used, a snapper could deliberately attract only photos and not videos. For instance, a tool set of prop snappers could be provided for a TUI that provides Goal-snappers with different behaviors. Alternatively, a user could configure constraints for a standard snapper to create custom behavior. If an object has been recognized in the first phase, the snapper attracts it in the second phase. In order to clarify the attraction process, the attraction process should be performed in an animated movement. In the third phase the snapper may automatically group, arrange and process snapped objects. For instance, if a snapper has attracted different kinds of documents like photos or videos, the snapper should group each kind of document in a pile. Within the piles, miniatures of snapped documents give a visual cue, which documents have been snapped. Additionally, miniatures provide interaction affordances that allow manual removing of accidently snapped objects. Additionally, it is conceivable to provide a possibility to release all snapped objects at once, for instance, with a shaking gestures on the snapper. The snapper could be coupled with further program logic. For instance, it could be thought of a shrink snapper that automatically downsizes snapped documents.

Summing up, Goal-snapping brings the basic snapping concept to hybrid surfaces. Due to the extended properties of interactive displays, compared to a classical desktop, Goal-snapping also implicates a vast design-space for concrete application scenarios.

4 User Study

We carried out a user study to make qualified statements about the Goal-snapping concept. 20 subjects participated (13 male, 7 female), who were aged between 20 and 28 years (25.15 years in average). 95% were familiar with using touch sensitive interaction devices

like a touch pad, 100% were familiar with using single-touch display devices like a public ticket machine, 65% were familiar with using multi-touch display devices like multi-touch enabled mobile devices and 45% were familiar with hybrid surfaces.

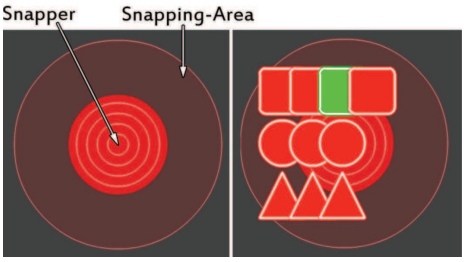
We created two different test setups to consider different aspects of the Goal-snapping concept. The first task consisted of a general grouping task where objects had to be sorted into pre-defined target areas. The second task took place in the area of collaborative work and tangible user interfaces. Here, we analyzed how Goal-snapping can support the positioning of virtual objects over a long distance.

The study was carried out on a hybrid surface called "TwinTable" that we have developed (see Figure 2). Its projection area has a size of 80×45 centimeters where a full HD resolution (1080p) is provided. The TwinTable supports multi-touch and tangible interaction.

4.1 Test1: Evaluating Object Grouping

In the first test, we examined the performance of Goal-snapping for the task of grouping virtual objects into pre-defined target areas. For this purpose, we compared the grouping task for snapping and non-snapping areas and tested it with different combinations of interaction techniques.

(a) The Twin Table consists of a hybrid surface and a passive display



(b) Goal-snapping without (left) and with snapped objects (right)

Figure 2: (a) Our Twin Table (b) Our realization of Goal-snapping

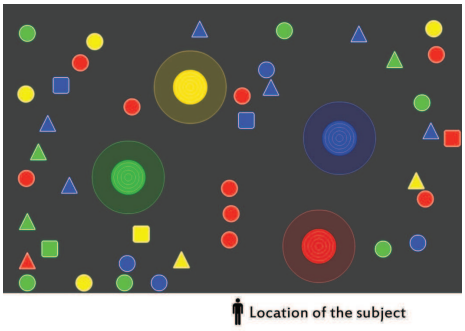
4.1.1 Snapping Design

We designed the snappers as circular objects with a circular, surrounding snapping area (see Figure 2). The participants generally could not move snappers. As long as a participant dragged an object over a snapper, it did not attract the object. Only if an object was dropped within or flicked to the boundaries of a snapping area, it was snapped. If the

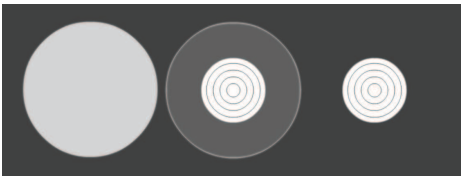
snapping process had triggered, the object was arranged in the snapper according to Figure 2. As illustrated, snapped objects were stacked together and squares, circles and triangles were arranged in different rows. Snapped objects could also be removed from a snapper by using drag & drop interaction.

4.1.2 Test Setup

The object grouping test consisted of five different tasks. At the beginning of each task, four target areas in the colors red, yellow, green and blue and 40 objects each with a random color out of red, yellow, green, blue and a shape out of square, circle, triangle were automatically positioned at random positions on the surface. This initial arrangement of target zones and objects was realized in a way that none of the objects were overlapping each other. Figure 3 shows an example distribution of target zones and objects at the beginning of a task. The 40 objects had to be grouped into the four different target zones according to their color. The different shapes of the objects indicated diversity among them, but did not have to be regarded for the tasks. The test scenarios differed slightly regarding the characteristics of the target zones as well as the allowed interaction with the objects. As different characteristics of the target zones were "passive target zones" (areas that did not snap), "snapping target zones with visible snapping area" and "snapping target zones without visible snapping area". These different target types are illustrated in Figure 3.



(a) A typical setup of target zones and objects right after the beginning of a tests scenario



(b) Passive target zone (left), snapping target zones with visible snapping area (middle) and snapping target zones without visible snapping area (right)

Figure 3: (a) Setup of the first test (b) Target zone designs for the user test

To evaluate the influence of flicking on grouping with Goal-snapping, either "drag & drop" or "drag & drop and flicking" interaction was allowed in the tasks. The subjects were allowed to use both hands and any number of fingers of each hand. This enabled to drag or flick more than one object at a time. If an object had hit the border of the interactive surface, it was automatically rebounded. The table in Figure 4 shows the five test scenarios.

The participants had to perform the tasks in random order to compensate for training effects. Prior to each scenario, the upcoming kind of target areas and the allowed interaction

Test Scenario	Target Zone	Drag&Drop enabled	Flicking enabled	Snapping-Area visible
1A	Passive	Yes	No	Not Available
1B	Passive	Yes	Yes	Not Available
1C	Snapping	Yes	No	Yes
1D	Snapping	Yes	Yes	Yes
1E	Snapping	Yes	Yes	No

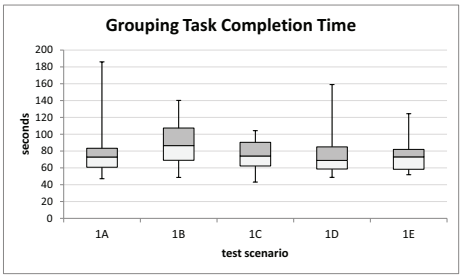
Figure 4: Properties of the five test scenarios for color grouping tasks

were introduced to the participants.

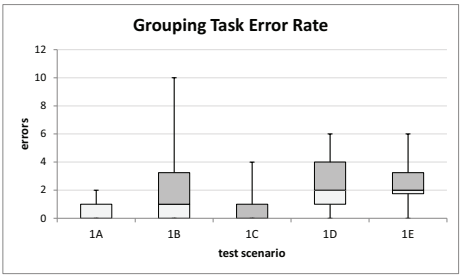
Within each scenario, the color grouping task could be performed with any of the enabled interaction techniques. The time to complete each task was measured. Although the snappers had different colors, each snapper attracted every object within its snapping area no matter which color or shape it had. The number of objects that were temporarily sorted into wrong snappers was recorded. After the five tasks participants had to fill-out a questionnaire. The questionnaire mainly consisted of questions that had to be answered based on a semantic differential scales and on Likert scales, each having seven items. Also, the different target area types had to be ranked by the participants according to their preference. Finally, the participants were asked to give additional textual feedback.

4.1.3 Results

Regarding the measured completion time, we could not make any significant observations about an acceleration of task completion caused by snapping as compared to non-snapping target zones. The completion times are illustrated in Figure 5. The data we gathered on accidentally wrongly snapped objects leads to more significant insights. Here, we could verify that with snapping target zones, flicking increases this error rate. This could be proved as well for the snapping target zones with visualized snapping area (binomial test, $p < 0.02$) as well as for snapping targets without visualized snapping area (binomial test, $p < 0.01$). The corresponding data is shown in Figure 5. Furthermore, we could perceive from the questionnaire that the subjects clearly preferred snapping target zones as compared to passive target zones for object grouping (binomial test, $p < 0.01$). Additionally, participants found the grouping of objects into snapping target zones more comfortable as compared to grouping into passive target zones (binomial test, $p < 0.01$). The visual feedback, that the snapping target zones offered, gave the participants a stronger impression of actually having added an object to a group (binomial test, $p < 0.01$). The automatic arrangement within the snapping target areas was perceived as more clearly arranged than the arbitrary arrangement within non-snapping target zones (Wilcoxon signed rank test, $p < 0.01$). We regarded flicking to be helpful for sorting objects into snapping zones if the participants chose one of the two most agreeing items of the seven items on the Likert scale. We could verify that flicking is helpful (binomial test, $p < 0.01$). However, it could not be verified that snapping was regarded as helpful for the grouping task.



(a) Completion time in seconds for the object grouping tasks



(b) Number of objects that were temporarily sorted to a wrong target group

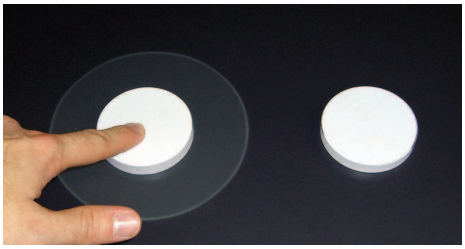
Figure 5: Measured data from the color grouping tasks

4.2 Test2: Evaluating Long Range Object Positioning

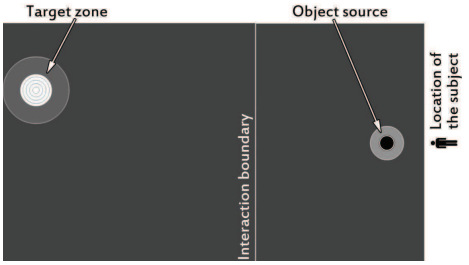
In contrast to object grouping, the second test focused on long range positioning of objects into a target that is out of a participant's reach.

4.2.1 Snapping Design

The snapping design for this test was almost equal as for the object grouping test. As it was not important to permanently group snapped objects, we changed their handling in the after phase. Snapped objects had been moved to the center of the snapper in an animated movement and were faded out afterwards. Additionally to the target zone types from the first test (Figure 3), we also employed props as snapping targets (Figure 6). These physically represented snappers were designed similar to virtual snapping zones.



(a) Physical props as snapping target zones – with and without a visualization of the snapping area



(b) The setup for the long-range positioning scenario

Figure 6: (a) Snapping Props (b) Setup of the second test

4.2.2 Test Setup

The setup for this test is illustrated in Figure 6. Participants had to flick objects into target areas on the opposite side of the table. In order that tall participants could not just bend over the table and drop objects into the target area, participants were not allowed to cross an interaction boundary with their fingers, which was located after one third of the surface. During each task, participants had 15 attempts to flick an object to the target. After every attempt, the target moved to a new random position on the far side of the surface. If a prop was employed, it was manually moved to the new position. In Figure 7, the five tasks are listed in a table. The tasks differed only in the target zone characteristics.

Test Scenario	Target Zone	Snapping-Area visible
2A	Passive	Not Available
2B	Virtual Snapper	Yes
2C	Virtual Snapper	No
2D	Prop Snapper	Yes
2E	Prop Snapper	No

Figure 7: Properties of the five test scenarios for long-range positioning tasks

Like in the first test, the order of the scenarios was randomized to compensate for training effects. We considered a passive target zone to be hit if the object stopped at least touching its boundary. In contrast to the grouping test, the rebounding of snappables from surface edges was disabled in order to make sure that the subjects had to hit the targets in a direct way. If the objects did not hit the target, they faded out at the position they stopped after flicking. For each task, the amount of objects that were successfully placed in the target area was determined. After the participants completed all tasks, they were asked to answer a questionnaire, which was structured accordingly to the object grouping test questionnaire in section 4.1.2.

4.2.3 Results

In this test, we could clearly measure a significant improvement of the hit rate within the snapping scenarios 2B (Wilcoxon signed rank test, $p < 0.01$), 2C (Wilcoxon signed rank test, $p < 0.01$), 2D (Wilcoxon signed rank test, $p < 0.01$) and 2E (Wilcoxon signed rank test, $p < 0.01$) as compared to the non-snapping scenario 2A. This result is also presented in Figure 8. In the questionnaire, the participants ranked snapping target areas over passive target areas concerning the time they needed to get accustomed to the positioning task (Wilcoxon signed rank test, $p < 0.01$) as well as to the easiness of the task (Wilcoxon signed rank test, $p < 0.01$). We regarded the subjects to evaluate the visualization of the snapping area as helpful if they chose one of the two most agreeing items on the Likert scale. However, it could not be verified that the participants found visible snapping areas helpful (Wilcoxon signed rank test, $p < 0.03$). Additionally, it could not be verified that the participants estimate a prop snapper's location better than the position of a virtual snapper.

In general, the participants preferred snapping zones to the passive target zones for long range object positioning (binomial test, $p < 0.01$). The question whether they preferred the virtual snappers or the prop snapper was not answered significantly. Concerning aspects between both tests, we can state that visible snapping areas were ranked more useful for long range object positioning than for object grouping (Wilcoxon signed rank test, $p < 0.03$).

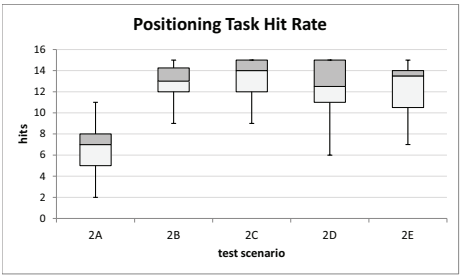


Figure 8: Number of hits for the long range positioning tasks

5 Conclusion

We introduce a basic concept for the use of snapping on hybrid surfaces, called Goal-snapping. It can aid sorting and aligning objects on an interactive surface by grouping them within snapping target areas. Goal-snappers can be employed in face-to-face collaboration on a tabletop setup to facilitate the exchange of objects between users over distances that are out of users' reach. This can be provided with a Goal-snapper for each user to which other users can flick objects. We carefully examined and described the design space of the basic concept which includes representation, visualization and interaction aspects of snappers, snappables and snapping areas. In a user study with 20 participants, we evaluated the usefulness and different design and interaction approaches for Goal-snapping in two scenarios for object grouping and long range object positioning. Although we figured out that Goal-snapping does not significantly accelerate the completion time for grouping, users prefer snapping target areas that automatically group objects as compared to passive target areas that do not snap objects. In terms of target area design, a visualization of the exact target area is more important for long range positioning tasks than for object grouping tasks. Although users regard flicking as helpful for grouping tasks, using flicking has the drawback of a higher rate of accidentally snapped objects. However, in the long distance positioning task, Goal-snapping zones significantly enhance the hit rate as compared to passive target zones. Additionally, participants agree that they find it easier to hit Goal-snapping targets than passive targets. The design choice to use a real world prop instead of a graphical representation does not significantly improve long distance positioning tasks.

Acknowledgements

This research was supported by the Federal Ministry of Education and Research (BMBF) of Germany.

References

[AB06] Anand Agarawala and Ravin Balakrishnan. Keepin' It Real: Pushing the Desktop Metaphor with Physics, Piles and the Pen. In *Proceedings of the SIGCHI conference on Human Factors in computing systems*, CHI '06, pages 1283–1292, New York, NY, USA, 2006. ACM.

[BS86] Eric A. Bier and Maureen C. Stone. Snap-dragging. In *Proceedings of the 13th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '86, pages 233–240, New York, NY, USA, 1986. ACM.

[HCV+06] Mark S. Hancock, Sheelagh Carpendale, Frederic D. Vernier, Daniel Wigdor, and Chia Shen. Rotation and Translation Mechanisms for Tabletop Interaction. In *Proceedings of the First IEEE International Workshop on Horizontal Interactive Human-Computer Systems*, pages 79–88, Washington, DC, USA, 2006. IEEE Computer Society.

[LPS+06] Jun Liu, David Pinelle, Samer Sallam, Sriram Subramanian, and Carl Gutwin. TNT: Improved Rotation and Translation on Digital Tables. In *Proceedings of Graphics Interface 2006*, GI '06, pages 25–32, Toronto, Ont., Canada, Canada, 2006. Canadian Information Processing Society.

[MHPW06] Meredith Ringel Morris, Anqi Huang, Andreas Paepcke, and Terry Winograd. Cooperative Gestures: Multi-User Gestural Interactions for Co-located Groupware. In *Proceedings of the SIGCHI conference on Human Factors in computing systems*, CHI '06, pages 1201–1210, New York, NY, USA, 2006. ACM.

[NBBW09] Miguel A. Nacenta, Patrick Baudisch, Hrvoje Benko, and Andy Wilson. Separability of Spatial Manipulations in Multi-Touch Interfaces. In *Proceedings of Graphics Interface 2009*, GI '09, pages 175–182, Toronto, Ont., Canada, Canada, 2009. Canadian Information Processing Society.

[RGS+06] Adrian Reetz, Carl Gutwin, Tadeusz Stach, Miguel Nacenta, and Sriram Subramanian. Superflick: a Natural and Efficient Technique for Long-Distance Object Placement on Digital Tables. In *Proceedings of Graphics Interface 2006*, GI '06, pages 163–170, Toronto, Ont., Canada, Canada, 2006. Canadian Information Processing Society.

[SCH+06] Xiaohua Sun, Patrick Chiu, Jeffrey Huang, Maribeth Back, and Wolf Polak. Implicit Brushing and Target Snapping: Data Exploration and Sense-making on Large Displays. In *Proceedings of the working conference on Advanced visual interfaces*, AVI '06, pages 258–261, New York, NY, USA, 2006. ACM.

[TKR+08] Lucia Terrenghi, David Kirk, Hendrik Richter, Sebastian Krämer, Otmar Hilliges, and Andreas Butz. Physical Handles at the Interactive Surface: Exploring Tangibility and its Benefits. In *Proceedings of the working conference on Advanced visual interfaces*, AVI '08, pages 138–145, New York, NY, USA, 2008. ACM.

A Robust Generation Technique of Common Information Based on Characteristic of Multipath Fading Channel by Shaking Handheld Devices

Tomohiro Iwamoto, Shigeaki Tagashira, Yutaka Arakawa, Akira Fukuda

Graduate School / Faculty of Information Science and Electrical Engineering
Kyushu University
744 Motooka, Nishi-ku, Fukuoka, Japan
{tomohiro, shigeaki, arakawa, fukuda}@f.ait.kyushu-u.ac.jp

Abstract: A rapid increase in handheld devices with wireless communication capabilities, such as cellular phones and smart phones, enables data communication in face-to-face situations. The easy realization of secure data communication in such situations is necessary for ensuring safe and reliable networking environments as short-range wireless networks become more popular. One approach involving the generation of secret keys in each communication device using variations of the received signal strength indication (RSSI) values has been proposed in literature. However, it has a problem in that there are positions where eavesdroppers are able to obtain RSSI values that are highly correlated with those of legitimate devices. To address this problem, we propose a filtering technique that eliminates eavesdroppable parts from RSSI variations. Additionally, it is important to shake either one or both of two legitimate devices to change the propagation, considering that the elimination exploits the periodicity of the shake. Furthermore, we implemented a prototype system realizing the proposed method and evaluated its effectiveness. The results indicate that the proposed method can increase the robustness of common information, which is generated into the secret key without degrading its generation speed.

1 Introduction

Recently, with the rapid proliferation of wireless-enabled applications and devices, such as cellular phones and smart phones, there is an increased opportunity for wireless connections to be made with bystanders. The public nature of wireless transmission for data communication has the potential to allow eavesdroppers easy access to the transmitted data. Therefore, many studies to realize secure communications, such as public key cryptography, quantum cryptography, have been proposed in literature. Among them, secret key cryptography is the most common because its processing speed is so high that computationally limited devices would have to process a large amount of encrypted data. However, it has a problem in that there may be leaks of the secret key during distribution and management. Therefore, the secret key generation technique that uses random fluctuations of the wireless channel between legitimate users has attracted considerable attention as a method that requires no distribution or management. This technique is based on the

reciprocity of radio wave propagation. With this technique, two devices do not have to distribute the secret key because it is generated by each device without the need to send any information regarding it. In addition, there is also no need to manage the secret key because the devices generate a different secret key each time. Furthermore, one of the notable features is that we can change the length of the secret key based on the generation time. The generation of a secret key with a certain length requires modification of the radio propagation and extraction of its characteristics. Methods for changing the propagation have been proposed such as the use of an electronically steerable parasitic array radiator (ESPAR) antenna [1] and multi-antenna [2]. For extraction of the characteristics, techniques including ways to utilize the deviation of the arrival time between direct waves and multipath waves [3] and orthogonal frequency-division multiplexing (OFDM) model [4] have been also proposed. These schemes achieve precise and high-speed generations owing to special devices.

On the other hand, methods for generating the secret key from variations of the received signal strength indication (RSSI) values [5], which can be measured by non-dedicated wireless devices, have been recognized as beneficial for ubiquitous communication systems. However, this method has two problems: one is that the speed with which the secret key is generated is as low as a few bits per second. The other is that there are positions where eavesdroppers are able to obtain highly correlated variations of RSSI values for legitimate users in certain environments [6, 7]. To address the latter problem, we propose a filtering technique that eliminates eavesdroppable parts from variations of RSSI values. Additionally, it is necessary to shake either one or both of two legitimate devices to change the propagation, considering that the elimination exploits the periodicity of the shake. We implement a prototype system that realizes the proposed scheme and generates common information that is processed into the secret key using quantization and reconciliation techniques. In this study, the evaluation of the generated information with correlation coefficients shows the effectiveness of the proposed scheme. The results indicate that the proposed method significantly contributes in improving the robustness of the generated information without degrading the generation speed.

The rest of this study is organized as follows. In Section 2 , we describe the rationale behind the method that shares RSSI values of two devices and implement a preliminary experiment to confirm their effectiveness in real environments. Section 3 presents details of the proposed scheme, and Section 4 shows the effectiveness of the proposed method. Finally, we conclude this paper and discuss our future tasks in Section 5 .

2 Reciprocity of radio wave propagation and fluctuation of RSSI

In this section, we show that legitimate users (Alice and Bob) can observe similar variations of RSSI values because of the reciprocity of radio wave propagation. First, we consider the variation of RSSI values. Second, we introduce the reciprocity of radio wave propagation. Finally, in actual experiments, we confirm that the variation of RSSI values observed by one legitimate user is similar to that by the other.

2.1 Variation of RSSI values

RSSI values fluctuate because of various causes. We summarize these causes as follows.

Cause1: Distance between sender and receiver

Cause2: Multipath fading

Cause3: Noise

Cause1 means that the RSSI value is closely correlated to the distance between legitimate users, i.e., the changes in the distance cause the variation of RSSI values. The RSSI value is high when the distance is short. In contrast, it is low when the distance is long or some obstacles exist between them.

A receiver receives radio waves from a transmitter through both direct and indirect paths (i.e., multipath). The direct wave interferes with the multipath wave, which leads to attenuation of the direct wave. This phenomenon is called ”fading.” The impact of fading varies with differences in the transmission distance between direct and indirect paths. Cause2 means that fading causes variations in RSSI values. Cause3 means that RSSI values fluctuate because of noise, which is dependent on employed devices, white noise, and others.

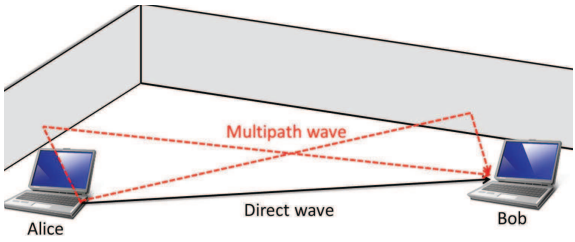
2.2 Reciprocity of radio wave propagation

Radio wave propagation traverses the same path in both directions, from Alice to Bob (Fig.1-(a)) and from Bob to Alice (Fig.1-(b)), unless either of them or any surrounding objects are moved. This is called the ”reciprocity of radio wave propagation.”

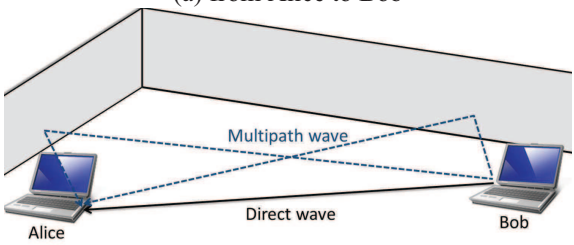
2.3 Preliminary experiment

Both devices can observe the same variation of RSSI values in environments in which the reciprocity of radio wave propagation works well. This similar variation of RSSI values in both directions is mainly due to Cause1 and Cause2, as was described in Section 2.1. In this section, we conduct an experiment with existing equipment and confirm that this works well in real world environments.

We prepared a quiet room (6 meters $\times$ 12 meters) and deployed three laptops: Alice (transmitter), Bob (receiver), and Eve (eavesdropper), as shown in Figure 2. Using a ping command, Alice sent 1000 ICMP echo request packets to Bob every 1.0 s and Bob replied to each packet with an ICMP echo reply packet. During this period, Alice shook her device to change the radio channel and recorded the RSSI values and sequence numbers for the packets in the reply. On the other hand, Bob and Eve recorded those parameters for the request packets. Figure 3 illustrates the positional relation between a laptop and an antenna, and the way that Alice shook her laptop. We used Atheros devices as wireless



(a) from Alice to Bob



(b) from Bob to Alice

Figure 1: Transmission path for radio signal between Alice and Bob.

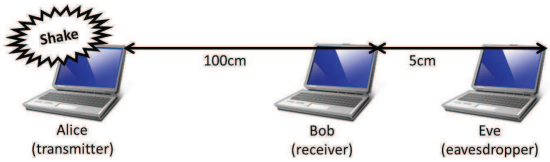


Figure 2: Location of terminal devices in this preliminary experiment

network cards, operating in the IEEE802.11a band. In addition, to avoid interference during the observations, we turned off all additional functions, such as diversity.

Figure 4 shows the variations of the RSSI values observed in this experiment. The vertical axis represents the RSSI value, and the horizontal axis represents the sequence number. Closed diamonds, triangles, and squares indicate the RSSI values recorded by Alice ($RSSI_{Ab}$ profile), Bob ($RSSI_{Ba}$ profile), and Eve ($RSSI_{Ca}$ profile), respectively. The recordings made by Eve were covert. It is clear from Figure 4 that legitimate users can observe similar fluctuations even in real environmental conditions. However, the fluctuations in $RSSI_{Ab}$ and $RSSI_{Bc}$ are not identical because Alice and Bob were not able to simultaneously transmit and receive the signals using typical commercial wireless transceivers. We also consider noise to be a factor contributing to these gaps. In this experiment, the deviation was small, since the round trip time (RTT) was much shorter than the time required for changing channels. In addition, the impact of Cause1 and Cause2 was much

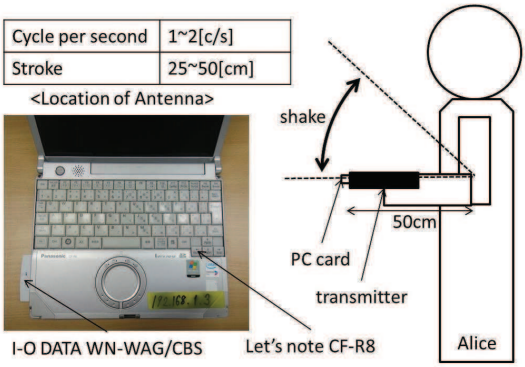


Figure 3: Method for mounting antenna and shaking terminal device

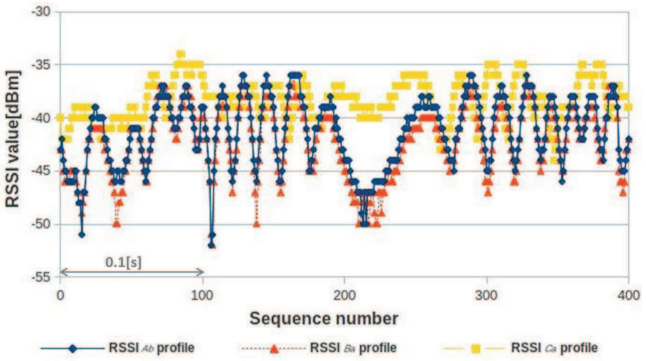


Figure 4: Fluctuation of observed RSSI values in this preminary experiment

more significant than that of Cause3. It is widely recognized that the fluctuation is an uncontrollable random phenomena. This makes it very difficult for it to be estimated by eavesdroppers. However, eavesdroppers can observe RSSI variations that correlate well with those of legitimate users, who may be present at a location less than a few wavelengths from either of them, as was the case with Eve in Figure 2.

3 Proposed method

In this section, we describe our proposed scheme, which provides legitimate users with common information generated into the secret key using RSS-based secret key extractions, as in [1, 8, 9, 10]. The secret key also makes it difficult for an eavesdropper to carry out sniffing attacks. The main idea of our proposed method is to use a bandpass filter to extract

only common fluctuations and eliminate the part of the RSSI variation that is vulnerable to eavesdropping. First, we explain the method for making RSSI profiles which are used to generate common information. Second, we discuss the implementation of the bandpass filter.

3.1 Procedure for making RSSI profiles

In this study, we call records for a tuple of two items: RSSI value and its sequence number "RSSI profile." We describe below instructions for making an RSSI profile.
[**Procedure for making an RSSI profile**]

Step1: Either or both users shake their devices.

Step2: During shaking, the transmitter sends an ICMP echo request packet to a receiver as often as needed to a predetermined set.

Step3: Whenever the receiver gets the request packet, the receiver replies with an ICMP echo reply packet to the transmitter.

Step4: The receiver records the RSSI value and sequence number for these request packets.

Step5: The transmitter records those parameters for the reply packets.

Each legitimate user makes an RSSI profile in accordance with these steps. In the next section, we will explain how to extract common information from this RSSI profile. To get closer common information, it is recommended that the transmitter sends a request packet more frequently and that the RTT be shorter. In the next section, we show that the transmitter has to send more than $2 \times \beta$ request packets every second.

3.2 Extraction of common information with a bandpass-filter

In this section, we propose a filtering technique to extract correlated common information that exists for legitimate users from the RSSI profiles. More specifically, we analyze the RSSI profile on the frequencies and reduce unsuitable frequencies in the spectrum using the discrete Fourier transform (DFT). In this study, "unsuitable frequencies" can be estimated by eavesdroppers.

In Section 2.1, we discussed the fluctuations that occur. The result of analyzing the fluctuations of the RSSI profile in the frequency domain is shown in Figure 5, which shows that fluctuation1 and fluctuation2 have almost the same variation of RSSI values between the legitimate users owing to the reciprocity of radio wave propagation. These variations gradually decline with increasing frequency. In contrast, fluctuation3 is observed as a different variation. If white noise exists, it has a constant power over the entire range of

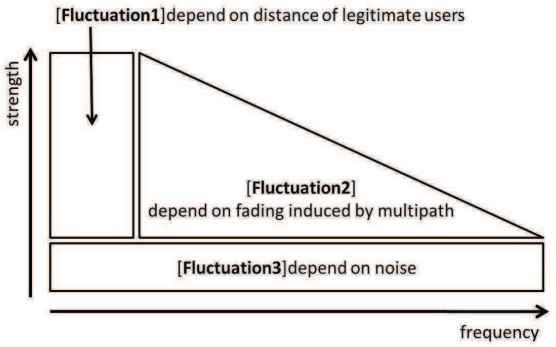


Figure 5: Modeling variation of RSSI values in the frequency domain

frequencies. Therefore, for high frequencies, the noise decreases the correlation of the RSSI profile between legitimate users. On the other hand, for low frequencies, the variation is likely to be estimated by eavesdroppers. For these reasons, we propose using a bandpass filter that allows only frequencies from α Hz to β Hz. In our method, α and β are customized parameters.

4 Evaluation

In this section, we discuss the effectiveness of our proposed scheme in a real environment. In particular, we confirm that our scheme reduces the correlation of the RSSI variation between the eavesdropper and legitimate users, even when the eavesdropper is very close to either of the legitimate users or on the axis of the signal connecting the legitimate users. In [6, 7], it was reported that in these positions, eavesdroppers are able to obtain RSSI variations that agree well with those of legitimate users. First, we explain about data sets used in this evaluation. Second, we select the customized parameters of the bandpass filter (α and β) and evaluate our scheme with the correlation coefficient.

4.1 Data set

We made many sets of RSSI profiles for this evaluation. The procedure for making RSSI profiles was described in Section 3. In this evaluation, we used setting values and environment similar to those used in Section 2.1, except for the distance between Bob and Eve, i.e., Eve is placed on the line connecting Alice and Bob. More specifically, we placed Eve at 30 points between 1 and 100 cm and made 10 sets of RSSI profiles for each point. We show an adversary model of this evaluation as follows. We follow the model as described in [5].

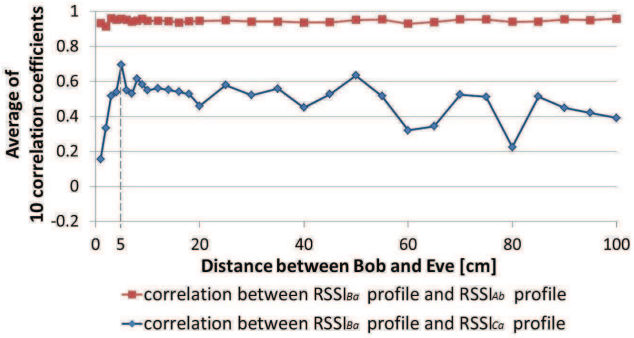


Figure 6: Relation between correlation coefficient and distance before using bandpass-filter

[**Adversary model**]

- Eve can listen to the communication between Alice and Bob.
- Eve knows our proposed scheme and the parameters used in our scheme.
- Eve can observe RSSI values everywhere.
- Eve is a passive adversary who cannot carry out a person-in-the-middle attack.

4.2 Parameter setup

Figure 6 shows the relationship between the strength of the correlation and Eve's position. We employ the correlation coefficient as a metric to evaluate the correlation strength. Closed diamonds represent the correlation coefficient between Alice and Bob, and closed squares represent that between Bob and Eve. From the figure, we observe that Eve can estimate a part of the legitimate RSSI profile, when she is on the line between Alice and Bob. In particular, Eve's RSSI profile has the highest correlation when the distance between Bob and Eve was 5 cm (nearly equal to one wavelength of 5 GHz). We focus on this point and select appropriate values for α and β.

We filtered the RSSI profiles ($RSSI_{Ab}$, $RSSI_{Ba}$, and $RSSI_{Ca}$) that were observed when the distance between Bob and Eve was 5 cm, with the bandpass filter allowing only frequencies from α Hz to β Hz. We call these outputs of the bandpass filter "common information." The effect of the tuning parameters (α and β) on the correlation coefficient between legitimate common information are exhibited in Figure 7. In Figure 8, we also show the correlation coefficient between the common information for Bob and Eve. When α is less than five, although the correlation between the legitimate common information

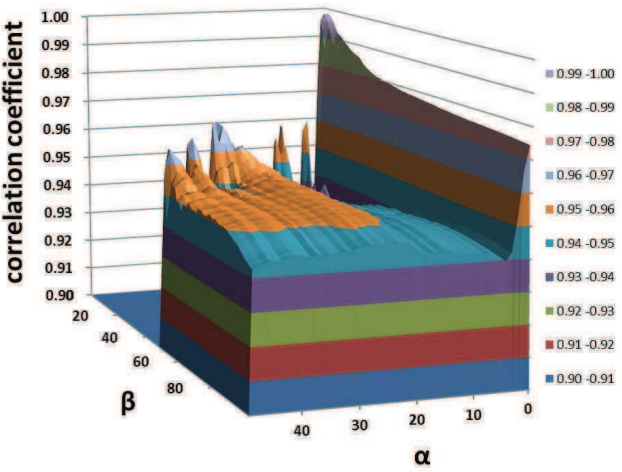


Figure 7: Effect of bandpass-filter on correlation between Alice and Bob

is high, the correlation between that of Eve and Bob is also high. Conversely, both correlations are low when α is high. Moreover, few frequencies pass the filter for sufficient common information to be generated when the range between α and β is small. We examine the best parameters when the difference in the correlation coefficient for legitimate common information and that of Bob and Eve is the highest. As a result, we found that the best-case scenario occurs when α equals 20 and β equals 80. In this evaluation, we determine the effectiveness of the bandpass filter using these values.

4.3 Effectiveness of bandpass filter

We filter the RSSI profiles that are observed in the preliminary experiment with the bandpass filter (α = 20, β = 80). The results are shown in Figure 9, which shows that the correlation between either of the legitimate users and the eavesdropper decreases after filtering. Furthermore, we observe that legitimate users can extract delicate changes of channels since the filter eliminates against the direct wave. This will contribute to increasing the speed with which information is generated.

Similarly, we filter all data sets obtained with the bandpass filter and generate common information. Figure 10 shows the relationship between the correlation of the common information and Eve's location. When compared with Figure 6, it is obvious that the inclusion of the bandpass filter can decrease the correlation between either the legitimate users and eavesdropper, without degrading the correlation between legitimate users.

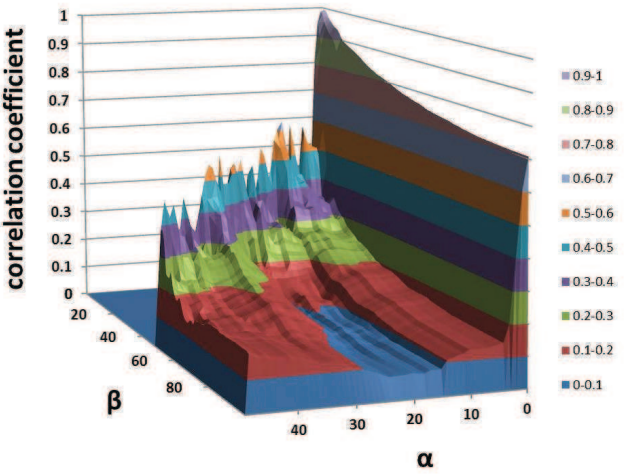


Figure 8: Effect of bandpass-filter on correlation between Bob and Eve

5 Conclusion

We resolved the problem that allowed eavesdroppers at specific positions in certain environments to obtain RSSI values that were highly correlated with those of legitimate users. This occurred when generating secret keys from RSSI variations on the wireless channel. In particular, we used a bandpass filter to make RSSI profiles which were generated into secret keys that were more robust to eavesdropping. Our bandpass filter eliminates vulnerable fluctuation and noise. We also conducted evaluations by performing actual experiments. Our experimental results indicate that our scheme can make it difficult for eavesdroppers to estimate the secret key, without degrading the correlation between legitimate common information. If our scheme is applied to approaches proposed in [1, 8, 9, 10], we can realize more robust secret keys. Also, our scheme appears to increase the speed with which secret keys are generated.

However, we have evaluated the proposed method in only a few environments. This makes our scheme more applicable to the evaluation of the proposed method in various kinds of environments. For example, we evaluate the proposed method in various rooms that have different dimensions and noise using both heterogeneous PC cards and laptops by shaking and setting in other ways, changing the positional relation between legitimate users, and so on. In the future, we will consider applying the proposed method to the automatic selection of tuning parameters with noise levels and analyze the most suitable approaches that generate secret keys from RSSI fluctuation.

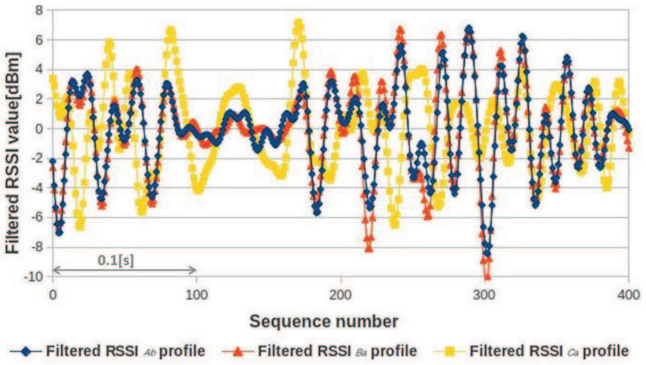


Figure 9: Result of filtering RSSI profiles in preliminary experiment

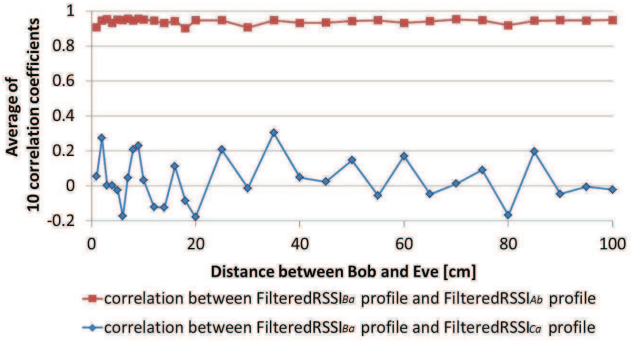


Figure 10: Relation between correlation coefficient and distance after using bandpass-filter

References

[1] T. Aono, K. Higuchi, T. Ohira, B. Komiyama, and H. Sasaoka. "Wireless secret key generation exploitingreactance-domain scalar response of multipath fading channels,"*IEEE Transactions on Antennas and Propagation*, Vol. 53, No. 11, pp. 3776–3784, 2005.

[2] T. Nishino, H. Iwai and H. Sasaoka. "A study on Secret Key Agreement Scheme in Multi-Antenna System Based on Radio Propagation Characteristics," *IEICE*, Vol. 108, No. 445, pp. 373–378, 2009.

[3] A. Kitaura, T. Sumi, T. Tango, H. Iwai and H. Sasaoka. "A Secret Key Agreement Scheme Based on Multipath Time Delay in UWB System," *Communication Technology, 2006. ICCT '06. International Conference on,* pp. 1-4, 2006.

[4] A. Kitaura, H. Sasaoka, "A scheme of Private Key Agreement Based on the Channel Characteristics in OFDM land mobile radio," *Electronics and Communications in Japan,* Vol. 88, No. 9, 2005.

[5] S. Jana, S. N. Premnath, M. Clark, S. K. Kasera, N Patwari, S. V. Krishnamurthy, “On the Effectiveness of Secret Key Extraction from Wireless Signal Strength in Real Environments,” *In MobiCom ’09: Proceedings of the 15th Annual international conference on Mobile computing and networking,* pp. 321–332, 2009.

[6] S. Kwamura, T. Shimizu, H. Iwai and H. Sasaoka. “Position Dependence of Key Capacity in Secrete Key Agreement Scheme Using ESPAR Antenna,” *International Symposium on Antennas and Propagation (ISAP2009),* 2009.

[7] M. Onishi, T. Kitano, Iwai and H. Sasaoka. “Improvement of Tolerance for Eavesdropping in Wireless Key Agreement Scheme Using ESPAR Antenna Based on Interference Transmission,” *International Symposium on Antennas and Propagation (ISAP2009),* 2009.

[8] S. Mathur, W. Trappe, N. Mandayam, C. Ye, and A. Reznik. “Radio-telepathy: Extracting a Secret Key from an Unauthenticated Wireless Channel,” *In MobiCom ’08: Proceedings of the 14th ACM International Conference on Mobile Computing and Networking,* pp. 128–139, 2008.

[9] M. A. Tope and J. C. McEachen. “Unconditionally securecommunications over fading channels,” *In Military Communications Conference (MILCOM 2001),* Vol. 1, pp. 54–58, 2001.

[10] B. Azimi-Sadjadi, A. Kiayias, A. Mercado, and B. Yener. “Robust key generation from signal envelopes in wireless networks,” *In CCS ’07: Proceedings of the 14th ACM Conference on Computer and Communications Security,* pp. 401–410, 2007.

Towards Integration of User Interaction and Context Event Processing in Intelligent Living Environments

Simon Lehmann Jan Schäfer Ralf Dörner Ulrich Schwanecke
Design Computer Science Media Department
RheinMain University of Applied Sciences
Wiesbaden, Germany
{simon.lehmann,jan.schaefer,ralf.doerner,ulrich.schwanecke}@hs-rm.de

Abstract: Event processing plays a significant role in the current development of intelligent living environments. It ranges from processing of information produced by a magnitude of sensors to gain insight into the activities of the inhabitants on a more global scale, to the processing of immediate and rather short-lived events of user input on and around interactive systems embedded in common household furniture like tabletops or tablets. Based on the work conducted separately in those two fields, we found that the still evolving field of complex event processing (CEP) provides the methods and tools to handle those distinct use-cases equally. Especially the application to interactive systems, while being novel and uncommon, is well suited and further shows the broad applicability of CEP. The comparison of the two application fields shows that, even though the events occurring in them are distinguished by their intention, commonalities do exist and provide integration points. Furthermore, the integration of those applications within the context of smart homes allows to provide demand-oriented resource management, which realizes self adaptation and control.

1 Introduction

Intelligent living environments, also known as smart homes, aim at providing its residents with useful services and assistance for everyday activities. These services cover different aspects, such as health monitoring, energy management, environment control, security, communication, or productivity tasks. Social and ethical challenges aside, major technical challenges in this field of research are modeling and prediction of the activities and actions of the residents, multi-modality of interaction with various kinds of computer systems, and intelligent monitoring of systems components [HSB09].

On a technical level, these applications are all based on the processing and interpretation of context events happening throughout the system and its environment. Be it events produced by sensors for monitoring of the residents location, events coming from user input devices, or events of the internal state of the system. So looking at the way event processing is done in different areas of intelligent living environments, which are usually treated separately, allows to gain insight about the individual requirements, but also the commonalities shared by all parts of the system.

In this paper, we explore the possible points of integration of two areas of event processing

in an intelligent environment setting. We identify those areas, where each application field can benefit from the other, but also define the central characteristic which distinguishes both fields from each other, namely the intention of the processed events. Additionally, we introduce a *demand-oriented resource management* and discuss how it is made possible within that context.

We base our work on two concrete systems. The first deals with systems management. The second employs complex event processing to handle the interaction with more complex input devices of interactive tabletop systems, which is a rather novel and uncommon field of application. Even though the two systems are not interacting with each-other and were developed separately for their individual purpose, both use complex event processing methods and techniques. More specifically, both use Esper [Esp11] as an underlying technology, which provides the additional benefit of having similarities on a technical level.

In the following, we will first give a brief overview of what complex event processing is and how the general methods and techniques are provided by a concrete implementation, Esper. We then proceed to describe the two areas of application, and how they relate to the context of intelligent living environments, in more detail. Individual requirements, challenges and results are presented with each application. After that, the distinguishing characteristic of both application fields is defined in more detail, followed by a discussion of possible points of integration between those systems. Finally, we present our proposal of a demand-oriented resource management and conclude with a short summary and outlook on future work.

2 Complex Event Processing with Esper

Complex event processing means the detection, analysis, and general processing of correlated raw or simple events, which results in more abstract and meaningful complex events [Luc02]. As a very general term, it covers a large field of techniques and methods for detection of relationships or patterns of events, event abstraction, modeling of events and event hierarchies, and abstraction of the whole process of event processing. It builds upon the concept of the event driven architecture (EDA), in which loosely coupled components communicate by emitting and consuming events [LS11]. In an EDA, any notable change of state or thing that happens is considered an event, which is propagated to all components interested in them. The components evaluate the information of the events and take any necessary action, which might include generation of further events [Mic06].

The software library *Esper* provides a general purpose component for complex event processing in Java or .NET. It is based on the principle of *continuous queries* [TGNO92], where processing, filtering and dissemination of events happens by querying the engine, which continuously tries to match and process events according to the active queries. Queries are formulated using the *event processing language* (EPL). The EPL is a SQL-like declarative language, which supports selecting events from streams, conditions, sliding windows (time and length based), aggregation functions, joins, pattern matching, insertion into event streams and many other features known from relational databases (examples of

EPL queries are given in section 3.2.2).

The engine provides two mechanisms for querying events. The first allows full processing and manipulation of event streams in the engine and is available by normal EPL statements. The second method allows notification about events matching a specified *pattern*, which is expressed by the pattern syntax of the EPL. While the second method is in fact a subset of the first, it has the benefit of a reduced syntax and also remove some overhead, as they are implemented using state machines only.

From a software development point of view, Esper allows to extend its functionality in many ways (e.g. custom sliding windows or aggregation functions), which allows using it in novel contexts and use cases. It also does not define a specific event representation or how processing of events is executed, i.e. single- or multi-threaded.

3 Event Processing in Intelligent Living Environments

3.1 Home Service Platforms

With increasing integration of distributed systems in living environments, the amount of generated data that has to be processed and interpreted by home service platforms is also on the rise. Ambient sensors collect, (pre)process and forward large amounts of data generated by inhabitants and their environment including but not limited to vital signs, deduced activities, device states and environmental parameters such as temperature or humidity. Together, these values represent the *context*, which is needed to develop context-aware, adaptive applications for the platform.

3.1.1 Information Integration and Deduction

Applications in intelligent living environments rely on context information to be able to adapt dynamically to the changing user's needs or preferences. Unless an application is very simple (e.g. clock or weather display), it requires deduced information, which can be gained by combination and interpretation of events created by the platform's components (devices and software). The resulting deduced information represents more abstract information of higher value for application development (e.g. a recognized inhabitant activity). If the platform is able to produce and provide this abstract knowledge for applications, it removes the burden from applications to calculate it individually.

The more heterogeneous the systems in an intelligent living environment are, the harder it becomes to interpret and relate the events generated by them. This can be solved by manually mapping input and output of each to be integrated system to each different system or application. A much more elegant approach to solve this problem is to integrate all context producing and consuming components into a common context model, a *context ontology*, which is managed by the service platform. This allows to semantically integrate and prepare information from arbitrary sources for consumption of context data by applications.

An approach for this context model has already been presented in [Sch10], which uses the *Web Ontology Language* (OWL) [Gro09] for ontology modeling. Here, the sub-ontologies of each system contributing to the context model have to be based and integrated with a common basic ontology.

A common context model offers additional benefits. Just like arbitrary context providers or consumers, *context processors* such as CEP engines can use the information stored in the model to deduce high-level information easily, which itself can become part of the model and, thus, be consumed by components of the system.

3.1.2 Information Filtering and Extraction

Creating and managing a context model is not cheap in terms of required processing power, as incoming and outgoing information has to be imported and exported constantly, which always requires information mapping of some kind (e.g. from OWL/XML to some transport protocol or to Java). Some sensors or applications produce large amounts of raw data, from which only the *right* information is required. As the raw data should not become part of the context model (to prevent bloat and unnecessary computational effort), this data has to be preprocessed. Often, the producing component is able to do this itself, but this is not always possible. Then, a CEP engine is used as *context preprocessor*. Instead of relying on information from the context model as input, it receives input streams directly from the data sources. In this case, only the processed output becomes part of the model and, thus, can be used by other applications.

3.2 Interactive Tabletop Systems

Interactive tabletop systems have been in the focus of HCI researchers for about two decades and gained more interest in the past years due to the development of powerful hardware for a variety of input methods. Additionally, the key benefits associated with interactive tabletops – like ease of use, intuitiveness and support for collaborative work – make them suitable for the use in the field of ubiquitous/pervasive computing, which is getting more and more important [TGS06].

Despite the general interest in interactive tabletops and the broad research of tools and techniques for various ways of interaction, the general processing of interaction events happening throughout such a system has not received much attention. Traditionally, input events are handled by some abstraction layer of the operating system, which provides an event dispatching mechanism for user interface toolkits. Events are typically processed by an event loop, i.e. a continuously running process which takes any occurring event as their input and publishes them to one or multiple subscribers. Besides the routing of events from their source to the appropriate user interface components, no additional processing takes place. Any complex event processing which extracts higher level information from events are provided by specialized frameworks on the application level, e.g. for gesture detection.

In contrast to the input devices used with PCs (e.g. keyboard and mouse), applications for interactive tabletop systems make potential use of a wide variety of input methods. Additionally, also other interactive systems do make use of more input devices as their use cases increase. Thus, the need for processing and aggregation of input events arises, which addresses the rising complexity and interdependence of the constantly generated input events by multiple input devices. Based on a previously described architectural approach [LDS+10], we propose a unified tabletop input layer (UTIL) to provide a middleware for processing of interaction events in tabletop systems. At its core it employs Esper for all event processing and querying tasks. The general architecture is implemented in large parts by the Esper engine, with additional components specifically built for the purpose of processing interaction events.

The collection of events from the various event sources is implemented by device dependent collectors, one per input device – an input device being any programmatically accessible source of input events, which does not necessarily correspond to a physical device. The collection process is handled individually for each device, though most use some OSC/TUIO-based protocol for communication. The collectors are responsible for producing the raw event objects, which are then fed into the processing engine. The rules for processing of simple input-events sent by the collectors to complex events are realized as EPL queries that put new or combined events into the event stream.

The interactive applications running on the tabletop system subscribe to events through a centralized query manager, which is provided by the middleware. Applications can connect to it and use it transparently to issue event queries. The application queries are based on the EPL pattern language, which are a subset of the full EPL. They allow for full specification of patterns of events and the issuer of the pattern gets notified whenever an event matches the given criteria. Application queries are fed into the same engine instance responsible for the processing of the rule queries described earlier. These are two design decisions which are related to each other: Giving application developers the ability to use full EPL queries would have two drawbacks while providing only little benefit of enabling applications to use the full potential of the event processing engine. The drawbacks, however, are two-fold. First, writing full EPL queries is much more complex than EPL patterns, thus imposing additional complexity on the application developers. Second, it also enables applications to directly interfere with the processing of the events, which is shared by all applications running concurrently. Thus, giving application developers only a limited subset of the full EPL for their queries, it enforces a true read-only access to the processing system and relieves them of quite some complexity which should not be needed on the application layer.

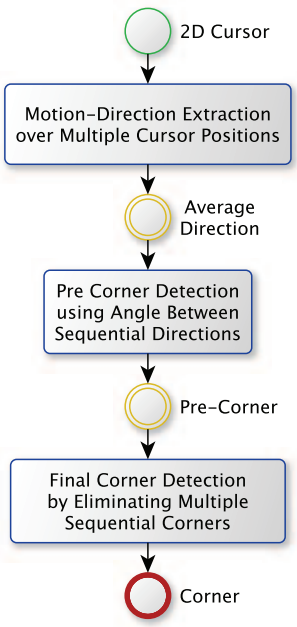
3.2.1 Processing of Spatial Data

Interactive systems are usually comprised of one or several input devices or sensors to allow users of the system to manipulate artifacts in the system. Most of this input deals with spatial information such as the position of a device itself (e.g. a mouse), body-parts of the user (e.g. fingers or hands) or other objects. When processing events mostly consisting of such spatial data, tools specifically designed for this purpose are required. Processing

events based on two or three dimensional spatial data has mostly been done in the area of geo-information processing [Lip09], but also in multimedia communication systems [GYJO10]. Esper itself only supports time or scalar value based processing of events, i.e. events can be processed depending on their temporal relations or based on the amount of events or other scalar valued attributes. Thus, we had to provide a set of single-row functions, aggregation functions, and data window views has to be provided as extensions to Esper to support processing of spatial data in two or three dimensional space.

3.2.2 Gesture Detection

The main purpose of UTIL is to find patterns and correlations of inputs produced by the users. Simple examples include the detection of a two-finger scroll gesture or a multi-finger pinching gesture. These use only one input device and thus also one modality. More advanced examples are correlation of input from different devices, such as multitouch and three-dimensional hand tracking above the table for determining which touches belong to which hand.



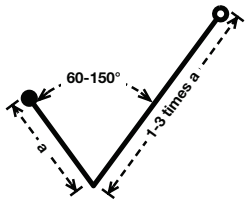

```
INSERT INTO AverageDirectionEvent
SELECT ISTREAM * as cursor,
               avgDirection(position) as dir
FROM Cursor2DEvent.win:length(LENGTH)
GROUP BY cursorId

INSERT INTO PreCornerEvent
SELECT cursor.cursorId AS cursorId,
       first(cursor.pos) AS pos,
       first(dir) AS firstDir,
       last(dir) AS lastDir,
       (last(dir)).angle(first(dir)) AS angle
FROM CursorDirectionEvent.win:length(LENGTH)
GROUP BY cursor.cursorId
HAVING leaving()

INSERT INTO CornerEvent
SELECT cursorId, pos, angle
FROM PreCornerEvent
MATCH_RECOGNIZE (
    PARTITION BY cursorId
    MEASURES A[0].cursorId AS cursorId,
             avgPosition(B.pos) AS pos,
             (CASE WHEN avg(B.angle) < 0
              THEN min(B.angle)
              ELSE max(B.angle) END) AS angle
    PATTERN (A B+ A)
    DEFINE
        A AS abs(A.angle) < ANGLE_THRESHOLD,
        B AS abs(B.angle) >= ANGLE_THRESHOLD
)
```

Figure 1: EPL queries used for detecting corners in 2D strokes drawn with a finger.

A more complex example for interaction event processing, is a two dimensional touch gesture detection based on strokes drawn on the surface of a table. It should be noted that, while the main goal of UTIL is to enhance processing of events from multiple input devices, it also allows to provide complex event processing for a single input device.

The general algorithm behind this gesture detection is based on the ideas presented in [WX10] and [WEH08]. First, corners in strokes drawn with a finger on the surface are detected. Based on the corners, shapes comprised of multiple corners can be detected. As shown in figure 1, the corner detection is fully implemented using only EPL queries. First, the average motion-direction is computed over a window of multiple cursor positions. Second, the angle between sequential directions is determined. Finally, corners are detected by looking for a sequence consisting of an angle below a threshold, followed by one or more angles above the threshold, and terminated by an angle below the threshold. Every time such a sequence is found, the average position and greatest angle is used as the position and angle of the final corner. Additionally, the start and end of each stroke is also considered as a corner.

The actual gestures are described using the match-recognize feature of Esper (which is also used in the final step of the corner detection), which allows to specify regular expressions of events. An example of this is shown in figure 2. The detected gestures are translated into gesture events, consisting of the average position of all corners that make up the gesture and an identifier of the gesture.




```
INSERT INTO GestureEvent
SELECT cursorId, "YES" AS type, pos
FROM CornerEvent
MATCH_RECOGNIZE (
    PARTITION BY cursorId
    MEASURES A.cursorId AS cursorId, B.pos AS pos
    PATTERN (A B C)
    DEFINE
        A AS A.angle = 0,
        B AS B.angle BETWEEN 60 AND 150,
        C AS (C.angle = 0) AND ((B.pos).distance(C.pos)
        BETWEEN (1 * (A.pos).distance(B.pos)) AND
                (3 * (A.pos).distance(B.pos)))
)
```

Figure 2: EPL query used for detecting a ‘Yes‘ gesture based on the previously detected corners.

4 Intentional versus Non-Intentional Events

The two fields of application for event processing presented here share many principal characteristics. Of course, all general characteristics of event driven architectures and event processing systems are found. However, they also share more specific traits, like the presence of multiple, heterogeneous event producing devices, the general need for integration and filtering of those events to deduce and extract higher level information, or the overall application area of dealing with human activities. Especially the last characteristic provides several opportunities for integration of both systems.

Considering all the commonalities, the question arises where to draw the line between event processing for home service platforms and interactive (tabletop) systems. It could be argued that they are actually the same, as user input in an interactive tabletop system can be considered just as another sensor like temperature or vital signs. Vice versa, the

context information collected from ambient sensors could be eventually interpreted as just another input to an interactive system (using a very broad definition of that term).

However, not all characteristics are identical. The two fields of application can be distinguished by the intention of processed events: on the one hand, events like 'opening a door' or 'walking across a room' are considered non-intentional with respect to the home service platform, which only observes these events through ambient sensors. The actions behind the events were not performed to trigger some behavior of the system and a response to that action is usually not its primary purpose. While anything that happens because of non-intentional events may possibly be expected by the residents, this is only expected by habit. On the other hand, events like 'touching a display surface', 'pressing a key', or 'saying a defined command' are considered intentional with respect to the home service platform. The actions are performed just for the purpose of interacting with the system and a reaction is expected. Moreover, the feedback is usually expected to be instantaneously and directly observable. Anything happening only after a certain delay[1] is then considered to be non-responding and may even confuse the users.

5 Points of Integration

The two distinct systems provide one the one hand context events, and direct interaction events on the other. As the comparison in the previous section shows, the separation of both systems should not be easily removed. However, there are potential points of integration, which can be identified.

Applications running on one or more interactive tabletops or surfaces in the living environment could benefit from additional information about the context in which they are used. For example, knowing that a person has entered the room could lead to activation of the interface next to him or her. This means turning the context events into direct interaction events. In addition to the use as simple input events, the context events can of course also be used in aggregation and creation of complex input events. Moreover, the context events can be employed in the realization of multimodal interfaces, where selection of the input modality for interaction might depend on the context in which the application is used. Of course, care must be taken not to interpret to many of the context events as direct interaction, because the users might not have intended this meaning and might not know why the interactive system behaves in a certain way.

In the other direction, the direct interaction events produced by the input devices themselves and also the complex events derived later could be fed into the context event processing system. They provide very fine grained information about the current activity of a person and thus can greatly support otherwise more coarsely captured data about the context. For example, the information that someone is currently interacting with the tabletop system in the living room can be used to simply infer that there is someone in the living room and how long they have probably been there. Also, as direct interaction with system usually requires the main focus of a user, it can be inferred that the person is busy working

[1] delays of up to a few hundred milliseconds are usually acceptable, depending on the kind of interaction

or some other task and in case of elderly people, that they are alive and well. Similarly to the use of context events as input to the interactive system, simply feeding all input events also into the context event processing system is probably not useful. Especially as the interaction events are very fine grained, it might be advisable to apply rate limiting[2] and aggregation on the events.

In the following section, we outline how the integration of the two systems allows for self adaptation by managing the resources based on the demand for events.

6 Demand-Oriented Resource Management

Home service platforms in smart homes have to provide as many services as needed and, at the same time, as few as possible. The former results from the requirement, that residents expect all services to be available when they need them, especially when it comes to interactive systems (as discussed above, delayed or even no response of the system is usually not acceptable in this context). The latter results from the limited resources (processing time, memory, bandwidth, energy, etc.) and the demand for reduced overall energy consumption. To provide a solution to this problem, usage of resources has to be managed based on *demand*. While this is not a novel concept in itself, we introduce how it can specifically be applied to intelligent living environments.

In *demand-oriented resource management*, demand is driven by the event consumers, which are the applications and components interested in the available context and interaction information. As most services in smart homes are eventually provided to the people living there, the demand for context and interaction information depends on the interaction of residents with the system. For example, if no one is currently using an interaction device (such as an interactive tabletop system), it would be sensible to turn the device off as well as turning off any services used by it. Vice versa, if someone enters a room, at least basic input detection of the interactive system has to be turned on again, in order to allow the potential user to start interaction. Also, all intermediary processing of events should be gradually started or stopped, depending on the need for it. This interdependence of the home service platform and the interactive systems are the reason why a demand-oriented resource management in an intelligent living environment requires integration of both systems.

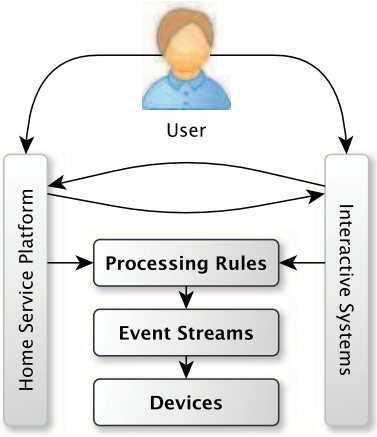


Figure 3: Overview of the demand chain in a demand-oriented resource management for intelligent living environments.

[2] Input events are generated at rates of hundreds to a few thousand events per second, depending on the number of devices and people involved. For example, two input devices with a sampling rate of 60 Hz and two people actively using multitouch gestures with two fingers each, will generate 480 events per second

In the context of complex event processing, managing resources means to manage devices, event sources/streams and processing rules. Figure 3 gives a general overview of the managed components and how each component is driven by the demand from other components. Processing rules, which translate to continuous queries in Esper, are the most specific components. They realize a specific processing step, operate on one or several event streams and may produce events for other rules or consumers. While the event processing engine tries to optimize execution, some resources are still consumed in the form of processing time and memory. As the most specific components, processing rules would be the first candidates for deactivation in case no demand is detected. Event sources or streams are more general, as they are used by a multitude of processing rules or consumers. The streams do usually not require much processing time, but the events have to be stored and possibly sent to other nodes for processing. When all processing rules or consumers interested in a specific event stream have been deactivated, the stream itself would be disabled, which means that no storage or transfer of events occurs. Finally, devices are the coarsest level to manage. They host all event generation, processing or consumption and the main resource consumed by them is energy. As all the previously named components are running on one or more devices, they can be deactivated or shut down, whenever all components currently running on them are deactivated.

These components are disabled or deactivated whenever no demand for the individual components is detected. When demand for certain information is detected again, the process of reactivation is done in reverse and only activates those components which are necessary to fulfill the demand. The management of event processing in this way results in a self adapting system, which efficiently uses resources based on the interaction and activities of the users.

7 Conclusion and Future Work

In this paper, we presented two important application areas for event processing in intelligent living environments and explored their characteristics and commonalities within that context. The potential points of integration indicate, that the two application areas can benefit from each other when information is shared between them. On the other hand, we also showed, that even though many similarities exist, each application area deals with a different kind of events, which are characterized by the intention behind them. Furthermore, we propose a *demand-oriented resource management* in intelligent living environments and outline how it can be realized by integration of user interaction and context event processing.

Based on this, several challenges have to be addressed and more work is required to realize a fully integrated system capable of providing both context and user interaction information and maintaining a self adapting operation. The integration of interaction and context information requires further analysis of their individual characteristics, in order to identify which information can and should be shared and how this information should be interpreted by the other system. Furthermore, the analysis of dependencies in the event processing systems requires to instrument the whole system in an appropriate way. Also, the

demand-oriented resource management requires a reliable way to detect demand based on the user interaction. As the user interaction itself can only be detected when input devices, input event sources and input event processing rules are active, it has to be examined how efficient resource management can be realized even though interdependencies like those exist.

References

[Esp11] EsperTech Inc. ESPER - An Event Stream Processing and Event Correlation Engine (Version 4.4.0). http://esper.codehaus.org, October 2011.

[Gro09] W3C OWL Working Group. OWL 2 Web Ontology Language Document Overview (W3C Recommendation). http://www.w3.org/TR/owl2-overview/, October 2009.

[GYJO10] Mingyan Gao, Xiaoyan Yang, Ramesh Jain, and Beng C Ooi. Spatio-temporal event stream processing in multimedia communication systems. In *Scientific and Statistical Database Management*, pages 602–620. Springer, 2010.

[HSB09] Annika Hinze, Kai Sachs, and Alejandro Buchmann. Event-based applications and enabling technologies. In *DEBS '09: Proceedings of the Third ACM International Conference on Distributed Event-Based Systems*, pages 1–15, 2009.

[LDS+10] Simon Lehmann, Ralf Dörner, Ulrich Schwanecke, Johannes Luderschmidt, and Nadia Haubner. An Architecture for Interaction Event Processing in Tabletop Systems. In Ralf Dörner and Detlef Krömker, editors, *Self Integrating Systems for Better Living Environments: First Workshop, Sensyble 2010*, pages 15–19. Shaker Aachen, November 2010.

[Lip09] Michael Lippautz. *Location-Aware Complex Event Processing in Mobile Environments*. PhD thesis, Salzburg University of Applied Sciences, 2009.

[LS11] David Luckham and Roy Schulte, editors. *Event Processing Glossary - Version 2.0*. Event Processing Technical Society, July 2011. http://www.complexevents.com/2011/08/23/event-processing-glossary-version-2-0/.

[Luc02] David Luckham. *The Power of Events: An Introduction to Complex Event Processing in Distributed Enterprise Systems*. Addison-Wesley Longman, Amsterdam, 2002.

[Mic06] BM Michelson. Event-driven architecture overview. *Patricia Seybold Group*, 2006.

[Sch10] Jan Schaefer. Towards a Platform for Self-Organizing AAL Applications. In Ralf Dörner and Detlef Krömker, editors, *Self Integrating Systems for Better Living Environments: First Workshop, Sensyble 2010*, pages 109–116. Shaker Aachen, November 2010.

[TGNO92] Douglas Terry, David Goldberg, David Nichols, and Brian Oki. Continuous queries over append-only databases. *ACM SIGMOD Record*, 21(2):321–330, June 1992.

[TGS06] Edward Tse, Saul Greenberg, and Chia Shen. Motivating Multimodal Interaction around Digital Tabletops. In *Video Proc. ACM CSCW Conf, Computer Supported Cooperative Work*, pages 6–7, 2006.

[WEH08] Aaron Wolin, B Eoff, and T Hammond. ShortStraw: A Simple and Effective Corner Finder for Polylines. *5th Annual Workshop on SketchBased Interfaces and Modeling*, pages 33–40, 2008.

[WX10] Lin Weiguo and Jin Xin. A sketch recognition algorithm for Pen-based Human-Computer interaction. In *International Conference on Computer Application and System Modeling (ICCASM), 2010*, volume 2, pages 248–251. IEEE, 2010.

Perception-influenced Animation

Daniel Schiffner, Detlef Krömker
Professur für Graphische Datenverarbeitung
Goethe Universität Frankfurt
Robert-Mayer-Str. 10
60054, Frankfurt (Main)
{kroemker,dschiffner}@gdv.cs.uni-frankfurt.de

Abstract: The heterogeneity of future living environments will increase the necessity to create applications that can run on any device. In the context of graphics applications, some kind of simplification must be included to enable rendering on devices with less computation power. Using perception to guide such a simplification is a common approach. However, existing methods generate levels of detail in advance, and only a selection is performed during run-time. In simulations, this not sufficient because an object will change over time.

We present a framework that adapts a simulation using perceptual measures. We use a visual salience model to extract regions where detail can be modified. This information is calculated during run-time, and by using a dynamic data structure, the representation is adapted without a definition of levels of detail in advance. We included the system in a physics library and so created an interactive and continuous simulation level of detail.

1 Introduction

Physical behavior of objects in a 3d scene create the impression of a real-world scenario. Simulations of fabrics and liquids, for example, require expensive calculations, and if accuracy is not crucial, a reduction in the number of simulated objects leads to a decrease in computation time. In real-time applications, such as games, a reduction may be applied as long as the plausibility is retained, e.g. a human would rate the visible outcome as valid. In this work, we propose to use perceptual information when altering detail, and we present a framework to modify the detail of a simulation during run-time. This adaptation is not bound to a predefined set of simulation levels of detail (SLODs). Thus, it can adapt to any hardware without manual adjustments. With this system, it is theoretically possible to transfer a running simulation from a high-performance system to another, maybe lower-powered device. A user is no longer bound to a specific hardware at a distinct location.

By simulation, we thereby mean changes to the surface representation based on physical laws. These changes, for example, can be introduced due to gravitation or compression of an object. Usually, objects are decomposed into multiple parts to increase detail, i.e. accuracy, of the simulation. Our framework will control this decomposition, and to complete this task, perceptual measures are used to steer the applied modifications, e.g. a reduction or increase in detail.

To account for perceptual information, a visual salience model is used. An object is considered salient if it figuratively "pops out" of its surround. We utilize a 3d visual salience model and include animation-specific features, e.g. motion. The Bidirectional Saliency Weight Distribution Function (BSWDF) [SK11] allows us to extract regions of interest within a 3d scenario, and thus we can account for areas that are important for a human spectator. Due to the inclusion of animation-specific features, regions that are modified by the simulation are taken into account as well.

We derive a Model-View-Controller-system that extracts, alters, and displays simulated objects. Our prototype is based on a soft-body simulation. A soft-body, which covers materials like cloth or fabrics, is well suited for dynamic adaptation as individual parts of a soft-body-simulation can be removed to reduce the accuracy. Our results show that real-time updates can be achieved using nowadays hardware.

After giving this introduction, we continue to look at related work. In section 3, we present our framework. Afterwards, the saliency features are derived, and the according BSWDF is defined. In section 5, we give some notes regarding the complete system and present performance measures as well as achieved results in section 6. We close with a conclusion and present intended future work.

2 Related Work

Soft-body objects have a high computational demand, and these are neither rigid, fluid, nor gaseous. To reduce simulation complexity of fluids and gaseous materials, point-based animations have been studied by several researchers. Mueller et al. [MKN+04] presented a separated data layout to reduce the size of the simulation data. Adams et al. proposed the "adaptively sampled particle fluids" approach [APKG07]. Single nodes are collapsed or expanded to reduce the size of the simulation. Similar to Mueller et al., a separated data layout is chosen.

The accuracy of a simulation can be modified during run-time by using multi-resolution representations. Beaudoin and Keyser [BK04] replace simulations of plants with approximations. These approximations are generated during a preprocess, and individual parts of a plant are exchanged using a recursive algorithm. Visual artifacts are avoided with smooth transitions. This geometric approach is both a LOD and SLOD as the detail in the representation and the simulation is reduced. It, however, requires to traverse the representation each time it is accessed, and the recursive algorithm does not account for perceptual information.

The perception of a human spectator can be modeled with visual salience as done by Itti et al. [IKN98]. Their method represents the processes of early vision, which cannot be influenced by tasks or other cognitive-related factors. For rendering, saliency information can be used to preserve detail in important regions [SK10, FSG09, LVJ05].

In [SK10], we presented a dynamic data structure, which selects a representation for rendering using a priority value. Carmona and Froehlich [CF11] proposed a similar approach simultaneously, and they presented a theoretical optimal algorithm for priority selection.

This priority, for example, can be saliency information, such as curvature of the surface. For general computation of saliency within a 3d scenario, the BSWDF has been defined [SK11]. This allows to model visual salience of an object without explicit projection into 2d space. In combination with the *TreeCut*, a perception-based LOD is established.

Some user studies regarding simulation and perception have been performed by several researchers [YRPF09, GDO08, O'S05]. The results of the different experiments show that the precision of physics simulations can be reduced without loosing plausibility.

3 Approach

Our approach is a combination of the *TreeCut* data structure – presented in [SK10] – and a soft-body simulation provided by the Bullet physics library [Bul]. We have chosen the library to show, on the one hand, the universal applicability of our approach, and on the other hand, avoid the need to verify a stand-alone physics calculation.

To establish this combination, both the *TreeCut* and the physics library need to be extended, so that a dynamic exchange of information between both is possible. Furthermore, to account for saliency information, a specialized BSWDF will be defined, which uses animation-based features, such as motion. While the BSWDF is not limited to these animation features, we will focus only on these within this work.

Our framework is depicted in figure 1. The complete system is highly dynamic as the change introduced by the simulation invokes new changes in the *TreeCut* representation. We use the simulated data to derive both the visual output (View) and the influence of visual salience computation (Feedback Stage). We control the Feedback Stage of the system with a threshold value for saliency values, and so limit the adaptation of simulation detail.

3.1 Physics Simulation

The Bullet physics library [Bul] provides various tools and data structures to compute a physics-based simulation. This includes detection and resolving of collisions as well as physics objects, e.g. rigid- and soft-bodies, as well as their computations

In our prototype, the Bullet library in version 2.78 is used. It provides a soft-body simulation class, which is supplementary to the normal library. We leverage this fact to plug-in our own *TreeCut*-SoftBody (TC-SB) that bases on the Bullet's `SoftBody`-class. It utilizes a Mass Spring System (MSS) to simulate the physical behavior. Only two data structures – the simulation nodes and links – are required for the computation of forces.

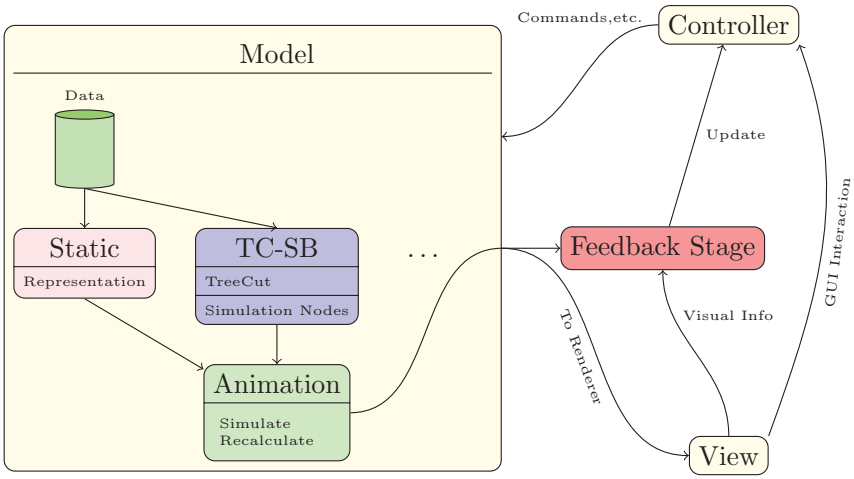
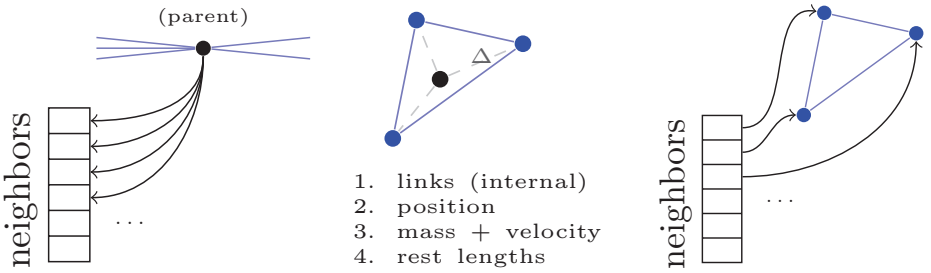


Figure 1: The proposed animation framework that is based on a Model-View-Controller-design. The Model contains the dynamic representation using a *TreeCut* and a soft-body simulation (TC-SB). Other representations can be added as well, which will be included in the animation. The Controller issues the *TreeCut*-operations based on the results from the Feedback Stage. The View is derived from the current representation of the Model and can introduce, via the user, new changes into the system.

3.2 *TreeCut*

The *TreeCut* utilizes a multi-resolution representation of an object. With the help of two core operations, `refine` and `coarse`, the detail of a local node is altered. In case of a `refine`-operation, a single node is replaced with a more-detailed representation. Assuming a tree, a node is replaced with its children. The `coarse`-operation is the inverse, i.e. the children are replaced with their common parent. We use a priority-selection of surface elements, so called surfels, to invoke a *TreeCut*-operation for the assigned node.

The *TreeCut* has been defined for geometrical LOD-methods, and thus both operations need to be extended to correctly account for SLOD-operations. Physics calculations need access to the surfel structure maintained by the *TreeCut*. We therefore use an index-based mapping between the surfels and the simulation nodes. The MSS-simulation uses links, and the inner structure of a mesh needs to be reestablished after a SLOD-method has been applied. Because of the locality of the *TreeCut*-operations, we include an incidence list, which avoids searching for connected links. An interpolation between the old and the new state of the MSS is done to reduce artifacts.



(a) Extract the neighbors that are adjacent to the parent node via the links. Also, mark these links for interpolation.

(b) Set the simulation parameters. Internal links (between children) are created. Masses, velocities, and rest lengths are calculated. Δ is the added displacement.

(c) Connect external nodes using the nearest node in the neighbor list. Mark these links for interpolation.

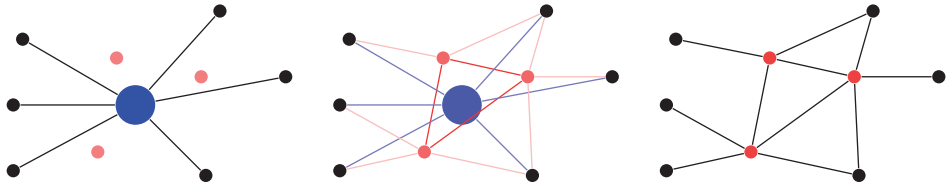
Figure 2: The `refine`-operation and the required steps to assure correct generation of simulation nodes. In the first step, the incident links are extracted while the physical properties are propagated to the child nodes in the second. In the third, the external links are distributed among the children and interpolation is enabled.

3.3 Beginning TreeCut-Operations

In the following, we assume a `refine`-operation to be processed – only small adaptations are required to perform a `coarse`-operation. To replace a simulation node with its successors, the following steps are performed:

1. Store (old) incident links
2. Propagate physical properties
3. Create (new) incident links
4. Mark links for interpolation

When `refine`-ing a node, the incident links are marked for interpolation and adjacent nodes are extracted. The physical properties of the parent node are acquired and propagated. For the position information of the children, a displacement relative to their parent is calculated. The child nodes are added to the set of simulation nodes, but are excluded from collision calculations, for now. After propagation, the new nodes are connected with their neighbors using local nearest neighbor search among the adjacent nodes. The created links are added to both incident link lists of the affected nodes and are marked for interpolation. In figure 2, the applied steps for propagation are visualized while figure 3 shows the interpolation.



(a) The current links to the node (blue) will be marked for delete and are interpolated due to the `refine`. The new nodes (red) will be connected with the neighboring nodes.

(b) The influence of the old links (blue) is reduced during the interpolation while the new links (red) gain importance. Inner links are excluded from blending.

(c) The simulation nodes after the applied `refine`-operation. The old nodes and links have been removed and the new links are no longer interpolated.

Figure 3: Application of the blending during the *TreeCut*-operations. It is performed to avoid the generation of artifacts. From left to right, a `refine`-operation is performed. From right to left, it is equivalent to the operations performed by a `coarse`-operation. In the captions, the `refine`-operation is explained.

3.4 Ending TreeCut-Operation

After completion of an interpolation, old nodes and links are removed from the simulation. We apply the following steps to complete a *TreeCut*-operation:

1. Finalize physical properties of the new nodes (e.g. insert collision shapes)
2. Delete old links (in sequential order)
3. Delete old nodes

First, the physical properties of the new nodes are finalized. Therefore, the collision shapes are inserted into the simulation. The incident links are no longer interpolated.

The incident link list of the old nodes is used to extract the affected links that will be deleted. To safely remove links, their internal order may not be altered. Otherwise, the calculated position will differ because links are evaluated sequentially. Therefore, a linear traversal is required for deletion, increasing the theoretical time complexity of the end-algorithms from $O(1)$ to $O(L)$ where L is the number of links. However, these links will be deleted in a lazy manner, and thus the overall time for computation is not influenced. After marking the links for removal, the nodes are removed as well.

4 Animation Features and Saliency

Animation features are accounted for by extracting local and global motion information that is generated during the simulation. These features will be stored in the surfel structure.

This allows combination with visual features as the BSWDF can be calculated during rendering.

The result of the BSWDF is a priority value that reflects the relative importance of a surfel. The *TreeCut*-evaluators use this priority to perform a reduction in simulation detail.

Definition of the BSWDF

Local motion is the relative motion of a node with respect to its previous position, i.e. its velocity. If a single node is moved differently than others, it is considered salient. This definition matches the processing of the human visual system (HVS). A `refine`-operation applied to that node will increase detail. In regions where no explicit nodes are found, the `coarse`-operation can safely be applied because it is unimportant for the HVS.

Global motion increases the saliency values as movable objects catch one's attention. However, this increase simultaneously limits the ability to focus an object [YPG01]. Thus, the maximal saliency value is clamped to an upper bound influenced by object's velocity.

We define a BSWDF that operates on both motion features. Global and local features are separated because the global features influence the local ones. Because of the restriction to two motion-related features, we state that other features need to be included when performing visual tests. In this special case, however, we give the following definition:

$$\text{BSWDF}(\omega_C, \omega_L, d_C, \vec{x}) = \text{Illu}(\omega_L, \vec{x}) \circ \text{Animation Features}(\omega_C, d_C, \vec{x}) \tag{1}$$

with ω_L being the light, ω_C the camera in spherical coordinates and d_C the distance to the camera. The $\circ$-operator applies the illumination model. In case of a Lambertian illumination, this is a multiplication. We define the Animation Features as

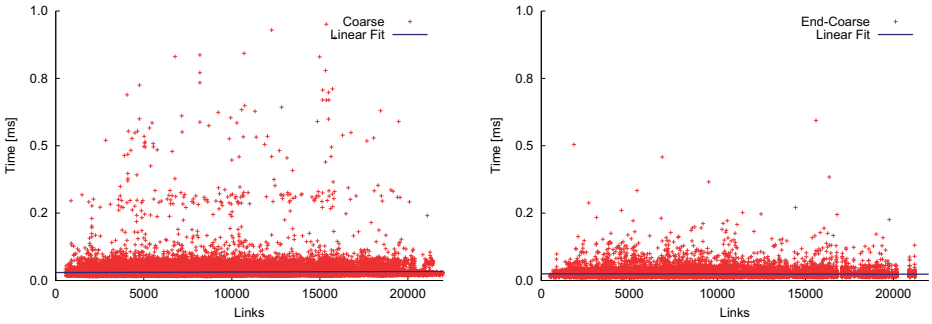
$$\text{Animation Features}(\omega_C, d_C, \vec{x}) = \text{Global Motion}(\omega_C, d_C) \otimes \text{Local Motion}(\omega_C, d_C, \vec{x}) \tag{2}$$

The position $\vec{x}$ is the current surfel's position assigned to a simulation node. If no illumination is used, the BSWDF simply evaluates to the result of the Animation Features. The $\otimes$-operator expresses the before mentioned limitation of the Local Motion features due to the Global Motion.

5 Complete System

The animation framework, as shown in figure 1, contains the physical calculations as part of the Model. This allows to include the extensions into an existing MVC-design to provide both physical simulations and perceptual evaluation.

The Feedback Stage receives input from the Model and the View to calculate the BSWDF.



(a) The `coarse`-operation

(b) The end-`coarse`-operation

Figure 4: The time complexity of the `coarse` and end-`coarse`-operations applied by the *TreeCut*-SoftBody. A linear fit is shown that uses all generated samples. For the `refine`, similar timings are achieved.

The *TreeCut*-evaluation is performed in parallel, and the according changes are sent to the Controller. This executes the `refine`- and `coarse`-operations.

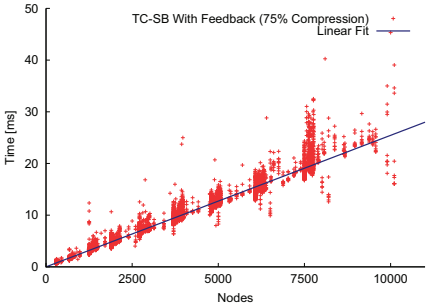
To allow adaptation of an object, some kind of restriction has to be imposed. In our prototype, we use a maximal node count, but also a maximal computation time or hardware capabilities can be utilized. For example, a maximal calculation time allows to adapt the simulation to achieve interactive rendering.

The complete system is highly dynamic and provides self-optimization capabilities. We include a threshold during *TreeCut*-evaluation to avoid repetitive change of a single node. For example, a `coarse`-operations could remove a node, which will be inserted in the next iteration again. With the threshold, the gain of inserting a node must be higher than the penalty of removing another node including the threshold.

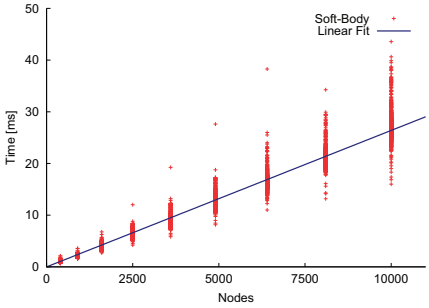
6 Performance and Results

We have implemented a prototype of the proposed system in `C++`. A cloth object is generated using a regular grid and the required LOD-hierarchy is generated in a preprocess. The corners are fixed to create a swinging net. In the tests, only gravitation is simulated. All results were taken on a system with an Intel i5 670 at 3.47 GHz, 8.0 GB RAM and a nVidia GeForce 260 GTX graphics card.

In a first test, we measured the performance of the dynamic SLOD-methods. In table 1, the manipulation of a single node is presented averaged over multiple applications of each *TreeCut*-operation. Each entry in the table contains the fraction of the complete processing time along with its measured times.



(a) *TreeCut*-SoftBody-simulation with Feedback enabled. The simulation is compressed to 75% of the initial size.



(b) `Soft-Body`-simulation. No compression can be applied, and thus the node count is constant.

Figure 5: The performance of the simulation in dependency of the node count used. The graphs plot the enhanced *TreeCut*-SoftBody and the plain `SoftBody`, i.e. where no compression is applied. The results are taken over a complete test set.

The *TreeCut*-operations have a high overall performance. Due to the neighbor information, both *TreeCut*-operations remain constant in time (see figure 4).

In a second test, the simulation time in dependency of the number of simulation nodes and links is evaluated. A test set is defined, which consists of multiple nets with varying node counts and compression rates. During each test, no collisions are performed.

The averaged timing result (see table 2) show that the *TreeCut*-SoftBody decreases the node count and increases performance. In the results, a reduction in processing time to approximately 47.76% is achieved (compression to 30%) despite the additional 25 interpolations performed each step. Figure 5 shows the linear dependency in node count of the *TreeCut*-SoftBody.

In figures 6a to 6j, some images generated with the *TreeCut*-simulation are shown. For comparison, the results achieved with the Bullet's `SoftBody` are included. The default `SoftBody` does not have the ability to change the SLOD. If a lower detailed version is required, a new `SoftBody` has to be created accordingly.

Operation	N	L	Node-Ops [ms]	Link-Ops [ms]	Complete [ms]
`refine`	4888	10419	0.0164 (91.14%)	0.0016 (8.94%)	0.0180
end-`refine`	4889	10794	0.0017 (94.39%)	0.0001 (5.53%)	0.0018
`coarse`	4966	10615	0.0319 (97.61%)	0.0007 (2.39%)	0.0327
end-`coarse`	4938	10917	0.0025 (99.20%)	0.000 (0.79%)	0.0025

Table 1: Different timing results when applying the *TreeCut*-operations to the simulation. N denotes the average number of nodes present while L is the number of links. The Node-Ops are all operations that affect a node of the simulation while Link-Ops modify its links.

Method	Compression	N	L	I	Total [ms]
Bullet SoftBody	-	4266	8413	0	11.09
TC-SB	NoCompression	4266	8647	0	10.08
	75%	3324	6931	30.5	8.40
	30%	1911	4220	25.1	4.80

Table 2: Timing results with and without the feedback-loop using varying node counts. Compression is the fraction the initial size is reduced to. N denotes the node count, L the number of links present and I the number of interpolations. All results are averaged over a complete test set.

7 Conclusion and Future Work

The presented system simulates a soft-body object, and the number of simulation nodes can be reduced during run-time without a definition of discrete levels in advance. As opposed to existing methods, we do not require a separated data layout. A SLOD-reduction is applied based on visual salience with animation-specific features. The representation can adapt to any hardware.

Unlike full point-based approaches, the removal of simulation nodes may not be performed without preconditions. We store incident links to circumvent restrictions and enable dynamic adaptation using the *TreeCut*.

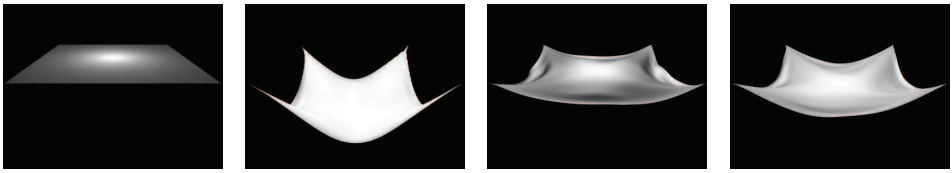
We defined a special BSWDF that operates on animation features, such as local motion. The results are used to control the dynamic adaptation via the *TreeCut*-evaluation. Because of the BSWDF, a human-oriented adaptation is performed.

Currently, only one *TreeCut* is applied for both rendering and simulation. We plan to apply a second *TreeCut*, which will maintain the simulation nodes. It will then be necessary to propagate the results of the simulation nodes to the surface representation.

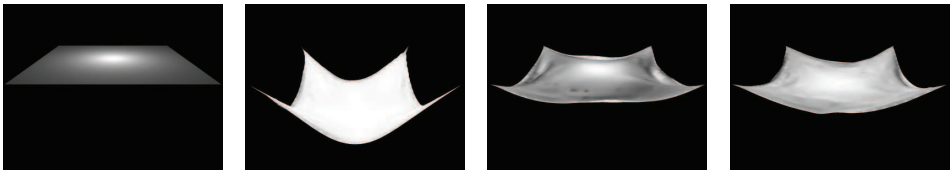
When applying a *TreeCut*-operation, the placement of new nodes is performed using only geometrical measures. This needs to be extended with animation and physical measures to increase stability and feasibility. Also, a smooth surface, e.g. a moving least squares surface, could provide more exact positions.

An implementation using the graphics card could increase performance. With the capabilities of the newest shader model 5, a soft-body can be implemented without restrictions. However, it remains to be seen what changes to the proposed system are required.

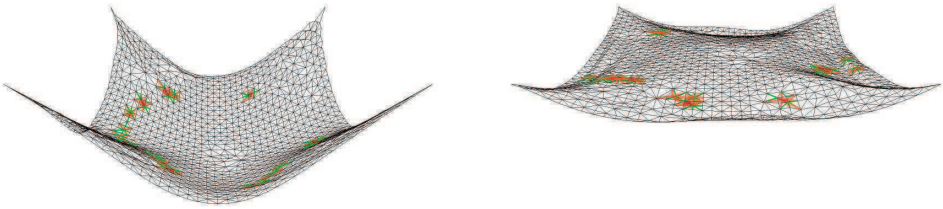
Our system can account for perception when evaluating a simulation object. User tests have to be performed to validate the gained impression that detail is preserved and that the simulation remains plausible even if nodes are `coarse`ned or `refine`d. A maximal compression factor could be derived to identify when a simulation looses its plausibility.



(a) `SoftBody` initial configuration (b) After 45 steps (c) After 90 steps (d) After 180 steps



(e) *TreeCut* initial configuration (f) After 45 steps (g) After 90 steps (h) After 180 steps



(i) *TreeCut* links after 45 steps (j) After 90 steps

Figure 6: Comparison of the `SoftBody` and the *TreeCut*. A fixed time step is used to generate the different configurations. In the last row, the links of the simulation are visualized. The highlighted links are being replaced by the *TreeCut*-operations. In case of the *TreeCut*-SoftBody, only an approximate surface reconstruction is applied. Yet, the detail is retained in the fast moving center region.

References

[APKG07] Bart Adams, Mark Pauly, Richard Keiser, and Leonidas J. Guibas. Adaptively sampled particle fluids. *ACM Transactions on Graphics*, 26(3):48:1–48:8, 2007.

[BK04] Jacob Beaudoin and John Keyser. Simulation levels of detail for plant motion. In R. Boulic and D. K. Pai, editors, *Computer animation 2004*, pages 297–304, Aire-la-Ville, 2004. Eurographics Association.

[Bul] `http://bulletphysics.org/`. visited on 19.12.2011.

[CF11] Rhadamés Carmona and Bernd Froehlich. Error-controlled real-time cut updates for multi-resolution volume rendering. *Computers & Graphics*, 35(4):934–944, 2011.

[FSG09] Miquel Feixas, Mateu Sbert, and Francisco Gonzalez. A unified information-theoretic framework for viewpoint selection and mesh saliency. *ACM Transactions on Applied Perception*, 6(1):1:1–23, 2009.

[GDO08] Marcos Garcia, John Dingliana, and Carol O'Sullivan. Perceptual Evaluation of Cartoon Physics: Accuracy, Attention, Appeal. In Sarah Creem-Regehr, K. Myszkowski, Bobby Bodenheimer, Betty J. Mohler, Bernhard Riecke, and Stephen N. Spencer, editors, *Proceedings APGV 2008*, pages 107–114, New York, 2008. ACM Press.

[IKN98] Laurent Itti, Christof Koch, and Ernst Niebur. A model of saliency-based visual attention for rapid scene analysis. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(11):1254–1259, 1998.

[LVJ05] Chang Ha Lee, Amitabh Varshney, and David W. Jacobs. Mesh saliency. In Markus Gross, editor, *Proceedings of ACM SIGGRAPH 2005*, pages 659–666, New York, 2005. ACM.

[MKN+04] Matthias Mueller, Richard Keiser, Andrew Nealen, Mark Pauly, Markus Gross, and Marc Alexa. Point based animation of elastic, plastic and melting objects. In R. Boulic and D. K. Pai, editors, *Computer animation 2004*, pages 141–151, Aire-la-Ville, 2004. Eurographics Association.

[O'S05] Carol O'Sullivan. Collisions and Attention. *ACM Transactions on Applied Perception*, 2(3):309–321, 2005.

[SK10] Daniel Schiffner and Detlef Krömker. Tree-Cut: Dynamic Saliency Based Level of Detail for Point Based Rendering. In Ralf Dörner and Detlef Krömker, editors, *Self Integrating Systems for Better Living Environments: First Workshop, Sensyble 2010*, pages 37–43. Shaker Aachen, 2010.

[SK11] Daniel Schiffner and Detlef Krömker. Three Dimensional Saliency Calculation Using Splatting. In Nenghai Yu, editor, *Sixth International Conference on Image and Graphics (ICIG), 2011*, pages 835–840, Piscataway, 2011. IEEE.

[YPG01] Hector Yee, Sumanta N. Pattanaik, and Donald P. Greenberg. Spatiotemporal sensitivity and visual attention for efficient rendering of dynamic environments. *ACM Transactions on Graphics*, 20(1):39–65, 2001.

[YRPF09] Thomas Y. Yeh, Glenn Reinman, Sanjay J. Patel, and Petros Faloutsos. Fool Me Twice: Exploring and Exploiting Error Tolerance in Physics-Based Animation. *ACM Transactions on Graphics*, 29(1):5:1–11, 2009.

Towards a Top-View Detection of Body Parts in an Interactive Tabletop Environment

Nadia Haubner, Ulrich Schwanecke, Ralf Dörner, Simon Lehmann, Johannes Luderschmidt

Design Computer Science Media
RheinMain University of Applied Sciences
Unter den Eichen 5
65195 Wiesbaden
nadia.haubner@hs-rm.de
ulrich.schwanecke@hs-rm.de
ralf.doerner@hs-rm.de
simon.lehmann@hs-rm.de
johannes.luderschmidt@hs-rm.de

Abstract: Integrating digital tabletop systems in private living environments is a promising approach to enhance people's everyday life with information technology. Apart from using the surface of such a tabletop, research on the detection of interaction above and around the surface is increasing rapidly. So far, detection is limited either to very specific gestures above the surface or to rather abstract detection of users in a larger scenario. The detection of body parts in tabletop setups has rarely been investigated, although the knowledge about the whereabouts of body parts would be helpful to establish relationships between users and interactions. In this paper, we propose a system that is capable to detect body parts above and around such a tabletop setup using a depth camera. We further take up an existing approach to present how the detection in this setup could work. Additionally, we propose a new approach to obtain training data for the detection using a color suit.

1 Introduction

Ubiquitous Computing (Ubicomp) [Wei91] promises to enrich people's everyday environments, such as private living spaces, with technology, in a way that people are surrounded by information technology anywhere in their home. In such private IT-spheres the human factor has to be considered from two points of view. First, information about people, such as their whereabouts, physical and emotional condition or current activities can be gathered with a diversity of sensors. On the one hand, this information can be exploited to configure and organize the IT-sphere and on the other hand increases the importance of handling this sensitive data securely. Secondly, people may intend to access the system and intervene in automatic configurations processes. In private living environments, people cannot be expected to be expert users regarding these intended interactions requiring an intuitive user interface. As a further requirement, user interfaces should be integrated

in the environment as naturally as possible to keep a homelike atmosphere.

In [Dou01] the concept of *embodied interactions* is introduced, meaning seamlessly integrated technology in peoples everyday life. Regarding horizontal surfaces of our everyday life, such as dining and coffee tables or cooking fields, a possible consequence to apply the Ubicomp concept is to replace these conventional surfaces by interactive surfaces. In this way, surfaces can be used both as traditional furniture or home appliances and as a means to interact with digital content.

Especially the field of interactive tabletops is topic of a lively research community and is growing further. Much effort has been put into researching all kinds of gestures or combinations of input methods *on* the surface to realize natural interactive systems [Saf09]. However, the main means of interaction are still touch and tangible objects. The space *above and around* the surface has also gained attention of researchers. They mostly experiment with free gestures of hands to be used as a separate means of interaction [LAM07, KGS07]. While several approaches have been developed, interaction techniques above and around the surface are very specific and mostly considered as isolated gestures rather than integrated in a larger scenario.

The concept of proxemic interaction [BMG10] ties in with Ubicomp environments by considering the information about relationships of people and devices, such as distance, orientation, movements and identity, to regulate implicit and explicit interaction techniques. In contrast to the above mentioned isolated gestures, proxemic interaction takes place at a higher abstraction level in a larger area, e.g. a living room. Yet, the concept only considers orientation and location of the whole body. However, when it comes to interaction in tabletop setups, a more detailed knowledge of body parts is necessary to allow for interaction techniques in a continuous interaction space [MJGJ11]. Additionally, the knowledge of the whereabouts of body parts becomes particularly helpful in multi-user scenarios to assign interactions to specific hands and users [DL01].

In this paper, we claim that an unobtrusive top-view system to detect body parts in a tabletop setup is helpful in an Ubicomp environment. One contribution of this approach is that intended human computer interaction is enhanced with a natural user interface. Further, self organizing and security systems can be supplied with dedicated information about users whereabouts. We propose an approach how the tabletop system could be achieved by employing a depth camera. We discuss the present constraints of the system and we further take up previous work which introduced a full body human pose recognition without using any kinematic or temporal dependencies [SFC+11]. A further contribution of this paper is a new approach to retain training data using a color suit.

In the remaining, we first review related work. Secondly, we describe a setup and the precision that is to be expected to detect above and around the surface interaction. Based on this we present an approach to detect interaction in this setup and propose a novel way to obtain training data. Finally, we give a conclusion and propose future work.

2 Related Work

The field of interactive surfaces is topic to a lot of research groups. In particular, the detection of interaction around the surface has gained attention recently. Usually, the approaches either investigate the space above horizontal surfaces, the space in front of vertical surfaces or a combination of both. Our approach also relates to previous work, that proposes systems to estimate the human pose and detect body parts from depth data. In the following, we first review previous work that focuses on extending interactive surfaces with depth. Secondly, we give an overview of approaches to detect body parts.

2.1 Extending Interactive Surfaces with Depth

In [HIW+09], Hilliges et al. present two different rear projection-vision tabletop setups to detect interaction above the surface. In one setup the height of hands above the surface is approximated by their brightness in the image using a standard camera. In the second setup a depth sensing camera is employed from behind a holographic screen to detect the exact height of hands above the surface. However, only specific interaction, e.g. a pinch gesture above the surface is proposed. Detecting human body parts is not subject of their work.

In [Ben09], Benko et al. introduce *DepthTouch*, where a vertical screen is combined with a depth camera to allow for interaction in front of the screen. The interaction space is limited to a defined area where hand blobs are detected as the closest segments to the camera. Therefore, the approach is not applicable in a top view setup. Furthermore, only one user can interact at a time which suspends multi-user scenarios.

Wilson et al. present *LightSpace* [WB10], a combination of a horizontal and vertical digital surface, multiple projectors and depth cameras. In their setup, they allow for carrying virtual object on the user's hand. However, they do not distinguish between body parts but treat any part of the mesh representing the detected surface of the user equally.

In [TMR10], Takeoka et al. present Z-touch which uses multi-layered infrared laser planes to detect interaction close to a tabletop surface. The system does not scale for larger interaction volumes above the surface and is unable to assign interaction to actual body parts. Annett et al. propose Medusa [AGWF11], also an extensive hardware setup composed of a digital tabletop and three rings of infrared-based proximity sensors, to detect the presence of users close to the tabletop using these sensors. Employing this technology, they can assign touches on the surface to hands of users. The authors point out, that their system is error-prone in situations when multiple users cross paths. Furthermore, accurate positions of hands above the surface or distinct body postures can not be determined.

2.2 Detecting Body Parts from Depth Data

There exists a vast body of literature on vision based approaches to realize markerless human motion detection, reviewed in [Pop07]. Since our approach is based on acquired depth data, we will give a short overview of work that focuses on the detection of body parts that involves depth data.

Several approaches employ a fusion of RGB and depth data and a human body model to detect and track a user's body parts. Knoop et al. use an iterative closest point (ICP) algorithm to fit the body model into the data in [KVD09]. Apart from kinematic constraints, no other assumptions are made. However, the authors point out, that the approach only works reliably, if the whole body is visible. Further, a higher tracking accuracy comes at the cost of a growing number of ICP steps, which increases the algorithms running time. Jain et al. propose a haar cascade based detection of head and torso followed by a locally fitting of limbs in [JSDM11]. The approach uses a frontal face detection classifier to estimate the head's position and then infer the upper body pose.

With the launch of the Kinect[1], the OpenNI SDK[2] has become available to realize a full body tracking in real-time by fitting a human skeleton into depth data. However, the system only works, if the whole user's body is visible and standing upright.

In [PGKT10], Plagemann et al. propose an approach to detect human body parts in depth data by identifying geodesic extrema on the surface mesh. These extrema are classified as head, hands and feet using a classifier trained on depth images patches. To overcome self-occlusion problems occurring in this approach, Schwarz et al. [SMMN11] employ optical flow to disconnect points that lie on different body parts. To correctly assign body parts, both approaches assume that the user is facing the camera to achieve a frontal view.

All previous approaches have in common that they all propose front view setups which assume certain situations. Among others, assumptions are that the full body is visible, that the head is above other body parts, that arms are reaching towards the camera or that the face is visible in the camera image. None of these assumptions can be made in a top view scenario. As a consequence, the so far reviewed approaches are not applicable in our setup.

Shotton et al. [SFC+11] present an approach to detect body parts from depth data without using any temporal or kinematic dependencies. Although they propose a front view setup where the whole body is visible, they do not make any assumptions that limits their approach to such setups. We therefore take their work up and propose to employ it in a top view setup to detect body parts.

[1] http://www.xbox.com/kinect
[2] http://www.openni.org

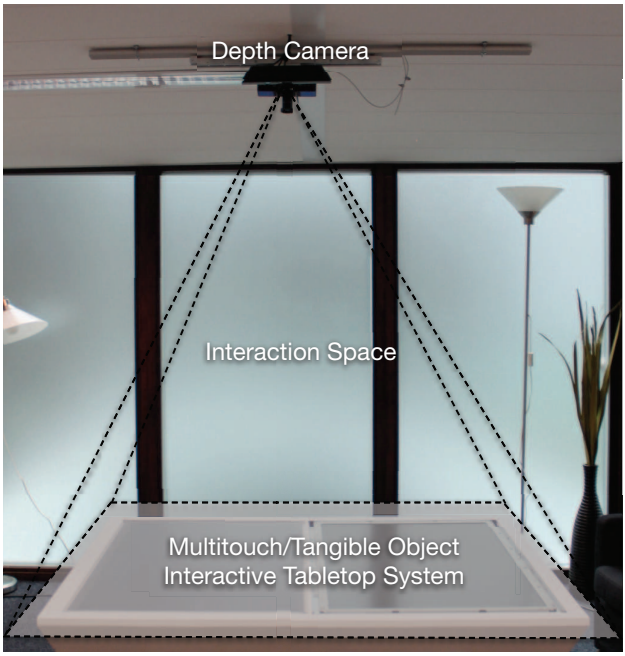


Figure 1: Overview of our setup.

3 System

In this section we propose a system to detect user input above and around a digital tabletop. We first present the setup before we describe the precision that is to be expected.

3.1 Setup

There exist various tracking solutions using markers that are attached to the user's body. Although these approaches achieve a very high tracking accuracy, these systems come at a cost. Firstly, the hardware tends to be expensive and often needs the installation of multiple cameras. Secondly, the severest drawback is that users must wear markers whenever they intend to interact with the system. That is not desirable in a private living environment and we therefore choose a highly preferable markerless vision based solution.

Since interaction takes place in 3D space, 3D information of the scene has to be extracted first. Although a setup with multiple cameras would avoid occlusion situations, the system would need an extensive hardware installation. A passive stereo-vision system could be used to obtain a depth image of the scene. However, these systems lack in reliability because corresponding points need to be found in every frame which can be difficult, e.g., in homogeneous regions. We therefore employ a single active depth camera that

uses infrared light to measure depth since this technology can achieve robust results at interactive frame rates.

In a setup where only a single depth camera is present, the position and orientation of the camera has to be chosen carefully. In a tabletop setup, interaction is likely to be performed from all sides of the table. Installing the depth camera to obtain a side-view of the scene would limit the system to only be usable from three sides. Furthermore, occlusions in multiple user scenarios might occur. Placing the camera inside the table would solve occlusion issues, but is not applicable with many tabletop system since rear projecting screens reflect infrared light. Therefore, we install the depth camera above the table to achieve a top view of the scene.

A suitable active depth camera we could use in our setup is a Time-of-Flight (ToF) camera. ToF cameras are robust sensors to obtain depth images of the scene at interactive frame rates by measuring the time that the emitted infrared light needs to return to the sensor [KBKL09]. However, ToF cameras are still expensive and provide a rather low image resolution. Furthermore, ToF cameras cannot be used with infrared illuminated tabletop systems since the camera's infrared light interferes with the tabletop systems and makes touch and tangible object recognition impossible by outshining the table's infrared light in the camera image.

Microsoft's Kinect determines the depth from a disparity map which is obtained by an infrared light pattern that is projected onto the scene. It captures a high resolution image at 30 frames per second. The pattern only causes minor artifacts in the camera image of an infrared illuminated table which can easily be removed with standard filtering techniques. Further, the Kinect is not troubled by the table's illumination during the detection of interaction above and around the surface.

Based on the observations from above, our proposed system concludes in a top-view setup with the Kinect as shown in Figure 1. We investigate the scenario of a living room, where the coffee table is equipped with a digital surface with a height of approximately 55 cm. With a ceiling height of 2.50 m, the camera is installed 1.88 m above the surface and captures the whole surface of the table and an additional zone of around 30 cm around the surface. This setup allows for detecting interaction that takes place above and closely around the tabletop system.

3.2 Constraints

To retrieve depth data, we use the Open Source library libfreenect provided by the OpenKinect project[3]. We obtain an depth image with 11 bit depth where 1 bit marks error pixels, thus 10 bit encode the actual depth. The project also provides an approximation for converting the raw disparity values r into the metric system. The approximation is given by $d = 1/(-0.00307 \cdot r + 3.33)$ where d is the depth value in meters and $r = 0 \ldots 1023$. This results in a visible range between 0.3 and 5.30 meters with a depth resolution that ranges from 3 mm to 844 mm respectively. Since the depth camera is installed 1.88 meters above

[3] http://openkinect.org

Distance	10 cm	20 cm	30 cm	40 cm
points/cm^2	11.1	12.7	14.5	16.3

Table 1: Average number of points per cm^2.

the digital surface we can rely on a depth resolution of at least 11 mm in our setup.

To evaluate the depth precision with respect to our tabletop setup we captured a planar cardboard at four different distances d to the surface. Table 1 shows the xy-resolution of the raw data as the average number of points per cm^2 for the four distances. The data is averaged over 100 frames. In the closest distance to the table we obtain approximately 11 points per cm^2 which restricts the minimum size of an object that can be captured theoretically to 3.3 mm $\times$ 3.3 mm.

These technical limitations have to be taken into account when designing the body detection system. For instance, fingers will probably not robustly be detectable with our system since noise due lighting conditions and fast movements is to be expected.

4 Detection of Body Parts

In this section we propose a system to detect body parts in the previously described tabletop setup. Since we build on previous work, we will first give a short summary of that approach before we introduce a novel way to obtain training data using a special garment that is specifically colored.

4.1 System Overview

In [SFC+11], Shotton et al. introduce an approach to quickly and accurately predict 3D positions of body parts and joints from depth data. They use neither temporal nor kinematic constraints and process each pixel independently, making the algorithm suitable for parallel implementation. Most importantly, no assumptions about the user's pose are made and it thus allows for 360° rotation of users and is able to detect multiple users at a time. We therefore adapt their approach for our top view tabletop setup.

Their system is composed of a training phase and a live detection of body parts. The first training step is to acquire depth data which contains body part labels. Based on this, depth image features are computed for each pixel independently. Theses features are combined in randomized decision trees where each leaf node represents a distribution over body part labels. In the detection phase, the foreground is segmented in the depth data first, before each foreground pixel is classified as a body part. Figure 2 shows an overview of the algorithm. For a more detailed explanation of the algorithm we refer to [SFC+11].

In their paper, Shotton et al. propose a full body detection, and therefore use 31 body parts that cover the whole body. In order to adapt the algorithm technique for our setup, we need

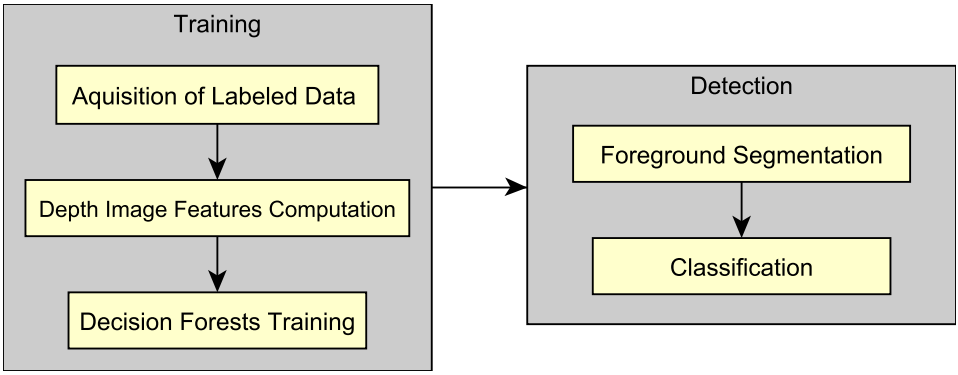


Figure 2: System overview of the algorithm presented in [SFC+11].

to define body parts that are feasible in a top view detection scenario. First, our setup sets constraints which we have discussed in section 3.2. Therefore, we do not aim to detect fingers but combine them with the palms. Further, we aim to detect the upper body from a top view where users bow over the surface to interact. This limits the visibility of the upper body and make it difficult to distinguish between areas of the user's back. We thus unite them to one body part. As a result, we propose to distinguish between 13 body parts as shown in Figure 3. These parts are composed of hands, lower and upper arms, torso and head and of important joints such as elbows, shoulders and neck.

4.2 Obtaining Training Data

In the above briefly reviewed approach [SFC+11] several training data is used and compared: Motion capture data, hand-labeled depth data and synthetically generated data. The authors found the synthetic test sets harder to classify since a higher variation in poses and body shapes is present. All of these data acquisition methods involve drawbacks. Firstly, motion capture systems are expensive and need a separate hardware setup. Furthermore, they do not cover the whole user's body but only capture those positions where markers are attached to the body. The remaining body parts need to be interpolated in some way which may differ from the real body parts. Hand-labeling depth data is clearly a cumbersome process especially when a high number of frames is necessary for reliable detection results. Finally, generating synthetic data is a quick and cheap alternative, yet it cannot cover the natural interaction space of users. On the contrary, it generates an overcomplete set of the space that users usually covers in a specific scenario, e.g. when sitting at a table, causing inefficiencies.

We therefore present a novel way to obtain training data using both RGB and depth data employing a color suit. Our approach is inspired by the work of Wang et al. [WP09] who use a color glove to recognize hand poses. The color glove is formed by twenty patches which are colored with ten different colors. While they obtain a database of hand-poses

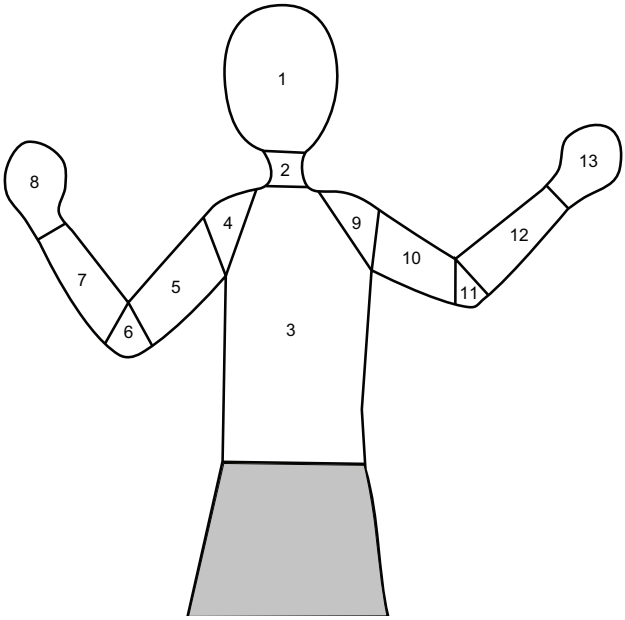


Figure 3: Schematic overview of the body parts to be detected.

with a dataglove they index these hand-poses with rasterized images of the glove in the respective poses. Users also have to wear the glove during the real-time tracking of hands.

We propose to use a color suit for the upper body where every body part is color coded with a unique color as shown in Figure 4. The suit is composed of a long sleeve shirt to cover upper body and arms, two gloves for hands and wrists and a balaclava for head and neck. Since the Kinect can obtain both RGB and depth data we are able to assign every depth value to a respective body part. To achieve this, we assume that the scenario is well-lit and that the background and other parts of the user, e.g. the legs, have low chrominance values. In this way, we achieve a cheap and easy to use motion capture system that is able to seamlessly determine body parts. The suit is only necessary in the training phase which allows users to wear their usual clothing to be detected while interacting with the tabletop system. Since we will use the same depth image features as described in [SFC+11], we assume, that the training process will have to be repeated if the observed setup is changed, e.g. the table is relocated or replaced. Regarding users, we claim that one training phase with users that sufficiently vary in weights and heights is also suitable for detecting body parts of any user that did not participate in the training process. A further benefit of our approach is that we use only one hardware setup. Thus, only a single setup has to be installed and both training and detection are based on the same depth data with respect to resolution and noise characteristics.

We had to choose the colors for the upper body suit from a set of available colors. We therefore determined a subset of 13 colors such that their intrinsic colors are equally distributed in the HSV color model. We carried out first tests to evaluate if the colors can be

Figure 4: Front and back view of the color suit.

distinguished reliably with the Kinect which encourage us to proceed with our work.

5 Conclusion and Future Work

In this paper we outlined a system that detects body parts in an interactive tabletop environment that would enrich private IT-spheres. In this context we have described a setup that is applicable to detect interaction from a top-view employing a depth camera.

We further have presented a novel approach to obtain training data for an existing body part detection using a color suit. First it seems to be reasonable that training the system works better with real data than with synthetic data because the natural motion space of users is covered. In addition to that, one can expect that using the same system to acquire train data and detect body parts increases the robustness of the system. Finally, only a single system has to be employed in the whole process, in contrast to training setups where a motion capture system is needed.

As future work, we have to evaluate the reliability of the proposed body part training approach using a color suit. Moreover, the data quality has to be compared with synthetic and motion captured training data. Additionally, the existing detection approach presented in [SFC+11] has to be evaluated in a top-view setup with respect to reliability and precision.

Finally, our approach implicates future work in other areas since large amounts of detailed data about users in their private living environment is produced. Firstly, this raises questions on how to deal with this sensitive data from a security point of view. In turn, the field of security can also benefit from the determined information to solve security issues, e.g. authorization processes. Secondly, self-organising systems can exploit the knowledge of the whereabouts of body parts in a Ubicomp environment.

References

[AGWF11] Michelle Annett, Tovi Grossman, Daniel Wigdor, and George Fitzmaurice. Medusa: a proximity-aware multi-touch tabletop. In *Proceedings of the 24th annual ACM symposium on User interface software and technology*, UIST '11, pages 337–346, New York, NY, USA, 2011. ACM.

[Ben09] Hrvoje Benko. Beyond flat surface computing: challenges of depth-aware and curved interfaces. In *Proc. of the 17th ACM int. conf. on Multimedia*, pages 935–944. ACM, 2009.

[BMG10] Till Ballendat, Nicolai Marquardt, and Saul Greenberg. Proxemic interaction: designing for a proximity and orientation-aware environment. In *ACM Int. Conf. on Interactive Tabletops and Surfaces*, pages 121–130. ACM, 2010.

[DL01] Paul Dietz and Darren Leigh. DiamondTouch: a multi-user touch technology. In *UIST '01: Proceedings of the 14th annual ACM symposium on User interface software and technology*, pages 219–226, New York, NY, USA, 2001. ACM.

[Dou01] Paul Dourish. *Where the action is: the foundations of embodied interaction*. MIT Press, Cambridge, MA, USA, 1. edition, 2001.

[HIW+09] Otmar Hilliges, Shahram Izadi, Andrew D. Wilson, Steve Hodges, Armando Garcia-Mendoza, and Andreas Butz. Interactions in the Air: Adding Further Depth to Interactive Tabletops. In *UIST '09: Proc. of the 22nd annual ACM Symp. on User interface software and technology*, pages 139–148. ACM, 2009.

[JSDM11] Himanshu Prakash Jain, Anbumani Subramanian, Sukhendu Das, and Anurag Mittal. Real-time upper-body human pose estimation using a depth camera. In *Proceedings of the 5th international conference on Computer vision/computer graphics collaboration techniques*, MIRAGE'11, pages 227–238, Berlin, Heidelberg, 2011. Springer-Verlag.

[KBKL09] Andreas Kolb, Erhardt Barth, Reinhard Koch, and Rasmus Larsen. Time-of-Flight Sensors in Computer Graphics. *Eurographics State of the Art Reports*, pages 119–134, 2009.

[KGS07] Raghavendra S. Kattinakere, Tovi Grossman, and Sriram Subramanian. Modeling Steering within Above-the-Surface Interaction Layers. In *Proc. of the SIGCHI conf. on Human factors in comp. sys.*, pages 317–326. ACM, 2007.

[KVD09] Steffen Knoop, Stefan Vacek, and Rüdiger Dillmann. Fusion of 2d and 3d sensor data for articulated body tracking. *Robot. Auton. Syst.*, 57:321–329, March 2009.

[LAM07] A. Lucero, D. Aliakseyeu, and J.-B. Martens. Augmenting Mood Boards: Flexible and Intuitive Interaction in the Context of the Design Studio. In *Horizontal Interactive Human-Computer Systems, 2007. TABLETOP '07. 2nd Annual IEEE Int. Workshop on*, pages 147 –154, oct. 2007.

[MJGJ11] Nicolai Marquardt, Ricardo Jota, Saul Greenberg, and Joaquim A. Jorge. The continuous interaction space: interaction techniques unifying touch and gesture on and above a digital surface. In *Proceedings of the 13th IFIP TC 13 international conference on Human-computer interaction - Volume Part III*, INTERACT'11, pages 461–476, Berlin, Heidelberg, 2011. Springer-Verlag.

[PGKT10] C. Plagemann, V. Ganapathi, D. Koller, and S. Thrun. Real-time identification and localization of body parts from depth images. In *Robotics and Automation (ICRA), 2010 IEEE International Conference on*, pages 3108 –3113, may 2010.

[Pop07] Ronald Poppe. Vision-Based Human Motion Analysis: An Overview. *Comput. Vis. Image Underst.*, 108:4–18, October 2007.

[Saf09] Dan Saffer. *Designing Gestural Interfaces*. O'Reilly Media, Inc., 1. edition, 2009.

[SFC+11] J. Shotton, A. Fitzgibbon, M. Cook, T. Sharp, M. Finocchio, R. Moore, A. Kipman, and A. Blake. Real-time human pose recognition in parts from single depth images. In *Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on*, pages 1297 –1304, june 2011.

[SMMN11] L.A. Schwarz, A. Mkhitaryan, D. Mateus, and N. Navab. Estimating human 3D pose from Time-of-Flight images based on geodesic distances and optical flow. In *Automatic Face Gesture Recognition and Workshops (FG 2011), 2011 IEEE International Conference on*, pages 700 –706, march 2011.

[TMR10] Yoshiki Takeoka, Takashi Miyaki, and Jun Rekimoto. Z-touch: An Infrastructure for 3d Gesture Interaction in the Proximity of Tabletop Surfaces. In *ACM Int. Conf. on Interactive Tabletops and Surfaces*, pages 91–94. ACM, 2010.

[WB10] Andrew D. Wilson and Hrvoje Benko. Combining Multiple Depth Cameras and Projectors for Interactions On, Above and Between Surfaces. In *Proc. of the 23rd annual ACM Symp. on User interface software and technology*, pages 273–282. ACM, 2010.

[Wei91] Mark Weiser. The Computer for the 21st Century. *Scientific American*, 265(3):66–75, January 1991.

[WP09] Robert Y. Wang and Jovan Popović. Real-time hand-tracking with a color glove. In *ACM SIGGRAPH 2009 papers*, SIGGRAPH '09, pages 63:1–63:8, New York, NY, USA, 2009. ACM.

CSECS 2012

1st International Workshop on Complex Sciences in the Engineering of Computing Systems

organized by

Ingo Scholtes, Eidgenössische Technische Hochschule Zürich, Switzerland
Claudio J. Tessone, Eidgenössische Technische Hochschule Zürich, Switzerland
Jacob Beal, BBN Technologies/ MIT CSAIL, USA

Forecasting Technological Innovation

Aimee Gotway Bailey[1,2], Quan Minh Bui[3,4], J. Doyne Farmer[4], Robert M. Margolis[5], Ramamoorthy Ramesh[1]

[1]U.S. Department of Energy
1000 Independence Ave. SW
Washington, District of Columbia 20585, United States

[2]American Association for the Advancement of Science
1200 New York Avenue NW
Washington, District of Columbia 20005, United States

[3]St. John's College
1160 Camino Cruz Blanca
Santa Fe, New Mexico 87505-4599, United States

[4]Santa Fe Institute
1399 Hyde Park Rd.
Santa Fe, New Mexico 87501-8943

[5]National Renewable Energy Laboratory
1617 Cole Blvd.
Golden, Colorado 80401-3305, United States

Abstract:
Using a database of sixty-two different technologies, we study the issue of forecasting technological progress. We do so using the following methodology: pretending to be at a given time in the past, we forecast technology prices for years up to present day. Since our forecasts are in the past, we refer to it as *hindcasting* and analyze the predictions relative to what happened historically. We use hindcasting to evaluate a variety of different hypotheses for technological improvement. Our results indicate that forecasts using production are better than those using time. This conclusion is robust when analyzing randomly chosen subsets of our technology database. We then turn to investigating the interdependence of revenue and technological progress. We derive analytically an upper bound to the rate of technology improvement given the condition of increasing revenue and show empirically that all technologies fall within our derived bound. Our results suggest the observed advantage of using production models for forecasting is due in part to the direct relationship between production and revenue.

Keywords: experience curve, learning curve, performance curve, technology evolution, innovation

1 Introduction and background

Technology forecasting is a pervasive tool in the fields of engineering, economics, management science, and public policy. Arguably, the most consequential applications rest at the intersection of these disciplines. Different strategies for forecasting technological progress have been proposed [Moo65, Wri36, KM06, KM08, God82, SSC00, Nor09]. Simple models that track a performance metric as a function of one or two explanatory variables are widespread. In this work, we use the term *performance curve* to describe such simple models, which we define very generally to be a model of some performance metric (here, unit price) as a function of some proxy for experience (such as time or production). Perhaps the most famous performance curve model is Moore's law [Moo65], which states that the technology improves exponentially with time. Moore proposed exponential improvement originally for the density of transistors on a chip but later found that the relationship held for many different metrics for progress, including unit price. Another widely used performance curve model today is a power law relationship between the unit price of a technology p and its cumulative production q. Specifically, $p \propto q^{-w}$, where the exponent w is the rate of improvement. This model is referred to as "learning-by-doing" or Wright's law after his seminal 1936 study on aircraft costs [Wri36]. A similar power law relationship betwen unit price and *annual* production was proposed by Goddard [God82]. Alternative hypotheses utilizing time [KM06, KM08] and combinations of time, annual production, and cumulative production [SSC00, Nor09] also exist in the literature.

Performance curves aggregate all sources of price change, including but not limited to changes in input prices, economies of scale, labor learning, product and process innovation, and standardization. Furthermore, technological progress is collapsed into a single performance metric, ignoring all other potential metrics of improvement. In spite of these simplifications, performance curves have been shown empirically to be plausible models for describing technologies from industries as diverse as chemicals, agriculture, energy, and information technology.

Despite the broad use of performance curves, no systematic study comparing competing hypotheses across an ensemble of technologies has been published to our knowledge. In this work, we do exactly that. We use hindcasting methodology to assess model performance. The significance of our results is assessed by analyzing randomly chosen subsets of technologies to determine whether the same conclusions hold. We broaden the analysis to study revenue dynamics and its interdependence with technological progress. We present an analytical framework for investigating revenue and compare predictions to empirical observations. Insight into revenue as a driver for technology evolution is discussed in the context of the results comparing competing hypotheses for performance curves.

2 Models

We analyze a suite of different hypotheses for technological progress, shown in Eq.'s 1-6. The first three – Moore's law [Moo65], Goddard's law [God82], and Wright's law [Wri36]

– are hypotheses proposed in the literature. They are all regression models with fitted intercepts, denoted here by b. The remaining models – Moore's law random walk, Goddard's law random walk, and Moore-Goddard's law random walk – are time series models that have not been proposed in the literature, to our knowledge. We refer to them as *random walk* models because unit technology price typically contains drift and noise; however, we do not write the noise term explicitly in Eq.'s 4-6. For brevity, the models will be referred to by the abbrieviation following their names from here onward (e.g. ML, GL, etc.). The variables p_t, x_t, and q_t are the unit price, annual production, and cumulative production in year t, respectively. The parameters m, g, w, $\bar{m}$, $\bar{g}$, $\bar{f}_1$, $\bar{f}_2$, and b are fitted using ordinary least squares using n consecutive years of data as the sample set. A bar above a parameter indicates it is for a time series, as opposed to regression, model.

Moore's law (ML)

$$\log p_t = b - mt \tag{1}$$

Goddard's law (GL)

$$\log p_t = b - g \log x_t \tag{2}$$

Wright's law (WL)

$$\log p_t = b - w \log q_t \tag{3}$$

Moore's law random walk (MRW)

$$\log p_{t+1} = \log p_t - \bar{m} \tag{4}$$

Goddard's law random walk (GRW)

$$\log p_{t+1} = \log p_t - \bar{g} \log \left(\frac{x_{t+1}}{x_t} \right) \tag{5}$$

Moore-Goddard's law random walk (MGRW)

$$\log p_{t+1} = \log p_t - \bar{f}_1 \log \left(\frac{x_{t+1}}{x_t} \right) - \bar{f}_2 \tag{6}$$

3 Methodology

The first part of this work is systematically analyzing the performance of the models across an ensemble of different technologies. We use sixty-two technologies from the Performance Curve Database [PCD], an online database of performance curves, as our test bed. Only data sets with at least ten consecutive years of annual price and production data

were used. Four IT technologies are incorporated into the analysis: hard disk drives, transistors, laser diodes, and DRAM. Acrylic fiber, titanium sponge, geothermal electricity, monochrome television, and beer are a few of the non-IT technologies. A complete list of technologies with references to their original sources and a selection of fit model parameters are in Section 8.1. All data sets are from studies with a scope at least as broad as a national industry (e.g. wind turbine prices in Denmark), while many are global average prices.

To evaluate performance of the models, we use hindcasting methodology. For a given data set, we select a specified number of data points in the series to use as the sample set. The sample set size n ranges from five to fifteen. Results are presented for n=6 unless otherwise noted. We use the sample set to fit the parameters of the model in question. Using the resultant parameter fits, we make a forecast of the unit price for each year through the end of the data series. Since our forecasts are actually in the past, we refer to it as *hindcasting* and compare our predictions to what happened in reality. To quantify forecast accuracy, we use the logarithmic hindcasting error

$$\epsilon = \log p - \log \hat{p}, \tag{7}$$

where $\hat{p}$ is the forecast and p is the historical unit price. The sample set is then shifted one year toward the future, the parameters refit, and a new forecast is made for each year through the last year of available data. The process continues until the sample set comprises the final n data points in the time series. We refer to the last data point in the sample set as the *origin*. The *horizon* is the number of years in the future relative to the origin of the forecast. For a given n, the error ϵ is therefore calculated for each combination of technology, model, origin, and horizon, where each combination thereof is referred to as an *event*.

To assess model performance across the ensemble of technologies, we first normalize the errors of each technology by the standard deviation (k) of the residuals fit to a Gaussian distribution with zero mean. The standard deviation is calculated using the entire data series of each technology. We take the residuals from the fit to MRW and note that all results discussed here hold irrespective of which model is used for normalization. Then, we take an event average of the absolute values of the errors as a function of horizon. Statistical significance is addressed in Section 4.

Implicit in this approach is the assumption that the underlying process of evolution is equivalent for every technology. Said in another way, the models that perform best do so irrespective of the specific technology or industry. Results analyzing subsets of data divided by industry – as labeled in the appendix – support that this is a valid assumption; however, we note that this hypothesis is the subject of ongoing re-evaluation as more data becomes available for analysis.

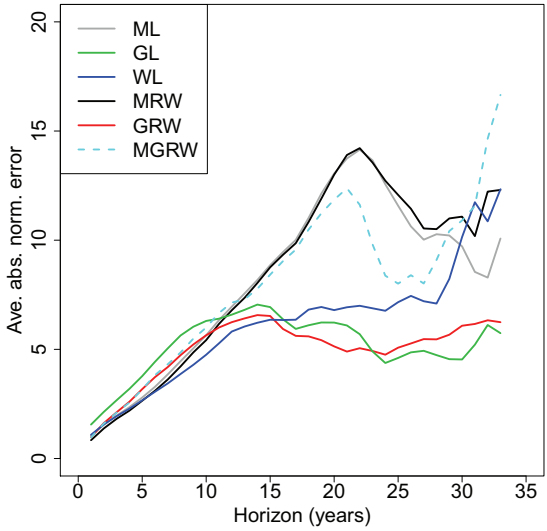


Figure 1: Absolute value of the normalized hindcast error averaged over the ensemble of technologies vs. horizon, for each model: Moore's law (ML), Goddard's law (GL), Wright's law (WL), Moore's law random walk (MRW), Goddard's law random walk (GRW), and Moore-Goddard's law random walk (MGRW).

4 Comparing competing hypotheses

Fig. 1 shows the normalized absolute value of the hindcast error averaged across all technologies for each model as a function of horizon. First, we note that the additional level of complexity brought about by using a multivariate model (MGRW) does not lead to more accurate forecasts. The same conclusion holds for other multivariate models we investigated, including those not presented in this work. All subsequent results will exclude multivariate models unless otherwise noted.

Let us now focus on the univariate models, shown as solid curves in Fig. 1. For horizons greater than approximately ten to fifteen years, the models bifurcate. Two models perform noticeably poorly: ML and MRW. These are the two models that use time as an explanatory variable. Forecasts based on some form of production (annual, cumulative, etc.) make better forecasts than those based on time. For horizons shorter than ten years, the models perform roughly equivalently, with the exception of GL, which performs notably poorer. This result is consistent with recent work by Bela *et al.* [NFBT11].

We observe a difference in relative forecasting accuracy of the performance curve models formulated in terms of production versus time; however, is the difference statistically significant? We note that multiple factors influence the increasingly erratic behavior ob-

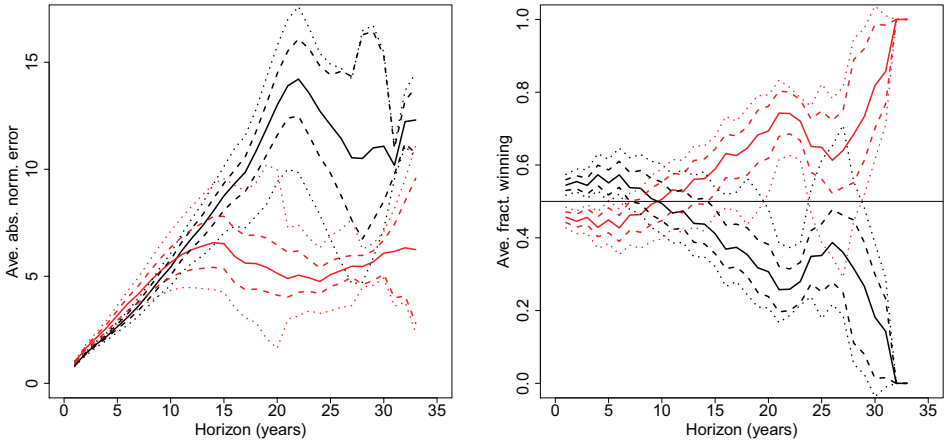


Figure 2: Left: Average of the absolute value of the normalized error for MRW (black) and GRW (red) as a function of horizon. Right: Average of the events won by MRW (black) and GRW (red) as a function of horizon. For both plots, the solid curve is an average over all sixty-two data sets; the dashed (dotted) curve is the average +/- the standard deviation across one hundred randomly chosen subsets of size forty (twenty).

served at increasing horizons. First, as one might intuitively expect, the forecasting error increases as a function of horizon, as does the spread of the distribution of forecasting errors for an ensemble of datasets. Second, as the horizon increases, a decreasing number of technologies are contributing to the average (average data set length is 18 years). These factors have the effect of rendering any observed differences between model performance less significant with increasing horizon. Said in another way, an error bar placed around each curve in Fig.1 would increase in magnitude with horizon. This motivates calculations assessing the statistical significance of our results.

To approach addressing this question, we perform the following robustness analysis. Of the sixty-two technology ensemble, we randomly select m data sets to form a subset. From the subset, we calculate two quantities for MRW and GRW: 1) the average of the absolute value of the normalized error as a function of horizon and 2) the fraction of events for which MRW has the lowest error compared to GRW and vice versa as a function of horizon (which add to unity at every horizon). We chose MRW and GRW specifically as representative of models formulated in terms of time and production. After analyzing one subset, we randomly select another subset and then repeat the analysis. The process is repeated for 100 randomly selected subsets. We take the average and the standard deviation of the two quantities as a function of horizon for each subset size m.

Results are plotted in Fig. 2 for $m = 40$ and 20. The left graph shows the average of the absolute value of the normalized error for MRW (black) and GRW (red) as a function of horizon. On the right, we plot the average of the events won by MRW (black) or GRW

(red) as a function of horizon. For both plots, the solid curve is an average over all sixty-two data sets; the dashed (dotted) curve is the average +/- the standard deviation across the randomly chosen subsets of size forty (twenty). When the subset size decreases from forty to twenty, there is greater variability in the resultant average curves, which is to be expected. However, even with a subset size of twenty, the confidence intervals overlap mildly at horizons greater than fifteen years.

We close this section by noting that assessing the statistical significance of our results is a subject of ongoing investigation. Both more and longer data sets would permit a more conclusive statement about the relative advantage of production over time in forecasting technological innovation. We continue to work toward expanding the Performance Curve Database with other technologies to continue to test this hypothesis. Additionally, we hope the Performance Curve Database will facilitate and promote research by other parties in the general area of technology evolution.

5 Relationship to revenue

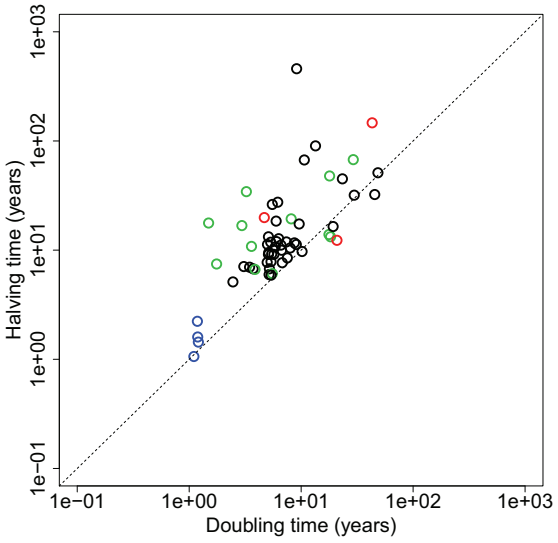


Figure 3: Halving time of unit price vs. doubling time of annual production. The color of the symbol reflects the industry of the technology: chemical (black), energy (green), IT (blue), and other (red). Please note that three data sets were excluded because of negative annual production growth (electric range, free standing gas range, and onshore gas pipeline).

In the previous section, our results indicate that production is a better indicator of price dynamics. To gain further insight into drivers of technological process, we propose one

additional model, which we call the "Revenue random walk".

Revenue random walk (RRW)

$$\log p_{t+1} = \log p_t - \bar{r} \log \left(\frac{x_{t+1} p_{t+1}}{x_t p_t} \right) \tag{8}$$

The product of the annual production (x_t) and unit price (p_t) is the annual revenue, which is the explanatory variable for this model. Eq. 8 is similar in form to Goddard's law random walk, with revenue in place of simply annual production. In fact, after rearrangement, one can show that Eq. 8 is equivalent to Goddard's law random walk, where $\bar{g} = \bar{r}/(1-\bar{r})$[1]. Therefore, one of the best performing models is effectively tracking revenue dynamics, given the direct relationship between revenue and production.

We can further probe the relationship between revenue and price dynamics by formulating the problem in the following manner. First, let us express the revenue as

$$r_t = x_t p_t. \tag{9}$$

We now drop the subscript $_t$ for brevity. The change in revenue is then

$$\frac{dr}{dt} = \frac{dp}{dt} x + p \frac{dx}{dt}. \tag{10}$$

Please note that this derivation is formulated in terms of continuous time dynamics. In order for the industry's revenue to grow or stay constant, we have the condition that

$$\frac{dr}{dt} = \frac{dp}{dt} x + p \frac{dx}{dt} \geq 0. \tag{11}$$

One notable set of solutions to Eq. 11 is

$$\begin{aligned} p &= k_p e^{-t/\tau_p} \\ x &= k_x e^{t/\tau_x}. \end{aligned} \tag{12}$$

Exponential decay of the unit price is simply ML (Eq. 1, which we know to be a plausible model, albeit not the most accurate). Furthermore, empirically we observe that production does grow roughly exponentially across all technologies investigated here [NFBT11]. Our solution set is therefore consistent with empirical observations.

Eq. 12 leads to the condition

$$\frac{\tau_p}{\tau_x} \geq 1. \tag{13}$$

When the firm's revenue is constant, and the price and production are exponential solutions, the timescale of exponential decay of the price must be greater than the timescale of exponential growth of the production for the revenue to increase.

[1] Similarly, a regression model in terms of annual revenue can easily be shown to be equivalent to Goddard's law.

Let us define the halving time as the amount of time (in years) it takes for the unit price to half. We define the doubling time as the amount of time it takes annual production to double. We calculate the halving and doubling times for every technology and construct the scatter plot in Fig. 3. In order for the revenue to remain constant or increase, the rate of production scale-up must be equal to or greater than the rate of price reduction. In other words, we expect the doubling time to be less than or equal to the halving time. Indeed, as seen in Fig. 3, the vast majority of the technologies lie above the identity line. This means that Eq. 13 is met; the overall industry revenue for nearly all technologies is increasing.

This derivation is somewhat unsatisfying because of the imposed functional form for the production. Let us consider another solution set to Eq. 11, and, in doing so, derive an upper limit for the exponent for GRW. We consider $p = k_p x^{-g}$. Using the condition of increasing or flat revenue, Eq. 11, we arrive at the condition $g \leq 1$. Therefore, the scaling exponent g must be less than or equal to unity, the bound for maintaining constant revenue. Empirically, there are no technologies in our analysis where g is greater than unity (outside the error of the fit). The only ones that approaches this value are the Hard Disk Drive, Pentaerythritol, and Phthalic Anhydride data sets, for which $g = 1.0$ (see Section 8.1 for values of g for other technologies). This section provides empirical support for the importance of revenue as a key driver for technological evolution.

6 Discussion

In this work, we comprehensively evaluated competing hypotheses for technology improvement. Using a database of sixty-two different technologies as a test bed, we applied hindcasting methodology to assess the relative performance of the models across the ensemble of data sets. Our results indicate that at long time horizons, production is a better indicator of price dynamics compared to time. This conclusion was robust from analyzing samples of randomly chosen subsets of twenty and forty technologies. However, we note this result is the subject of ongoing investigation.

We then considered revenue as a driver for technological progress. We show that for nearly all technologies, the halving time of the price is less than the doubling time of the annual production, the condition required for increasing industry revenue. We formulate our observations in terms of a simple analytical framework and derive an upper bound for the rate of technological progress in terms of annual production given the condition of increasing revenue. The derived bound is consistent with empirical results from our test bed, where categorically every technology is within this limit. Our results support that revenue is a key driver for technological evolution.

The use of production for forecasting technological progress via a learning or experience curve is often justified in the literature by Arrow's explanation [Arr62]: production is a proxy for accumulated experience, and *learning-by-doing* provides the opportunities for innovation and cost reductions (for further discussion, see [DT84, LE90, MS01, Nem06]). However, our results investigating revenue dynamics suggest that the success of the production models is likely due in part to the direct relationship between production and

revenue. Revenue may better account for industry-wide decision-making that affects technology price dynamics. This is a new angle to the typically posited explanation for using production. Furthermore, our results emphasize the importance of analyzing technology evolution in the context of a broader economic framework.

7 Acknowledgements

AGB acknowledges an appointment to the U.S. Department of Energy under the American Association for the Advancement of Science Fellowship Program administered by Oak Ridge Institute for Science and Education. JDF and QMB were funded under the National Science Foundation grant NSF-0738187. All the conclusions and opinions are those of the authors and do not necessarily reflect those of the NSF. We thank Jessika Trancik, Bela Nagy, and members of the U.S. Department of Energy's SunShot Initiative for useful discussions.

8 Appendix

8.1 Data sets

All data sets can be found on the Performance Curve Database [PCD]. In the tables below, we list the sixty-two data sets used in the above analysis and their original sources. The technologies are divided into separate tables by industry, labelled in the table caption. We also include a selection of parameter fits, including Moore's law (m), Goddard's law (g), Wright's law (w), the halving time of unit price (τ_h), and the doubling time of annual production (τ_d).

Table 1: Industry: energy.

Technology	m	g	w	τ_h	τ_d
CCGT Electricity [CC02]	0.020	0.10	0.12	34	3.2
Crude Oil [Gro72]	0.010	0.38	0.17	68	29
Electric Power [Gro72]	0.036	0.42	0.34	19	8.1
Ethanol [GCNL04]	0.052	0.89	0.36	13	18
Geothermal Electricity [SE09]	0.050	0.81	0.50	14	18
Motor Gasoline [Gro72]	0.014	0.32	0.21	48	18
Offshore Gas Pipeline [Zha99]	0.11	0.21	0.49	6.1	5.5
Onshore Gas Pipeline [Zha99]	0.015	0.13	0.11	45	-
Photovoltaics [May05]	0.064	0.34	0.30	11	3.6
Photovoltaics 2 [Nem06]	0.10	0.56	0.49	6.7	3.9
Wind Electricity [SE09]	0.093	0.17	0.18	7.5	1.8
Wind Turbine [NAD$^+$03]	0.041	0.14	0.13	17	3.0
Wind Turbine 2 [NAD$^+$03]	0.039	0.085	0.072	18	1.5

Table 2: Industry: other.

Technology	m	g	w	τ_h	τ_d
Beer [Gro72]	0.035	0.23	0.20	20	4.7
Electric Range [Gro72]	0.023	-0.023	0.29	31	-
Free Standing Gas Range [Gro72]	0.020	-0.48	0.56	35	-
Monochrome TV [Gro72]	0.056	0.44	0.28	12	21
Refined Cane Sugar [Gro72]	0.0047	0.14	0.32	150	43

Table 3: Industry: chemical.

Technology	m	g	w	τ_h	τ_d
Acrylic Fiber [Lie84]	0.10	0.70	0.58	6.8	5.2
Acrylonitrile [Lie84]	0.076	0.49	0.43	9.1	5.1
Aluminum [Lie84]	0.010	0.14	0.13	67	11
Ammonia [Lie84]	0.090	0.81	0.83	7.7	6.8
Aniline [Lie84]	0.058	0.48	0.93	12	6.0
Benzene [Gro72]	0.062	0.56	0.74	11	6.6
Bisphenol A [Lie84]	0.061	0.43	0.41	11	5.0
Caprolactum [Lie84]	0.12	0.85	0.54	6.0	5.2
Carbon Disulfide [Lie84]	0.021	0.25	0.47	32	45
Cyclohexane [Lie84]	0.052	0.33	0.37	13	5.1
Ethanolamine [Lie84]	0.062	0.77	0.53	11	9.0
Ethyl Alcohol [Lie84]	0.014	-0.083	0.17	51	49
Ethylene [Gro72]	0.037	0.31	0.18	18	6.0
Ethylene 2 [Lie84]	0.065	0.55	0.49	11	5.9
Ethylene Glycol [Lie84]	0.066	0.72	0.70	10	8.0
Formaldehyde [Lie84]	0.060	0.71	0.63	12	8.7
Hydrofluoric Acid [Lie84]	0.0015	0.035	0.018	460	9.1
LD Polyethylene [Gro72]	0.10	0.50	0.38	6.8	3.7
Magnesium [Lie84]	0.0077	0.12	0.15	90	13
Maleic Anhydride [Lie84]	0.054	0.47	0.43	13	6.3
Methanol [Lie84]	0.058	0.63	0.68	12	7.4
Neoprene Rubber [Lie84]	0.022	0.80	0.28	32	30
Paraxylene [Gro72]	0.10	0.43	0.42	7.0	3.5
Pentaerythritol [Lie84]	0.042	1.0	0.45	17	19
Phenol [Lie84]	0.082	0.87	0.84	8.5	7.5
Phthalic Anhydride [Lie84]	0.071	1.0	0.88	9.7	10
Polyester Fiber [Lie84]	0.13	0.47	0.48	5.1	2.5
Polyethylene HD [Lie84]	0.10	0.40	0.46	7.1	3.1
Polyethylene LD [Lie84]	0.089	0.68	0.50	7.8	5.4
Polystyrene [Gro72]	0.058	0.34	0.24	12	5.3
Polyvinylchloride [Gro72]	0.075	0.57	0.43	9.2	5.5
Primary Aluminum [Gro72]	0.025	0.23	0.25	28	6.2
Primary Magnesium [Gro72]	0.026	0.18	0.17	26	5.5
Sodium [Lie84]	0.015	0.38	0.47	45	23
Sodium Chlorate [Lie84]	0.040	0.51	0.40	17	9.6
Styrene [Lie84]	0.069	0.66	0.59	10	6.7
Titanium Sponge [Gro72]	0.12	0.44	0.37	5.9	5.4
Urea [Lie84]	0.073	0.54	0.49	9.5	5.1
Vinyl Acetate [Lie84]	0.076	0.61	0.60	9.1	5.7
Vinyl Chloride [Lie84]	0.090	0.63	0.64	7.7	5.0

Table 4: Industry: information technology.

Technology	m	g	w	τ_h	τ_d
DRAM [Cul08]	0.43	0.74	0.72	1.6	1.2
Hard Disk Drive [Cou08]	0.65	1.0	1.0	1.1	1.1
Laser Diode [LS99]	0.31	0.45	0.39	2.2	1.2
Transistor [Moo06]	0.48	0.84	0.82	1.4	1.2

References

[Arr62] K. Arrow. The economic implications of learning by doing. *Review of Economic Studies*, 29(3):155–173, 1962.

[CC02] U. Colpier and D. Cornland. The Economics of Combined Cycle Gas Turbines - An Experience Curve Analysis. 2002.

[Cou08] T. Coughlin. Personal communication., 2008.

[Cul08] S. Cullen. Personal communication., 2008.

[DT84] J.M. Dutton and A. Thomas. Treating progress functions as a managerial opportunity. *The Academy of Management Review*, 9(2):235–247, 1984.

[GCNL04] J. Goldemberg, S.T. Coelho, P.M. Nastari, and O. Lucon. Ethanol learning curve – the Brazilian experience. *Biomass and Bioenergy*, 26:301–304, 2004.

[God82] C. Goddard. Debunking the learning curve. *IEEE Transactions on Components, Hybrids, and Manufacturing Technology*, 5(4):328–335, 1982.

[Gro72] Boston Consulting Group. Perspectives on Experience, 1972.

[KM06] H. Koh and C.L. Magee. A functional approach for studying technological progress: Application to information technology. *Technological Forecasting and Social Change*, 73(9):1061–1083, 2006.

[KM08] H. Koh and C.L. Magee. A functional approach for studying technological progress: Extension to energy technology. *Technological Forecasting and Social Change*, 75(6):735–758, 2008.

[LE90] L.Argote and D. Epple. Learning curves in manufacturing. *Science*, 247(4945):920–924, 1990.

[Lie84] M.B. Lieberman. The learning curve and prices in the chemical processing industries. *Rand Journal of Economics*, 15(2):213, 1984.

[LS99] T.E. Lipman and D. Sperling. Experience curves for policy making: the case of energy technologies, 1999.

[May05] P.D. Maycock. PV review: World Solar PV market continues explosive growth, 2005.

[Moo65] G.E. Moore. Cramming more components onto integrated circuits. *Electronics Magazine*, 38, 1965.

[Moo06] G.E. Moore. https://www.youtube.com/watch?v=fxLnbVjKXmQ, 2006.

[MS01] A. McDonald and L. Schrattenholzer. Learning rates for energy technologies. *Energy Policy*, 29(4):255–261, 2001.

[NAD+03] L. Neij, P.D. Andersen, M. Durstewitzand, P. Helby, and M. Hoppe-Kilpper P.E. Morthorst. Experience curves: a tool for energy policy programmes assessment, 2003.

[Nem06] G.F. Nemet. Beyond the learning curve: factors influencing cost reductions in photovoltaics. *Energy Policy*, 34:3218–3232, 2006.

[NFBT11] B. Nagy, J.D. Farmer, Q.M. Bui, and J.E. Trancik. Predicting technological progress. 2011.

[Nor09] W.D. Nordhaus. The perils of the learning model for modeling endogenous technological change. *SSRN eLibrary*, 2009.

[PCD] Performance Curve Database. http://pcdb.santafe.edu.

[SE09] M.A. Schilling and M. Esmundo. Technology S-curves in renewable energy alternatives: Analysis and implications for industry and government. *Energy Policy*, 37(5):1767–1781, 2009.

[SSC00] G. Sinclair, S.Klepper, and W. Cohen. What's Experience Got to Do With It? Sources of Cost Reduction in a Large Specialty Chemicals Producer. *Management Science*, 46(1):28–45, 2000.

[Wri36] T.P. Wright. Factors affecting the cost of airplanes. *Journal of Aeronautical Science*, 3(4):122–128, 1936.

[Zha99] J. Zhao. The diffusion and cost of natural gas infrastructures, 1999.

Hot Topic Detection in Local Areas Using Twitter and Wikipedia

Shota Ishikawa, Yutaka Arakawa, Shigeaki Tagashira, Akira Fukuda

Graduate School / Faculty of Information Science and Electrical Engineering
Kyushu University
744 Motooka, Nishi-ku, Fukuoka, Japan
{ishikawa, arakawa, shigeaki, fukuda}@f.ait.kyushu-u.ac.jp

Abstract: As microblog services become increasingly popular, spatial-temporal text data has increased explosively. Many studies have proposed methods to spatially and temporally analyze an event, indicated by the text data. These studies have aimed a extracting the period and the location in which a specified topic frequently occurs. In this paper, we focus on a system that detects hot topic in a local area and during a particular period. There can be a variation in the words used even though the posts are essentially about the same hot topic. We propose a classification method that mitigates the variation of posted words related to the same topic.

1 Introduction

Over the past several years, participation in social media, e.g., posting and/or reading, has gradually become a routine part of many peoples' lives. The posts cover a wide range of topics, including daily activities, events, opinions, comments, photographs, and links to web pages. The popularity of this form of communication has been driven by advances in mobile phone technology. Smartphone, which enable access to internet services, are becoming increasingly popular at an unprecedented rate. Social media applications for smartphones have also been developed and popularized. These client applications have features that exploit smartphone's ancillary functions such as a global posting system (GPS) and a camera, which, for example, enable users to post still or video images, and determines their current location. These associated features greatly improve the usefulness and will further spur the growth of social media services. The number of people who use a variety of microblog service is astonishing. In particular, $Twitter$ has attracts an estimated 200 million participants globally since 2006 when the service started (Twitter, Inc. estimated that the number of active users is 100 million) [1]. In Japan, the growth rate of Twitter users is remarkable. In [9], a good correlation was reported between smartphone owners and Twitter users. Since smartphones are spreading increasingly rapidly, we can predict that the number of users participating in social media will continue to rise.

Due to the convergence of technology and high participation rates, it is now possible to

[1] http://blog.twitter.com/2011/09/one-hundred-million-voices.html

readily collect more spatial-temporal text-based Twitter posts, known as "tweets," than before, since a tweet can be associated with not only the posting time but also the posting position. By analyzing the contents of tweets, it is possible to forecast a market [4], sense a circumstance (weather and noise level) in a specific area [7], and identify hot topics that coincide at a particular time and location; it would lead several studies on complex networks.

Examples of hot topics are predictable events such as a soccer game or a festival, and unpredictable events such as a natural disaster or traffic accident. These events would probably be common hot topics for users in corresponding location. Additionally, in [3], it was reported that word (hot topic) is often used at a specific location in the analysis of the relationship between the input words typed by users and the users' positions. It is expected that this word would also be of interest to other users in the same area. Therefore, it would help a user to detect the hot topics associated with their location. This detection can also contribute to artificial intelligence services such as suggesting keywords for web search system, which would save time while inputing the words into a search engine. Furthermore, it is possible to evaluate the efficacy of a specific advertising campaign and comprehend a disseminating flow of the detected hot topics by observing the spatial-temporal changes in hot topics. It would be helpful to produce a marketing strategy. Similarly, this analysis would be applied to the spreading model of an infection disease and the network of people's relationship, which has been focused in complex network science.

We propose a novel detection scheme for hot topics on Twitter. The basic approach is to classify tweets into topics according to their content and select the top topics, ranked according to the number of topics' tweets, as hot topics. In this classification, the tweets, including words referring to the same event must be exactly associated with the corresponding topic. However, due to semantic fluctuations, this classification does not work particularly well. For example, tweets can use different words to refer to the same event, and consequently, they will be classified as different topics. In this paper, we identify semantic fluctuation as spelling, spatial, and temporal fluctuations. For example, the word "stadium" included in a tweet could refer to a baseball game in a particular region, but it could also refer to a soccer match in a different location This is an example of a spatial fluctuation. Similarly, the word "festival" has seasonal or temporal fluctuation. It could imply a music festival held in May or a food festival held in July, although both events take place in the same region.

Here, we focus on spatial and temporal fluctuations and propose a clustering method to cope with these fluctuations. Basically, we interrelate words based on their interpreted meanings graphically using Twitter and Wikipedia. By analyzing the tweets, we can understand the local meaning of time- and position- dependent words. In Wikipedia, the structure of global meanings for common words in every region has been built. Next, we associate these graphs with topics, on the basis of their similar occurrences on the graph. Finally, we detect hot topics by calculating the frequency of each topic in a specific period.

The paper is organized as follows; In section 2, we introduce the related work analyzing spatial-temporal text data and make a sharp distinction between these studies and our approach; In section 3, we describe our approach in detail; we then present the result and discussions of a preliminary experiment in section 4; in section 5, we propose a technique

to speed-up the process; finally, we present our conclusions and suggest possible future work in section 6.

2 Related Work

Several recent research have analyzed the spatial-temporal text data of social media. The main goal of our research is to detect hot topics (including semantic summarizing); other researches have pursued similar goals. For example, Fujisaka et al. [8] collected tweets using Twitter's application programming interface (API), and analyzed the movement histories of several users. They discovered characteristic mobility patterns in an urban area.

Yamanaka et al. [20] has proposed an extraction method that detects events in a given observation area. Initially this method categorizes messages with attached GPS information by using a support vector machine (SVM) model [10]. Secondly, the messages are clustered based on messages' category and the position. Finally, a burst is detected for each cluster. The burst-detection method, which has been proposed by Kleinberg [11], detects whether the interval between messages is more dense than that in a normal condition. However, this method needs to predefine a query set for each area and condition.

In addition, Sakaki et al. [18] has proposed an event-detection method using Twitter. For example, the method focuses on an earthquake as the event. It uses the SVM as an event classifier and then detects the event by calculating the occurrence probability. However, this method only handles specific words relating to the particular event. It lacks general versatility for event detection. Similar to the above method, this method also requires the configuration of a query set for each situation and event in advance. Thus, to apply this method to a wide range of situations, a considerable amount of time would be required to prepare the queries.

In addition to the above studies, a few event detection approaches that do not need predefined query sets have been proposed. Mathioudakis et al. [13] has developed a technique for detecting a trend based on the co-occurrence probability between events. However, in this method, the resultant trend sometimes includes phrases commonly used in Twitter conversations. Becker et al. [5] [6] has detected an event by analyzing tweets and identified whether the event had actually happened. Here, tweets are classified by calculating characteristics from temporal, social, topical, and Twitter-centric features, and then separating events from non-events. However, the process involves using capitalizations rules to split multiple word hashtags into single words. This could not be applied to Japanese as the language does not use capital letters.

Another set of related studies examined semantic summarizations of topics. This approach often exploits hashtags. A hashtag, which is a Twitter specific annotation format, is used to designate or assign a topic in a tweet. For example, in Canada a tweet about professional hockey matches are often tagged with the hashtag, #NHL(National Hockey League). Using hashtags, enable us to summarize tweet topics effectively. Rosa et al. [17] categorized tweets into six predefined topics using hashtags. Long et al. [12] has attempted to summarize tweets using hashtags in $SinaMicroblog$. Moreover, Motooka et al. [14] has re-

searched tweets about a similar event using hashtags. This approach collects a set of users using a specified hashtag. It displays top events ranked according to the similarity between the specified hashtag and all events associated with the tweets. Here, it is important to note that a tweet can have multiple hashtags. However, the number of tweets marked by hashtags is still small; the percentage of hash-tagged tweets in geotagged tweets is only 0.4% [2]. Therefore, in order to ensure a broader event detection the proposed method did not employ a hashtag-based detection.

It is possible to use information resources other than Twitter, used in above approaches, for detecting events or topics. WordNet [1] has published a meaning dictionary that groups words into sets of concepts and links conceptually similar sets to indicate relationship. However, WordNet's primary classification is based on parts of speech, i.e., nouns , verbs, adjectives, and adverbs. Therefore, for example, it is impossible to find a relationship between the noun "earthquake" and the verb "to shake." Consequently, several ontology construction methods using Wikipedia and Folksonomy have been proposed in [16] [19]. Wikipedia has already covered a wide range of vocabulary related to global areas or regions. Moreover, Wikipedia has been built semi-structured data (redirect link, category tree, and infobox), which makes it possible to construct the relationship between concepts. The goal of the ontology construction approach is to clearly define the relationships among concepts (is-a relationship, and a-part-of relationship). Our purpose is also to build the relationships and a path from upper concepts to lower ones. In our approach, we also use Wikipedia to arrange words in terms of semantics.

3 Proposed Detection Method

In this section, we present a detailed description of our proposed hot-topics detection method.

We collect tweets that are associated with geotag using Twitter's streaming API [3]. Firstly, we eliminate tweets posted from $foursquare$ [4]. Because most of these tweets only contain the location information and a URL. Thus, they are not useful samples for our hot topic detection method. Similarly, we also eliminate URLs from the text in all the tweets [17].

Next, we perform a morphological analysis using MeCab (Japanese morphological analysis engine) [5] to decompose the text into parts of speech. Since it is difficult to accurately associate the qualified word with an adjective or adverb, we only focus on nouns and verbs extracted by the morphological analysis. In this paper, we do not focus on the morphological analysis's methodology; however in future, we intend to improve the accuracy of the extraction process.

After building a set of semantically related words contained in the obtained tweets, we

[2] We surveyed tweets posted in Tokyo area from Jun to Sep. in 2011. As a result, among 277,249 geotagged tweets, there are only 1,088 hashtagged tweets.

[3] http://dev.twitter.com/pages/streaming_api

[4] Foursquare : www.foursquare.com

[5] MeCab : http://mecab.sourceforge.net/

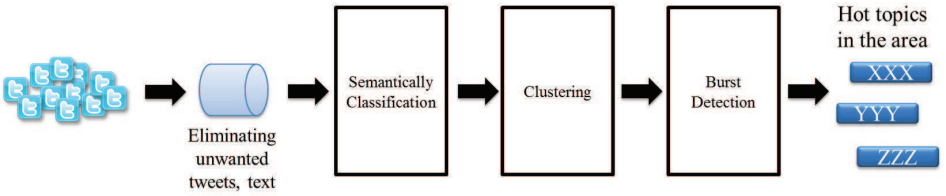


Figure 1: Flow of proposed method.

detect hot topics from the set. The detection method consists of the following procedures.

- Building relationship among a set of words (section 3.1)
- Classify the words into topics (section 3.2)
- Detect hot topics using a burst detection method (section 3.3)

3.1 Building relationship among a set of words

It is possible that different words indicate the same topic, and the converse being true. This possibility is termed as "semantic fluctuation." As described previously, we consider that the semantic fluctuation in Twitter consists of spelling, spatial, and temporal fluctuations. For example of spatial fluctuation, although a word "stadium" included in a tweet is concerned with the topic of baseball in a region, the word might express about that of soccer elsewhere. As for temporal fluctuation, a word "festival" implies different topics in each season, e.g., the word indicates not only a music festival held on May but also a food festival held on July even in the same region.

While building relationship among a set of words, we have assumed that each tweet refers to a particular topic. This is a reasonable assumption considering the limit on the number of input characters (a tweet must not exceed 140 characters).

To determine relationships between words, firstly we build relationships between pairs of words contained in each tweet, i.e., the link of weight (w_i) is established between any combination of words. Secondly, if any two tweets contain the same pair of words, the link of weight (w_r) is established between the pairs, where w_r is proportional to the number of word combinations. Additionally, if a word exists in a hash-tagged tweet, the weight of the link related with the word is w_h ($w_i < w_h$). If a word is included in Wikipedia, the rank of the word is added by one. Finally, we obtain links from upper concepts defined in Wikipedia, by the matching category name or Infobox template, and scraping, as proposed in [19].

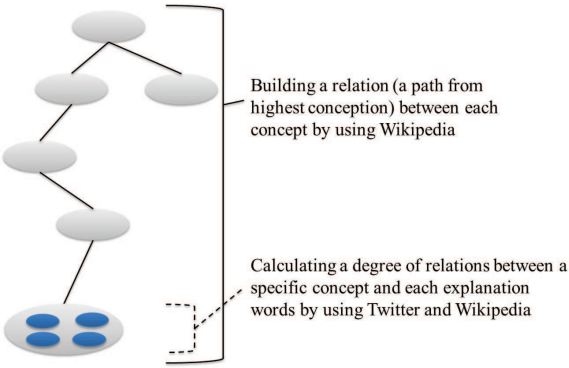


Figure 2: Building conceptual path and relationship between concepts and words.

3.2 Clustering

We utilize a clustering method to classify the words into topics. This clustering method introduces a similarity degree of association between words and topics. We adopt an incremental clustering method such as that proposed in [2], rather than recreating the sequential clustering. The proposed method calculates a degree of similarity $sim(m_i, c_j)$ between words $M = \{m_1, .., m_i, .., m_n\}$ and existing clusters $C = \{c_1, .., c_j, .., c_k\}$. If $sim(m_i, c_j)$ exceeds τ, the word is classified into the maximized cluster c_j. On the other hand, when $sim(m_i, c_j)$ is less than τ, we classify the word into a new cluster c_{k+1}. In addition, a threshold τ is determined empirically.

3.3 Detecting Burst Topics

Next, we use a burst-detection method to determine the frequency of a topic in a given period. The method has been proposed in [11]. This method detects whether the interval of the arriving messages is denser than that in a normal condition through comparison with other document streams such as bulletin board threads and current news articles.

The burst-detection method defines a probabilistic automaton (A) consisting of two states: (1) When A is in state q_0, messages arrive at a slower rate. (2) When A is in state q_1, messages arrive at a faster rate. The period (T) is defined as the interval between the arrival of the first message and that of the last message, $n + 1$. If the arrival time is random, a gap time x between messages i and $i + 1$ follows an exponential distribution. According to Poisson distribution, in state q_0, the probability of arrival of the next message at interval x is $f_0(x) = \alpha_0 e^{-\alpha_0 x}$, where $\alpha_0 = n/T$. In state q_1, the gap time is shorter than in state q_0. Consequently, the probability of interval x between any two consecutive message is $f_1(x) = \alpha_1 e^{-\alpha_1 x}$, where $\alpha_1 > \alpha_0$.

In addition, we determine a given set of n messages with a specified arrival time as inner-

arrival gaps $\mathbf{x} = (x_1, x_2, ..., x_n)$, where $x_i > 0$. Similarly, we set the conditional probability of a state sequence $\mathbf{q} = (q_{i_1}, q_{i_2}, ..., q_{i_n})$. Each state sequence q derives a density function f over sequences of gaps, which is represented by the following formula.

$$f_q(x_1, ..., x_n) = \prod_{t=1}^{n} f_{i_t}(x_t) \tag{1}$$

Hence, when the inner-arrival gaps is equal to $\mathbf{x}$, a conditional probability that the state sequence is formed by $\mathbf{q}$ exists and, is given by the following formula.

$$Pr[\mathbf{q}|\mathbf{x}] = \prod_{t=1}^{n} f_{q_t}(x_t) \tag{2}$$

We have assumed that a maximum likelihood (burst) state is equal to $\mathbf{q}$ when it takes the highest value among probability $Pr[\mathbf{q}|\mathbf{x}]$. i.e., it is equivalent to minimum of the following values.

$$-\ln Pr[\mathbf{q}|\mathbf{x}] = \sum_{t=1}^{n} -\ln f_{i_t}(x_t) \tag{3}$$

We can detect a cluster burst by finding the state $\mathbf{q}$ that has the lowest value among those described.

4 Experiments

We conducted a preliminary experiment to examine the semantic fluctuation of words included in tweets. In this experiment, we collected tweets that were associated with geotags in a 5 $\times$ 5 km area around a baseball stadium. Fig. 3 shows the possibility of words and Fig.4 shows the log distribution of words rank and frequency of words.

From the result, we can observe that the spatial words “Tokyo” and “Shinjyuku,” which are related with the analyzed area, frequently appear in the whole range of date. However, the temporary words “strike” and “preemptive point,” expressly related to baseball, only appear on a specific date, i.e., a baseball game is held on that date.

In addition, the words “strike” and “preemptive point” did not appear on every days when a baseball game were held. This is because the percentage of tweets that indicate the event is primitively low. The contents of major tweets are conversations and do not indicate specific topics. The distribution of word in these tweets sampled through the API follws a Zipfian distribution, as shown Fig. 4. On the other hand, the number of topical tweets increases sharply only when the specific event occurs, likely to cause bursts in temporal. Namely, these burst words depend on time. Therefore, we can pick up a word which is related with the specific topic and is not general word in the tail of Zipfian distribution by calculating IDF (Inverse Document Frequency) (we consider tweets per unit time as a single document), and collect topical words efficiently.

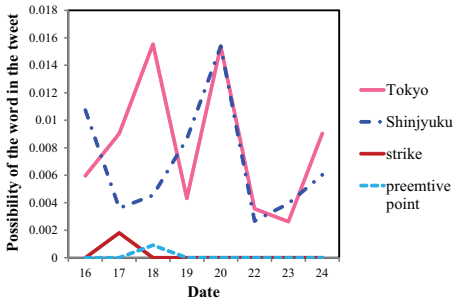


Figure 3: Possibility of the word.

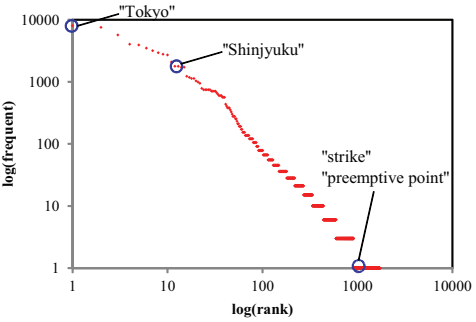


Figure 4: Log distribution of words rank and frequency

5 Technique to speed up the process

In our approach, we analyze tweets in each geographical region, which are positioned in a regularly spaced grid, as shown in Fig. 5. Analyzing tweets in every region requires a considerable amount of computation time. If our target application is not responsive in real time due to the time required to process the calculations, the system would not be helpful for users. Thus, we propose decreasing the computation time by reducing the absolute number of analyzed areas.

Although Fujisaka et al. proposed an area splitting method in [8], their purpose was to avoid the use of the API as much as possible. In our approach, our purpose is to consistently reduce the number of analyzed areas. Besides, the number of tweets varies in each area, as shown in [9]. If the difference in the number of tweets in each area is caused by a simple splitting method, the statistical result would be unreliable. In that case we would detect hot topics in a specific region where the number of tweets is a few, a group of few tweets (compared to the average number of tweets in whole regions) indicates an event that would even affect a result. Therefore, to speed up the process, if the number of tweets in a region does not reach a certain number, we could expand the grid area until the number of tweets reaches a specified threshold. As a consequence, the number of tweets in every grid is normalized and the number of grids is reduced (see Fig. 5).

We define the threshold number of tweets as N. If the number of tweets in area E is less than N, the number of adjacent areas are calculated. Then, if the number of adjacent areas is over α, we discontinue detecting hot topics and use the result for hot topics in these areas because we consider that degree of interest in a topic is a spatial continuous value and the hot topics also appear in adjacent areas. On the other hand, if the number of areas is less than α, we integrate these adjacent areas into one area.

As mentioned above, we decrease the number of areas by expanding areas recursively and referring to adjacent hot topics. Consequently, the computational effort is lowered.

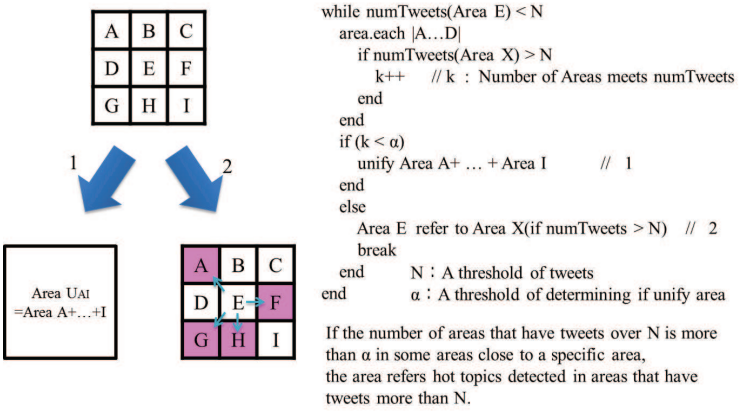


Figure 5: Saving calculation time by expanding the area and using hot topics of adjacent areas.

6 Conclusions

In this paper, we proposed a novel detection scheme for hot-topics on Twitter. The basic approach is to classify tweets into topics according to their content and to select top topics ranked according to the number of the topics' tweets. This detection can contribute to artificial intelligence services, such as suggesting keywords for web search system and thus save time while inputting words into search engines. In addition, it helps to comprehend the trend flow in a marketing analysis by observing the detected hot topics changes in temporal-spatial. In future, we intend to elaborate the detail of the proposed system and implement the same.

References

[1] Japanese WordNet, http://nlpwww.nict.go.jp/wn-ja/index.en.html

[2] J. Allan, R. Papka, and V. Lavrenko, "On-line New Event Detection and Tracking," *Proc. the 21st ACM International Conference on Information Retrieval (SIGIR)*, pp. 37–45, 1998.

[3] Y. Arakawa, S. Tagashira, and A. Fukuda, "Relationship Analysis between User Contexts and Input Word with Twitter," Transactions of Information Processing Society of Japan, Vol. 52, No. 7, pp. 2268–2276, 2011. (in Japanese).

[4] S. Asur, and B. A. Huberman, "Predicting the Future with Social Media," *Proc. the 2010 IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology*, Vol. 1, pp. 492–499, 2010.

[5] H. Becker, M. Naaman, and L. Gravano, "Beyond Trending Topics: Real-World Event Identification on Twitter," *Proc. the 5th International AAAI Conference on Weblogs and Social Media (ICWSM)*, 2011.

[6] H. Becker, M. Naaman, and L. Gravano, “Beyond Trending Topics: Real-World Event Identification on Twitter,” Technical Report cucs-012-11, Columbia University, 2011.

[7] M. Demirbas, C. Akcora, M. Bayir, Y. Yilmaz, and H. Ferhatosmanoglu, “Crowd-Sourced Sensing and Collaboration Using Twitter,” *Proc. the 2010 IEEE International Symposium on A World of Wireless, Mobile and Multimedia Networks (WoWMoM)*, pp. 1–9, 2010.

[8] T. Fujisaka, R. Lee, and K. Sumiya, “Exploring Urban Characteristics Using Movement History of Mass Mobile Microbloggers,” *Proc. the 11th Workshop on Mobile Computing Systems & Applications (HotMobile)*, pp. 13–18, 2010.

[9] Hakuhodo DY Media Partners Institute of Media Environment,“2011 Media Teiten chosa,” http://www.media-kankyo.jp/upload/files/article_128/teiten2011.pdf, 2011. (in Japanese).

[10] T. Joachims, “Text Categorization with Support Vector Machines,” *Proc. the 10th European Conference on Machine Learning (ECML)*, pp. 137–142, 1998.

[11] J. Kleinberg, “Bursty and Hierarchical Structure in Streams,” *Proc. the 8th ACM International Conference on Knowledge Discovery and Data Mining (SIGKDD)*, pp. 91–101, 2002.

[12] R. Long, H. Wang, Y. Chen, O. Jin, and Y. Yu, “Towards Effective Event Detection, Tracking and Summarization on Microblog Data, ” *Proc. the 12th International Conference on Web-age Information Management (WAIM)*, pp. 652–663, 2011.

[13] M. Mathioudakis and N. Koudas, “TwitterMonitor: Trend Detection over the Twitter Stream, ” *Proc. the 2010 ACM International Conference on on Management of Data (SIGMOD)*, pp. 1155–1158. 2010.

[14] R. Motooka, T. Yumoto, M. Nii, Y. Takahashi, and K. Sumiya, “A Similar Event Search System Using Hashtag of Twitter,” The Database Society of Japan, The 3rd Forum on Data Engineering and Information Management (DEIM), A1-5, 2011. (in Japanese).

[15] B. O’Connor, M. Krieger, and D. Ahn, “TweetMotif: Exploratory search and topic summarization for twitter,” *Proc. 4th International AAAI Conference on Weblogs and Social Media (ICWSM)*, 2010.

[16] S. P. Ponzetto and M. Strube , “Deriving a Large Scale Taxonomy from Wikipedia,” *Proc. the 22nd National Conference on Artificial Intelligence (AAAI)*, pp. 1440–1445, 2007.

[17] K. D. Rosa, R. Shah, B. Lin, A. Gershman, and R. Frederking, “Topical Clustering of Tweets, ” The 3rd Workshop on Social Web Search and Mining (SWSM), 2011.

[18] T. Sakaki, M. Okazaki, and Y. Matsuo, “Earthquake Shakes Twitter Users: Real-time Event Detection by Social Sensors,” *Proc. the 19th International Conference on World Wide Web (WWW)*, pp. 851–860, 2010.

[19] S. Tamagawa, S. Sakurai, T. Tejima, T. Morita, N. Izumi, and T. Yamaguchi , “Learning a Large Scale of Ontology from Japanese Wikipedia,” Transactions of the Japanese Society for Artificial Intelligence, Vol. 25, No. 5, pp. 623–636, 2010. (in Japanese).

[20] T. Yamanaka, Y. Tanaka, Y. Hijikata, and S. Nishida, “A Supporting System for Situation Assessment using Text Data with Spatio-temporal Information,” Journal of Japan Society for Fuzzy Theory and Intelligent Informatics, Vol. 22, No. 6. pp. 691–706, 2010. (in Japanese).

A Network Perspective on Software Modularity

Marcelo Serrano Zanetti, Frank Schweitzer

Chair of Systems Design
ETH Zürich
Kreuzplatz 5
CH-8032 Zürich
{mzanetti, fschweitzer}@ethz.ch

Abstract: Modularity is a desirable characteristic for software systems. In this article we propose to use a quantitative method from complex network sciences to estimate the coherence between the modularity of the dependency network of large open source JAVA projects and their decomposition in terms of JAVA packages. The results presented in this article indicate that our methodology offers a promising and reasonable quantitative approach with potential impact on software engineering processes.

1 Introduction

The modularity of a software architecture is considered a key feature that contributes to the sustainability of large scale software projects [PCW85]. Ideally, modularization fosters the decoupling of software development efforts, which can then be performed independently if a binding standard interface is established. As the software evolves in time, modularity might even favor its maintainability and expandability. If the development of a given system is meant to be sustainable, the amount of effort required to perform modifications in the software architecture must be compatible with the resources (time, human, etc) available at any time. Therefore monitoring the modularity of an evolving software system promises to be an important step towards a sustainable software development regime, however such a task would be tedious and slow if performed manually.

In this article we propose an efficient automatic quantitative approach to estimate the coherence between the modularity of the dependency network of large open source JAVA projects and their decomposition in terms of JAVA packages. Our method is based on the well-established complex networks framework [AB02][New03b]. In order to adopt this framework, the first necessary step is to restate software modules and software systems in terms of network structures (see [HR92][Mye03][Koh09][GS11]).

Through a network perspective, it is straightforward to visualize that the expected functionality of a software module is provided by the cooperation of fundamental software entities (functions, classes, procedures, etc) which perform the necessary operations. Thus a software module is a mesoscopic abstraction for a collection of entities acting microscopically. At the mesoscopic scale, software modules themselves become interdependent

when integrated into a software system. Therefore the challenge in modularization of software consists in clustering highly dependent microscopic software entities, which are then packaged into software modules by minimizing the number of dependencies across modules after a system integration. This can be directly mapped to the software engineering literature, where modularity is defined as a high degree of intra-module *cohesion* and low inter-module *coupling* [GJM03]. As an example, since the number of dependencies across modules is expected to be minimized, a modular system is relatively easy to be upgraded through the replacement of an obsolete software module by a new one.

Our contribution is based on a quantitative metric that measures the coherence between the decomposition of a software system into software modules and the cluster structures found in the network model of the software at a microscopic scale. However, here we do not attempt to construct module mappings that optimize this coherence. We only monitor the modularity of a software system already decomposed in terms of software modules. For this, we use a quantitative metric which describes a macroscopic property of a system composed of microscopic and mesoscopic structures (software dependencies and modular decomposition respectively). In other words, our method can measure the global impact of modifications made locally during the time evolution of a given software project. To illustrate the dynamics of this process, we study the time evolution of the degree of modularity expressed through our method for 28 open source JAVA projects. Our dataset contains different versions of the source code which were extracted periodically from the respective online software repositories. We argue that the application of the complex systems framework in the study of software systems provides valuable insights into the software engineering processes and the sustainability of large scale software projects.

In section 2 we present the details of our implementation and approach. Section 3 discusses our preliminary results and in section 4 we comment on related work. Finally, in section 5 we conclude our work and we then elaborate on further research ideas.

2 Methodology

The starting point of our methodology is the re-expression of source code dependencies in terms of network structures. Conceptually, such an approach will differ for the targeted programming language and programming paradigm.

We choose to focus our efforts on software written in JAVA, for it is an object-oriented programming language which suggests a straightforward re-interpretation in terms of networks: JAVA classes are taken as network nodes, while a network edge will connect any two nodes if the corresponding JAVA classes share at least one software dependency (call, access of property, inheritance, etc). Another relevant aspect of JAVA is its built-in support for software modularization through the assignment of classes to packages. Last but not least, JAVA is a very popular programming language among free and open source software developers, and therefore plenty of examples containing the complete source code evolution is available online in software repositories and web software development platforms,

such as GITHUB[1] and SOURCEFORGE[2].

Figure 1 presents a visual example of the software network resulting from the application of the aforementioned method to one of the versions of the source code of ASPECTJ, which is a JAVA framework supporting the implementation of software using the aspect-oriented programming paradigm. In our dataset, this network grows from 654 up to 1651 nodes (classes). In this example, each color represents the package membership (module) of each class found in source code. This network perspective on source code can be extended in a relatively easy way to other programming languages and paradigms. See [HR92][Mye03] for more examples and approaches.

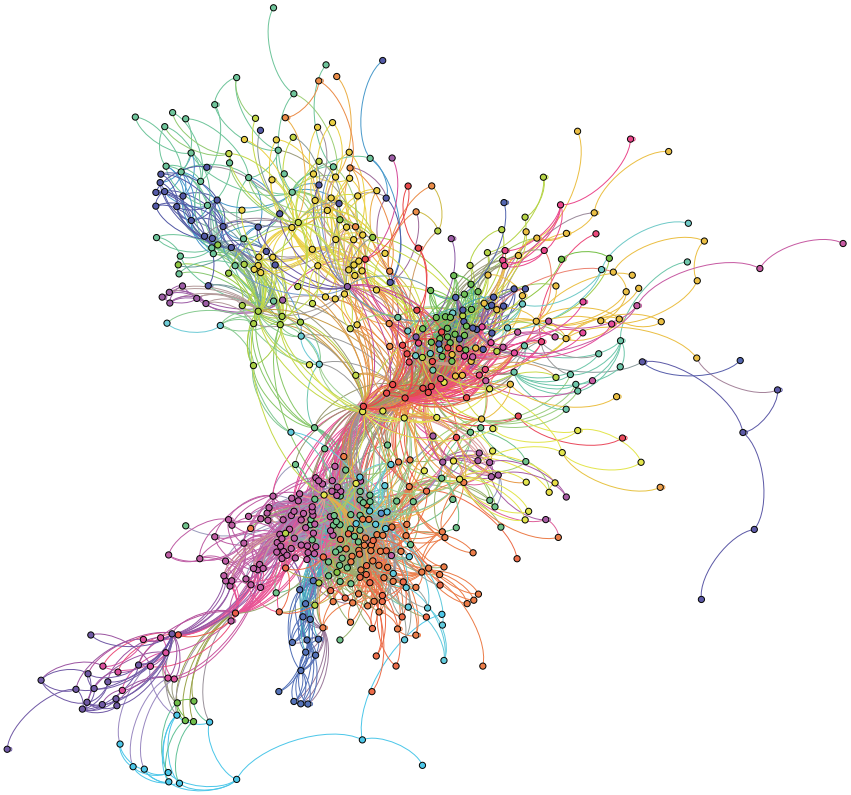


Figure 1: Visualization of the modular network structure of ASPECTJ as of 01-Aug-2004 (only the largest set of nodes connected via direct or indirect edges - largest connected component). This visualization was generated by GEPHI [BHJ09].

[1] https://github.com/
[2] http://sourceforge.net/

As demonstrated in Figure 1, the visualization of network structures is a very useful technique for the analysis of the modularity of a given software architecture. However, a quantitative approach is still desirable since it allows us to capture the structural organization of a network in terms of a single numeric measure. This can be used to analyze the time evolution of a modular software architecture and can also be applied in a statistical correlation analysis when considering different quantitative metrics.

In recent years, the network sciences community has developed a number of quantitative metrics which capture structural features like e.g. clusters as well as the impact of nodes, clusters or any other structural entities on dynamical processes like e.g. information or failure spreading, consensus, opinion formation or synchronization [New10]. According to our needs, we adopt a network metric which was first used to study assortative mixing in networks, which is the tendency for network nodes to be connected to other nodes that are like (or unlike) them in some way [New03a]. Assuming that sharing the same module membership makes nodes alike (and unlike otherwise), this metric could then be used to measure the modularity of network structures [NG04]. For a given definition of *modules* or *clusters* and their underlying network structure, its respective degree of modularity is defined by

$$Q = \frac{\sum_i^n e_{ii} - \sum_i^n a_i b_i}{1 - \sum_i^n a_i b_i} \tag{1}$$

where e_{ij} is the fraction of all edges in the network that link nodes in module i to nodes in module j, $a_i = \sum_j^n e_{ij}$, $b_i = \sum_j^n e_{ji}$ (column and row sum respectively) while n is the total number of existing modules. If the network is an undirected graph the matrix defined by $\mathbf{e}$ is symmetric and $a_i = b_i$ [New03a]. The metric defined by equation (1) measures the fraction of network edges that connect nodes within the same module ($\sum_i^n e_{ii}$) minus the expected value of the same quantity measured from a random network with the same node/module allocation ($\sum_i^n a_i b_i$). If the first is not better than random $Q = 0$ [NG04]. However, Q would not be defined if all edges are concentrated within a single module because the scaling factor $1 - \sum_i^n a_i b_i = 0$ (no modular structure). In such a case we define $Q = 0$ as well. In general, $Q \in [-1, 1]$, i.e. the more modular the network, the closer Q is to 1. Figure (2) provides two examples of networks and their respective Q scores.

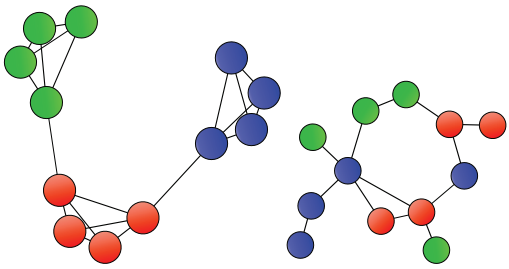


Figure 2: Two examples of undirected networks where nodes (circles) with the same color are part of the same module. (left) modular network Q=0.8499. (right) random connectivity Q=0.0545.

In the analysis of software structures, this metric is useful because in many cases the definition of modules is given by means of programming constructs like classes, files, namespaces or packages. The Q–metric can thus be used to study how well the cluster structures in the network of dependencies correspond to the modular decomposition of a project in terms of packages, namespaces, etc. We applied the Q-metric in an analysis of the evolution of the modularity of the software architecture of a set of JAVA open source projects and we discuss our preliminary results in section 3.

3 Preliminary Results

Our analysis is based on a dataset containing the detailed time evolution for the source code of 28 open source JAVA projects. The snapshots of the source code of each project were extracted from the respective CVS online software repositories, on a monthly basis. Table 1 displays the recorded period for each project. Most of those projects are hosted at SOURCEFORGE and were selected because they were the largest (number of classes) at the time the dataset was collected. The single exception is ECLIPSE, which has its own online facilities[3]. The source code for ECLIPSE was thus obtained through a different setup. For each project the CVS change history and class dependence structure were extracted, processed and stored in a directed graph format, i.e. (c_1, c_2, T) which reads as c_1 *depends on* c_2 *at time* T.

Table 1: The 28 JAVA projects which compose our source code evolution dataset. Most of those projects were extracted from the respective CVS software repositories hosted by SOURCEFORGE.

project name	record start	record end	project name	record start	record end
architecturware	2004-04-01	2007-12-01	jnode	2003-06-01	2005-12-01
aspectj	2003-01-01	2008-02-01	jpox	2003-09-01	2006-12-01
azureus	2003-08-01	2008-01-01	openqrm	2007-04-01	2008-03-01
cjos	2000-11-01	2007-12-01	openuss	2003-06-01	2006-12-01
composestar	2003-12-01	2005-12-01	openxava	2004-12-01	2007-12-01
eclipse	2001-05-01	2008-03-01	personalaccess	2004-11-01	2007-12-01
enterprise	2002-11-01	2007-12-01	phpeclipse	2002-08-01	2007-12-01
findbugs	2003-04-01	2007-12-01	rodin-b-sharp	2005-11-01	2007-12-01
fudaa	2003-02-01	2007-12-01	sapia	2002-12-01	2007-12-01
gpe4gtk	2005-08-01	2006-12-01	sblim	2001-07-01	2007-12-01
hibernate	2001-12-01	2005-12-01	springframework	2003-03-01	2007-12-01
jaffa	2003-03-01	2007-12-01	squirrel-sql	2001-12-01	2007-12-01
jena	2001-02-01	2008-02-01	xmsf	2004-02-01	2007-12-01
jmlspecs	2002-03-01	2007-12-01	yale	2002-04-01	2008-02-01

Using the schema described in section 2, we applied the Q-metric to the network extracted from each snapshot within the recorded period. In order to facilitate the presentation of the time evolution of these projects, we first compose all projects into four groups, according to the degree of fluctuation of the Q–metric. In Figure 3, we thus compute the mean

[3] `http://www.eclipse.org`

fluctuation in time of the Q-metric, i.e. $< Q(t+1) - Q(t) >$ where t and $t+1$ are consecutive snapshots of the software and the average $< \cdot >$ is over all snapshots in the dataset. This approach captures the average incremental change of the Q-metric over the observation period. In the same figure, we also show the standard deviation of $Q(t+1) - Q(t)$, which captures the degree of fluctuation of the changes in modularity over the same period. We performed a ranking of projects along both the average incremental change and the fluctuations of modularity and these rankings are indicated in the abscissae of the respective plots (see Figure 3).

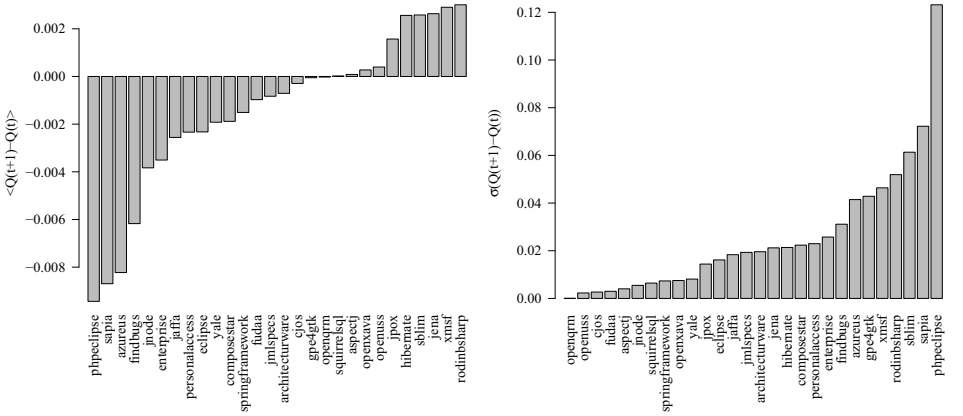


Figure 3: Ranking software projects using the Q-metric. (left) ranking by average incremental change of the Q-metric over the observation period, estimated with $< Q(t+1) - Q(t) >$. (right) ranking by degree of fluctuation of the changes in the modularity over the studied period, estimated with $\sigma(Q(t+1) - Q(t))$.

The resulting plot, with the projects grouped and ranked by the average incremental change of Q (see the left pannel of Figure 3), is shown in Figure 4. Here, we observe that the Q-metric effectively classifies projects according to different dynamic regimes. In Figure 3 (left) we can for instance focus on those projects that increase or decrease the software modularity, while Figure 3 (right) can be used to study the most dynamical and the most stable software development regimes.

In the following we discuss two projects with contrasting evolution of modularity in more detail. In particular, for this we chose the projects AZUREUS, which is a torrent client being one of the projects with the largest average decrease in the Q-metric, as well as JENA which is a framework for building semantic web applications. In our dataset JENA actually shows one of the largest average increase of Q (see the left plot in Figure 3). In Figure 5, the time trajectory of the evolution of Q is shown for both projects as a function of the total number of classes. As indicated in the Figures 5(a) and 5(b), three snapshots of the source code have been selected which cover the states of minimum and maximum modularity, as well as an intermediate state.

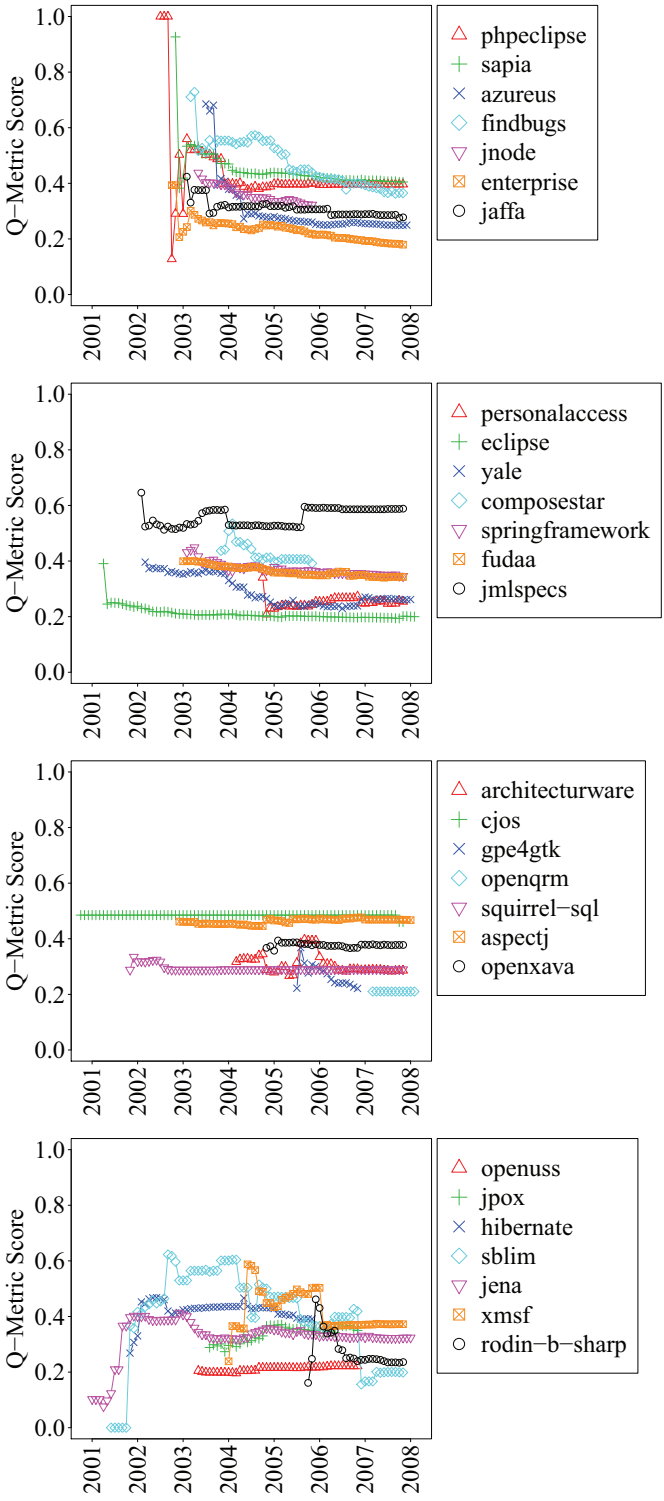
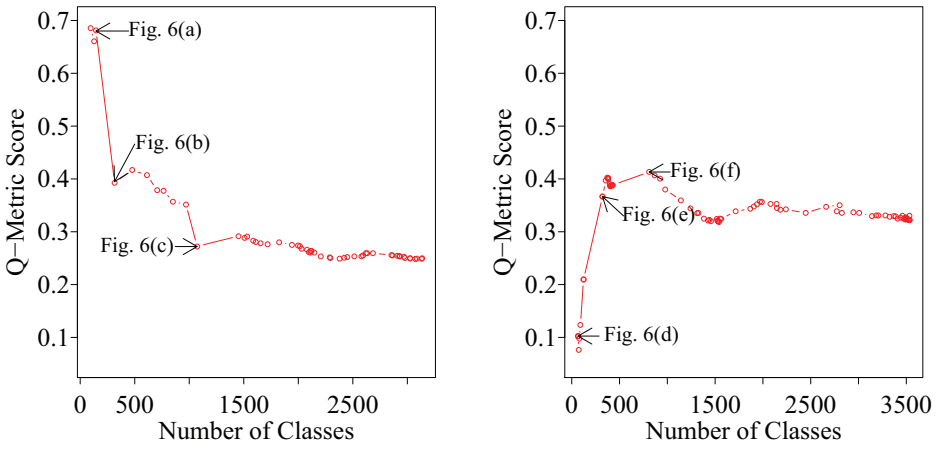
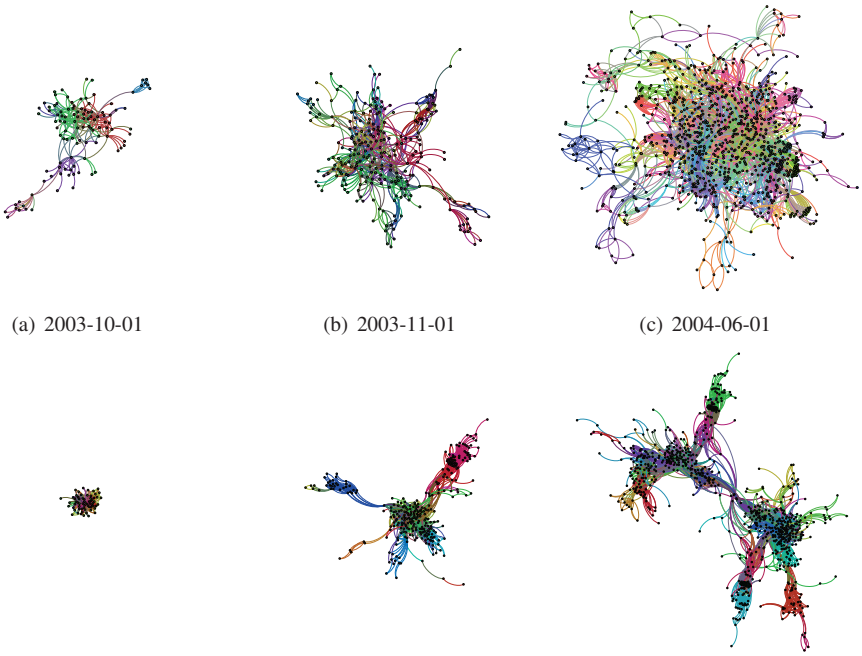


Figure 4: Time evolution of the Q-metric score for each project in our dataset. The projects were sorted by the mean fluctuation in time of the Q-metric, i.e. $< Q(t+1) - Q(t) >$, and displayed in increasing order of value (top-to-bottom). (top) highest mean decrease in Q. (bottom) highest mean increase in Q.



(a) AZUREUS (b) JENA

Figure 5: Detailed time evolution of the Q-metric for AZUREUS and JENA.



(a) 2003-10-01 (b) 2003-11-01 (c) 2004-06-01

(d) 2001-02-01 (e) 2001-10-01 (f) 2003-01-01

Figure 6: Three snapshots of the dependency networks of the projects AZUREUS (a-c) and JENA (d-f). Node colors in the individual networks indicate the decomposition in JAVA packages.

In Figure 6, we show the dependency networks for the snapshots mentioned above. These networks have been created according to the methodology described in section 2, i.e. each node represents a JAVA class, while a dependency indicates a call, inheritance or usage relationship. Furthermore, nodes have been colored according to package membership. In order to visualize the coherence between the package decomposition of the classes and the modular organization of the dependency network, the networks have been layouted with the force-directed Yifan-Hu layout algorithm [Hu05], which spatially organizes nodes according to cluster structures. In particular, nodes in networks with highly modular structures will be densely clustered in the resulting layouts and the modules will become clearly distinguishable. In the resulting networks we can visually examine how well the modular structures of the dependency network match the package structure of a project and thus obtain a visual impression of the module coherence expressed by the Q-metric.

The effect of the different dynamical regimes in terms of the evolution of the Q-metric can easily be seen in the respective network structures. For the AZUREUS project, which is shown is Figures 6(a) - 6(c), the coherence of the modular structure of the network of software dependencies with the package decomposition actually worsens over time, thus making it difficult to clearly separate packages in the resulting network structure. On the contrary, the evolution of the JENA project shows a very different dynamics. While the growth in terms of the number of nodes, packages and dependencies is in the same order of magnitude, the project maintains and even improves its modular decomposition, as is clearly shown in the Figures 6(d) - 6(f). From a software engineering perspective, the structure of JENA shown in Figure 6(f) is favorable, since it allows for an easy decomposition, maintenance and replacement of individual packages. One of the possible reasons for the discrepancy between JENA and AZUREUS is that the first is a framework aimed at an audience of developers. Thus, its structure must be well organized to facilitate its adoption, while the second is an end-user application and therefore the focus is on functionality rather than structural quality.

We are currently working on the extension of our approach in a way in which we hope to uncover the full potential of the Q-metric and its correlation with other software development processes, by modeling this dynamics as a simple network growth process with an underlying modular decomposition. This is the subject of ongoing research [ZSTS12]. Along the way we aim at improving our research methodology with more insights based on the network science framework as well as aligning it with existing results from the software engineering community. Prior to concluding this article and giving details on future research, in the next section we comment on related work.

4 Related Work

One of the eye catching features of the time evolution of the Q-metric, as presented in Figure 4, is the large fluctuation of Q at early stages of the project development. This is in accordance with results reported in [TGS11]. There, it was shown that young open source software projects display an accelerated growth rate while mature projects stabilize their dynamics and can grow further in a sustainable regime.

Another possible, complementary, reason for fluctuations are refactoring events, where software is usually rewritten or restructured in order to improve multiple features such as functionality, flexibility, reusability or structural quality. Such events could lead to the sudden jumps observed in Figure 4 along the time evolution of a software project. In [DDN00], refactoring metrics are proposed which take into account the dynamics of changing code. This line of research is well aligned with our purposes and can be easily adapted and augmented by our network perspective on software development processes.

For an early attempt of the application of network science to the analysis of software engineering processes we recommend [Mye03], which also contains a short review of classical approaches used in the software engineering literature. Finally, a recent article published in the PNAS journal used a similar network approach, though with a different metric, to study modularity of code and its relation to module survival, drawing a parallel to ecological systems and making use of a predator-prey model variation [FBL11].

5 Conclusion and Outlook

The results presented in section 3 indicate that the Q-metric known from the analysis of cluster structures in network science is a promising and reasonable approach to quantify the coherence between the package decomposition of large software projects and their dependency structures. As such, it constitutes a macroscopic measure that allows us to monitor and evaluate software engineering processes and reason about the sustainability of software architectures. In particular, it provides a simple mapping from local development activities to their respective impact on the mesoscopic and macroscopic structures of software systems. One of the problems of the current version of the Q–metric is that it is not scaled according to the size of the corresponding network, therefore making it hard to compare the Q score of different projects with vastly different sizes. This is a well known issue [FB07]. Although the current metric offers interesting insights, a further problem is that it is being influenced by intra-module dependencies. However it would be more thoughtful to look at the impact of inter-module dependencies because these are the most relevant dependencies in a modular structure. Last but not least, JAVA packages which were used as proxy for modularity in JAVA source code have a hierarchical structure. Therefore, dependencies between packages $a.b.c.d$ and $a.b.c.e$ are of less concern than between packages $a.b.c.d$ and $x.y.z$.

While all these issues are the subject to future investigations, our study already foreshadows a number of interesting research questions: How does the evolution of Q impact the sustainability of distributed software engineering efforts? Can the incorporation of such macroscopic measures into software development tools improve the design and maintenance of software architectures? How is the dynamics of Q over the lifetime of software projects correlated with software development acts like refactoring or bug fixing? How is it correlated with social aspects, coordination acts or communication processes taking place between developers? Intuitively, one would assume that a reasonable modular decomposition of complex software systems facilitates distributed development processes and mitigates change propagation between interdependent modules. An interesting future

work is thus to augment the results in this paper with data on coordination and communication acts in the respective projects. In this line of arguments, a further interesting question is whether the pronouncedness of modular structures in the dependency network allows us to infer statements about the hierarchical organization of development teams.

While the exploration of these questions in this study has been necessarily incomplete, we argue that the associated line of research is a good demonstration for the potential impact of complex systems science on the engineering of complex software systems.

Acknowledgment

We acknowledge the financial support provided by the Swiss National Science Foundation through grant CR12I1_125298 and also Ingo Scholtes, Claudio Juan Tessone and our three reviewers for valuable comments.

References

[AB02] R. Albert and A. L. Barabási. Statistical mechanics of complex networks. *Reviews of modern physics*, 74(1):47, 2002.

[BHJ09] M. Bastian, S. Heymann, and M. Jacomy. Gephi: An Open Source Software for Exploring and Manipulating Networks. In *Proceedings of the ICWSM '09*. AAAI, 2009.

[DDN00] S. Demeyer, S. Ducasse, and O. Nierstrasz. Finding refactorings via change metrics. *ACM SIGPLAN Notices*, 35(10):166–177, 2000.

[FB07] S. Fortunato and M. Barthelemy. Resolution limit in community detection. *Proceedings of the National Academy of Sciences*, 104(1):36, 2007.

[FBL11] M. A. Fortuna, J. A. Bonachela, and S. A. Levin. Evolution of a modular software network. *PNAS*, 2011.

[GJM03] Carlo Ghezzi, Mehdi Jazayeri, and Dino Mandrioli. *Fundamentals of Software Engineering*. Prentice Hall, 2nd edition, 2003.

[GS11] M. M. Geipel and F. Schweitzer. The Link between Dependency and Co-Change: Empirical Evidence. *IEEE Transactions on Software Engineering*, 2011.

[HR92] S. Horwitz and T. Reps. The use of program dependence graphs in software engineering. In *ICSE Proceedings*, pages 392–411. ACM, 1992.

[Hu05] Y. Hu. Efficient, high-quality force-directed graph drawing. *Mathematica Journal*, 10(1):37–71, 2005.

[Koh09] G. A. Kohring. Complex Dependencies in Large Software Systems. *Advances in Complex Systems*, 12(6):565–581, 2009.

[Mye03] C. R. Myers. Software systems as complex networks: Structure, function, and evolvability of software collaboration graphs. *Physical Review E*, 68(4):046116, 2003.

[New03a] M. E. J. Newman. Mixing Patterns in Networks. *Phy. Review E*, 67:026126, 2003.

[New03b] M. E. J. Newman. The structure and function of complex networks. *SIAM review*, pages 167–256, 2003.

[New10] M. E. J. Newman. *Networks: an introduction*. Oxford Univ Press, 2010.

[NG04] M. E. J. Newman and M. Girvan. Finding and evaluating community structure in networks. *Phy. Review E*, 69:026113, 2004.

[PCW85] D. L. Parnas, P. C. Clements, and D. M. Weiss. The modular structure of complex systems. *Software Engineering, IEEE Transactions on*, 11(3):259–266, 1985.

[TGS11] C. J. Tessone, M. M. Geipel, and F. Schweitzer. Sustainable growth in complex networks. *Europhysics Letters*, 96:58005, 2011.

[ZSTS12] M. S. Zanetti, I. Scholtes, C. J. Tessone, and F. Schweitzer. Evolution of Software Modularity. In Preparation, 2012.

VERFE 2012

Workshop on Dependability and Fault Tolerance

organized by

Karl-Erwin Großpietsch, Fraunhofer IAIS, Germany
Jörg Henkel, Karlsruhe Institute of Technology, Germany

Foreword

The growing use of networked computers in work environments as well as individual homes has demonstrated the importance of system dependability also to a broader public. The criticality of dependable, fault-tolerant IT system design will further increase in the future because continuously decreasing feature sizes of standard CMOS semiconductor technologies cause the fundamental device operation to become increasingly sensitive to various forms of fabrication and environment related variations. Designing dependable systems from inherently unreliable components conbstitutes one of the grand challenges of technical informatics. To thoroughly address and solve this fundamental problem, a variety of approaches is being developed and discussed. It is the aim of this workshop to present contributions to the mentioned area and to bring together scientists working in this field.

The Joint Technical Committee on Dependability and Fault Tolerance (VERFE) of the three German computing societies, Gesellschaft für Informatik (GI), Informationstechnische Gesellschaft (ITG), and Gesellschaft für Mess- und Automatiserungswesen (GMA) is continuously working to support and coordinate activities in the research area of dependable computing within the German research community. An example is the DFG SPP 1500 (see http://spp1500.itec.kit.edu). With this year´s ARCS conference event, we proceed in a series of workshops which has started more than twelve years ago in connection with the ARCS 1999 conference.

The programme committee has selected ten contributions to be presented in the workshop programme. The contributions are dealing with fault diagnosis and fault tolerance of hardware systems, dependable software, and dependability of autonomous systems. One invited talk completes the program.

Finally, we would like to thank the members of the programme committee for their reviewing work, and the local organizers of the ARCS 2012 conference for their activities to organize this workshop event.

Sankt Augustin / Karlsruhe, January 2012

K.-E. Großpietsch, J. Henkel

Workshop Chairs

Programme Committee

L. Bauer, Karlsruhe
F. Belli, Paderborn
M. Berekovic, Braunschweig
O. Bringmann, Paderborn
G. Bronevetsky, Livermore, CA, USA
R. Buchty, Karlsruhe/Tübingen
K. Echtle, Essen
W. Ehrenberger, Fulda
R. Ernst, Braunschweig
R. Brück, Siegen
B. Fechner, Augsburg
F. Freiling, Mannheim
M. Gössel, Potsdam
E. Gramatova, Bratislava
J. Hülsemann, Karlsruhe
K.-E. Großpietsch, St. Augustin (Chair)
J. Henkel, Karlsruhe (Chair)
J. Hülsemann, Karlsruhe
J. Hursey, Oak Ridge, TN, USA
M. Imai, Tokyo, Japan
J. Kaiser, Magdeburg
J. Keller, Hagen
V. Kharchenko, Ukraine
H.-D. Kochs, Duisburg
P. Limbourg, Essen
M. Malek, Berlin
E. Maehle, Lübeck
E. Nett, Magdeburg
D. Nikolos, Patras
A. Pataricza, Budapest
W. Rosenstiel, Tübingen
F. Saglietti, Erlangen
T. Sato, Fukuoka, Japan
M. Schulz, Livermore, CA, USA
M. Shafique, Karlsruhe
P. Sobe, Dresden
J. Sosnowski, Warsaw
A. Stopp, Berlin
C. Trinitis, München
P. Tröger, Potsdam
T. Vierhaus, Cottbus
M. Walter, Nürnberg
H. Wedde, Dortmund
N. Wehn, Kaiserslautern
J. Weidendorfer, München
H.-Y. Youn, Sungkyunkwan, Korea
T. Yoneda, Japan

Fault Localization in NoCs by Timed Heartbeats

Bernhard Fechner, Arne Garbade, Sebastian Weis, Theo Ungerer
Department of Computer Science
University of Augsburg
Universitaetsstr. 6a
D-86159 Augsburg
{fechner, garbade, weis, ungerer}@informatik.uni-augsburg.de

Abstract: Future computing systems will contain more and more cores on a single die. Permanent faults occur not only during manufacturing but may also arise at runtime. To detect these faults, a group of cores is monitored by a single unit, receiving heartbeats from all cores. In this paper, we present a simple method to localize permanent faults in a 2D mesh-based NoC by using heartbeats and by measuring the time from source (core) to destination (monitoring unit). We introduce a heartbeat network along with the normal application message network to guarantee a deterministic heartbeat timing and no interferences with application messages. If the time for a heartbeat exceeds a given interval, it can be concluded that the heartbeat is missing or delayed, e.g. because of a faulty core, link or router. As this is not sufficient to localize a fault, we introduce the concept of Timed Heartbeats, which uses different routing directions in contrary to the intended routing to introduce a fixed, additional delay for rerouted heartbeats. The delay helps to localize the fault without any additional bandwidth consumption.

1 Introduction

Advances in VLSI technology enable to integrate a high number of cores on a single die, communicating over an interconnection network. Examples are the 64-core TILE64 from Tilera [BGH86], the experimental Intel Polaris 80-core [Dii09], or the recently presented 1024-core accelerator Rigel [JJK+11]. All of these many-core chips include packet-based Network-on-Chips (NoC) in order to scale with the number of cores and reduce the hardware complexity of the processor. In particular, 2D mesh topologies are currently popular because they combine scalability with low design complexity and support simple dimensional ordered routing algorithms. Therefore, we take the 2D mesh as our baseline topology.

Due to shrinking feature sizes, rising chip temperatures etc. more and more permanent, intermittent, and transient faults in cores, routers, and links can occur. These can have many manifestations—not only at manufacturing time but mainly during operation. We assume permanent link and router faults and that the end-to-end communication within the application message network (AMN) is secured against transient faults, i.e. safeguarded with error detecting/correcting codes (EDC/ECC).

To check whether a component (core or router) of a many-core system is alive, we assume a monitoring architecture similar to [WGSU11], where periodic heartbeats are sent from each component to a monitoring unit, called Fault Detection Unit (FDU). The FDU is the abstraction of a monitoring unit. It can be implemented either as a complex circuit with the ability to dynamically analyze the information gathered from the monitored components (running sophisticated algorithms) and starting actions to maintain the functionality of the chip or (as in this work) as simple unit collecting information about faults.

The proposed Timed Heartbeat technique uses the arrival times of periodically sent heartbeats to localize link and router faults. By comparing the expected (deterministic) arrival time with the actual arrival time, it can be concluded if a heartbeat was rerouted due to a fault or not. A simple switching of the routing algorithms allows to localize the fault. Please note, that this technique can be applied to any topology as long as arrival times of heartbeats are deterministic.

We introduce a heartbeat network (HN) along with the normal application message network (AMN) to guarantee that the heartbeat timing is deterministic and heartbeats do not interfere with the much larger application messages. In this work, we mean the HN if not otherwise stated. In comparison with a standard packet-based communication, we do not have any additional traffic overhead (packet header, fixed packet length, EDC/ECC) and therefore can save bandwidth $((N-1)*(length\_of\_packet-1))$ and power each network cycle.

This paper is structured as follows: Section 2 discusses related work and Section 3 provides a description of the basic network architecture. Section 4 presents the localization of faults with Timed Heartbeats. Section 5 concludes the paper. For the convenience of the reader, we provide a list of abbreviations in Table 1.

Abbreviation	**Meaning**
AMN	Application Message Network
(B)MQ	(Bandwidth) Multi-Quadrant
(B)SQ	(Bandwidth) Single-Quadrant
EAT	Expected Arrival Time
FDU	Fault-Detection Unit
HN	Heartbeat Network
MAT	Measured Arrival Time
MD, d_m	Manhattan Distance
NoC	Network on Chip
QHC	Quadrant Heartbeat Cycle

Table 1: List of abbreviations

2 Related work

Heartbeats are a well-known timing-based mechanism to detect faults in distributed systems [CT96, BMS02, SPTU07, SND11]. Examples can be found in many early commercial fault-tolerant systems such as Tandem [BGH86] and later in the Globus Heart-

beat Monitor [SFK+98]. Details on routing problems and algorithms can be found in [GHKS98]. More details on heartbeats and timing intervals in multicomputers can be found in [HS98].

Steinert and Gillblad [SG10] recently proposed to use collaborating nodes to locate a fault in a distributed system. They applied statistical methods to compute the expected timing from measured network delays. However, this requires the costly evaluation of a probability density function. In contrast to fault localization in distributed systems, our work exploits the determinism of on-chip interconnects for a much simpler localization of link and router faults.

3 Basic network architecture

As topology we assume a (n, m) 2D-mesh, where $N = n * m$ is the number of cores. For simplicity, we assume the mesh as quadratic ($m = n$). Let $G = (V, E)$ be a graph G with a set of vertices V and edges E. The number of cores N is given by $N = |V|$. Since one router is directly associated to a core, the number of routers is $R = N$. The number of edges is $|E| = 2(N - \sqrt{N})$. The mesh has a diameter of $2\sqrt{N} - 1$ and a maximal degree of 4. The mesh (s. Figure 1 and 2) is divided in four quadrants (North-West [NW], North-East [NE], South-West [SW], South-East [SE]), called *multi-quadrant* (MQ) and four quadrants North [NO], South [SO], East [EA], West [WE], called *single-quadrant* (SQ). We regard one MQ (NW) and two SQs (NO, WE) since all other quadrants can be handled analogously. The communication of the core to the router and vice-versa is accomplished via a local link. The router has four additional ports (North [NO], South [SO], East [EA], West [WE]) for the communication to its neighbors. Links are assumed as bidirectional. Several possibilities concerning the communication times can be made, since there are four neighboring nodes and two wire delays (Δx, Δy), introducing different skews between nodes. We assume the simplest case where all wire delays are (for every direction) equal to one (Δx=Δy=1). Since the FDU resides in the center, the transmission delays are equal to the Manhattan distance $d_M(x, y) = |x| + |y|$.

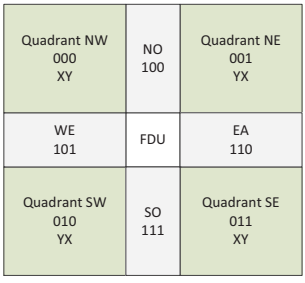


Figure 1: Coding of Quadrants

3.1 Sending of Heartbeats

All cores send heartbeats to the FDU residing in the center of all quadrants (s. Figure 2). The localization of a faulty link or router is simple, if we allow to send the ID of a faulty link or router to the FDU, complicating the router design. We assume that the FDU (router) is able to handle one heartbeat at a time. It is possible to handle more heartbeats at a time, since the FDU can be implemented effectively by using a single decrementer and a table holding all excepted/measured arrival times.

The heartbeats propagate each network clock cycle from router to router. A heartbeat transmission is always initiated from a core via the local router with the FDU as destination. It can only be sent if all local links (one for the heartbeat and the ones for the AMN) are fully functional. The links are considered as functional, iff no fault occurred before. The router therefore has a counter for each link, holding the number of detected faults in the AMN. Additionally, we can measure the time from one fault to another, e.g. to relate the number of faults in time to be able to detect permanent faults.

3.2 Bandwidth composition, pure XY routing

Regarding XY-routing for heartbeat transmission, the heartbeat is first routed in X direction (dimension 0) until it reaches the destination column, then in Y direction (dimension 1). Each router has a field, which determines the quadrant relative to the FDU. The location field is used for routing decisions. Figure 1 shows the coding scheme for each quadrant of the network. The first bit determines if the quadrant is a MQ (set to 0) or SQ (set to 1). Figure 2 shows details about fault location, mesh assembly, distances and quadrants.

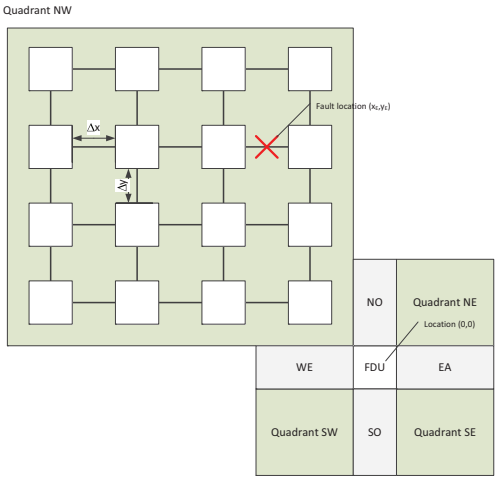


Figure 2: Quadrants, mesh assembly and distances

Routers located in the SQs NO and SO are the only ones routing in Y direction. All others route in X direction. Since we have one FDU and a quadratic mesh, the maximum number of cores sending to the FDU concerning each MQ/SQ can be easily computed. For SQs WE, EA, NO, SO: $BSQ = \frac{\sqrt{N}-1}{2}$. For MQs NE, NW, SE, SW: $BMQ = BSQ * BSQ = \frac{1}{4}(\sqrt{N}-1)^2$. The total bandwidth of heartbeats in relation to each router input (R_{NO}, R_{EA}, R_{SO}, R_{WE}) results to $R_{SO} = R_{NO} = 2BMQ + BSQ$, $R_{EA} = R_{WE} = BSQ$. We check the total number of cores: $4BMQ + 4BSQ = (\sqrt{N}-1)^2 + 2\sqrt{N} - 2 = N - 1$. Obviously the number of heartbeats is unfairly distributed over the SQs (NO, SO). The solution is to alternate between XY and YX routing (s. Figure 1) for different quadrants to distribute the number of heartbeats over all SQs.

4 Fault localization in a NoC with Timed Routing

4.1 Receiving timing information

A fault can either occur at a router or link. We assume that a permanent (link or router) fault can be detected by the router or a neighboring router due to e.g. high impedance of the link or the methods briefly sketched in Section 3. If a link fault has been detected, the appropriate router will reconfigure itself to route around the fault. In subsection 4.4 we describe, how this can be accomplished. The FDU receives the heartbeats and compares the arrival times with the stored timing information, which is equal to the Manhattan distance (MD). The heartbeat send pattern ensures that we have no conflicts, i.e. not more than a single heartbeat arrives at a router. The heartbeat pattern is like a rope of pearls arriving at the FDU, first heartbeats of SQs then heartbeats of MQs starting from the bottommost (MQ NW) to the topmost row. To ensure that we have no conflicts, a delay must be introduced from row to row. The tables $EAT_{\{NW,NO,NE,EA,SE,SO,SW,WE\}}$ in the FDU hold the expected arrival times for heartbeats for each quadrant. For example, the table for the quadrant EAT_{NW} is $\begin{pmatrix} n^2 & \dots & n \\ \vdots & \ddots & \vdots \\ n & \dots & 1 \end{pmatrix}$, whereas the following matrix holds the times when heartbeats are sent: $\begin{pmatrix} n^2 & \dots & n^2-n+1 \\ \vdots & \ddots & \vdots \\ n & \dots & 1 \end{pmatrix}$. The FDU associates the heartbeat to a table entry of the measured arrival times (MAT) according to its specific arrival time. The value in the MAT is then decremented. We must therefore conduct four subtractions in parallel in each network clock cycle. Since the results with equal MD are the same, we need to compute it once per quadrant table for a single MD and distribute it to the other tables, if all heartbeats arrived. If a heartbeat arrives too late, the according value will be less than zero. Alternatively, we can wait until all heartbeats arrived, then XOR the MAT with the EAT, $F = EAT \oplus MAT$. The fault matrix F signals any differences between the expected and measured arrival times, iff $F \neq 0$. In the following, a *deviation* means a deviation in the timing of a single heartbeat. To generate the heartbeat at a specific time,

only a single decrementer is needed. If the decrementer triggering the heartbeat is defect (permanent or transient), this will lead to a deviation. Therefore, the decrementer must not be protected against faults.

4.2 Localization of faulty links

On link-level, a fault is assumed to occur permanently at position (x_L, y_L). Note, that we still regard the MQ NW.

Horizontal link fault (XY-routing): If a link fault occurs at position (x_L, y_L), obviously no router above $> y_L$ or below $< y_L$ the fault position will have to reroute, not leading to any deviation. Therefore the row in which the fault occurred can be localized perfectly. All (fault-free) links with position $< x_L$ will not lead to a deviation. All heartbeats initiated leftwards $\geq x_L$ will arrive later. The fault matrix

$$\begin{pmatrix} 0 & \dots & 0 \\ \vdots & \ddots & \vdots \\ 1 & \dots 1 & 0 \end{pmatrix}$$

illustrates this. The most rightwards "1" is the router with the faulty link, since it signals the first occurrence of the fault and thus the first deviation. Naturally, the fault can be perfectly localized in this case, but in the worst case, we must wait BSQ network cycles. XY-routing does not provide information about faulty links in Y direction in the MQ NW (other quadrants analogously). Therefore, the routing algorithm is switched from XY to YX routing after all heartbeat messages of a quadrant arrived (a quadrant heartbeat cycle, QHC).

Vertical link fault (YX-routing): Obviously no router with position left ($> x_L$) or right ($< x_L$) of the fault will have to reroute, not leading to any deviation. Therefore the column in which the fault occurred can be localized. All links below or above the fault ($< y_L$, $\geq y_L$) will lead to an increased timing. The fault matrix

$$\begin{pmatrix} 0 \dots & 1 & 0 \\ \vdots & \ddots & \vdots \\ 0 \dots & 1 & 0 \end{pmatrix}$$

depicts this.

4.3 Localization of faulty routers

A router fault is modeled by a maximum of five concurrently occurring link faults (note, that core and router faults are not distinguishable and we therefore do not regard faulty local links). The router is assumed to exhibit a fail-stop behavior. Thus, it is not able to send or receive messages any more after a fault. Concerning XY-routing two link faults have to be regarded (WE and EA, YX analogously). A single router failed, if the FDU

receives only $N-2$ heartbeats in a given time interval. If there is a faulty router at position (x_R, y_R), no router above $> y_R$ or below $< y_R$ the fault position will have to reroute (due to the assumed XY-routing). As for link faults, the row in which the fault occurred can be localized. All cores located right the fault $< x_R$ are able to produce heartbeats. All heartbeats from cores leftwards $\geq x_R$ are rerouted and introduce an additional delay. Since the routing algorithm is switched from XY to YX routing, we detect two faulty links in two successive QHCs and missed one heartbeat in each QHC. The following fault matrix illustrates the situation of a router fault in the bottommost row after two QHCs:

$$F_{bottommost} = \begin{pmatrix} 0\ldots & 1 & 0 \\ \vdots & \ddots & \vdots \\ 1\ldots & 1 & 0 \end{pmatrix}.$$

We can also distinguish router or link faults in the topmost row of the mesh. Since the routing algorithm is switched from XY to YX routing, we detect faulty horizontal links in one QHC and missed one heartbeat in each QHC. F depicts the situation after two QHCs:

$$F_{topmost} = \begin{pmatrix} 1\ldots & 1 & 0 \\ \vdots & \ddots & \vdots \\ 0\ldots & 0 & 0 \end{pmatrix}.$$

4.4 Rerouting around a faulty link

Until now, we did not specify, how to reroute around a faulty link. There can be several routing possibilities. Figure 3 shows the possibilities for a link fault rightwards core S for a heartbeat propagating to core T. We distinguish two main cases:

1. it is allowed to route back in the direction the heartbeat came from: (s. Figure 3) if we allow to reroute back (route 3), in the MQ NW (XY routing QHC), we have two possibilities, route westwards (WE, route 3), then to the north (NO, route 4, delay +4) or south (SO, route 3, delay +2). Since the router receives a heartbeat from an unexpected direction (EA instead WE), it knows that a fault occurred and that it has to reroute to a different location (not in the direction of the fault, EA).
2. if this is disallowed: the heartbeat can be rerouted in such a way that no delay occurs (route 1, SO) or with route 2, delay +2, NO.

Port uesd as input	i
Port used as output	o
Port has permanent fault	x
Delay introduced by rerouting	(+j)

Table 2: Notation used in Table 3

Table 3 lists the routing combinations for all quadrants, whereas we use the notation introduced in Table 2.

Type	Quadrant	From	To	Fault	Reroute	Reroute back
MQ	NW	iWE	oEA	xNO	–	–
		iWE	oEA	xSO	–	–
		iWE	oEA	xEA	NO(+2),SO(+0)	WE,NO(+4); WE,SO(+2)
SQ	NO	iNO	oSO	xWE	–	–
		iNO	oSO	xEA	–	–
		iNO	oSO	xSO	WE(+2),EA(+2)	NO,(WE,EA)(+4)
MQ	NE	iEA	oSO	xNO	–	–
		iEA	oSO	xSO	–	–
		iEA	oSO	xWE	NO(+2),SO(+0)	EA,NO(+4); EA,SO(+2)
SQ	EA	iEA	oWE	xNO	–	–
		iEA	oWE	xSO	–	–
		iEA	oWE	xWE	NO(+2),SO(+2)	EA,(NO,SO)(+4)
MQ	SE	iEA	oWE	xNO	–	–
		iEA	oWE	xSO	–	–
		iEA	oWE	xWE	NO(+0),SO(+2)	EA,NO(+4); EA,SO(+2)
SQ	SO	iSO	oNO	xWE	–	–
		iSO	oNO	xEA	–	–
		iSO	oNO	xNO	WE(+2),EA(+2)	SO(WE,EA)(+4)
MQ	SW	iWE	oEA	xNO	–	–
		iWE	oEA	xSO	–	–
		iWE	oEA	xEA	NO(+2), SO(+2)	WE,NO(+4); WE,SO(+2)
SQ	WE	iWE	oEA	xNO	–	–
		iWE	oEA	xSO	–	–
		iWE	oEA	xEA	NO(+2),SO(+2)	WE,(NO,SO)(+4)

Table 3: Routing combinations for all quadrants

4.5 Congestions and optimality

The FDU receives heartbeats from $N-1$ cores. Since the FDU router was assumed to accept one heartbeats at a time, optimally in time $N-1$ all heartbeats arrived. Note, that the faster the heartbeats arrive, the faster we are able to locate and handle a fault. Two cases must be distinguished to avoid congestions (assuming that different quadrants are not allowed to simultaneously issue heartbeats):

1. No fault occurred: If we allow to send heartbeats from the bottommost row first, we have to wait row-1 times for each MQ (SQ first).
2. A heartbeat must be rerouted due to a fault: We consider the MQ NW and XY routing. Obviously, no congestion occurs, if we reroute in a direction causing no deviation (s. Figure 3, route 1). This brings no advantage, since we want to locate faults with the introduced delays. Let d_0 be the introduced delay due to a reroute. Then all nodes sending heartbeats with $MD \geq d_0$ will cause a conflict. The solution is simple: We introduce a delay from row to row which is exactly the number of elements in a row.

5 Summary, conclusions and future work

In this paper, we presented a mechanism to localize and distinguish faults on router and link level in NoCs with timing as sole information. Cores are sending heartbeats to the FDU. We extended the existing XY-routing to receive different timings for rerouted messages. We concluded and showed that the position of a fault can be determined. The simple heartbeat network saves bandwidth and energy and enables a faster detection since the heartbeat network can have higher clock rates in comparison with the AMN due to the simple assembly of heartbeats. The method is able to localize four concurrent link and router faults in different MQs perfectly. Furthermore, our technique can be applied to any network topology as long as arrival times of heartbeats are deterministic. Many interesting research opportunities result: what is the relation between different routing algorithms, the localization accuracy and the mean time to detect a fault? How precisely

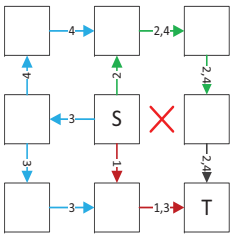


Figure 3: Rerouting around a faulty link

can this combination locate a fault under the influence of multiple faults in a single MQ or SQ? Is there a lower bound concerning the timing for all rerouted messages? Our future work will consider these open questions and also multiple link and router faults.

Acknowledgements. This work was partly funded by the European FP7 project TERAFLUX, ID 249013, http://www.teraflux.eu.

References

[BGH86] J. Bartlett, J. Gray, and B. Horst. Fault Tolerance in Tandem Computer Systems. Technical report, Tandem Technical report 86.2, 1986.

[BMS02] Marin Bertier, Olivier Marin, and Pierre Sens. Implementation and Performance Evaluation of an Adaptable Failure Detector. In *Proceedings of the 2002 International Conference on Dependable Systems and Networks*, DSN '02, pages 354–363, Washington, DC, USA, 2002. IEEE Computer Society.

[CT96] Tushar Deepak Chandra and Sam Toueg. Unreliable failure detectors for reliable distributed systems. *J. ACM*, 43:225–267, March 1996.

[Dii09] S. Diighe. Lessons Learned From The 80-Core Tera-Scale Research Processor. In *Intel Technology Journal*, volume 13, 2009.

[GHKS98] Miltos D. Grammatikakis, D. Frank Hsu, Miro Kraetzl, and Jop F. Sibeyn. Packet Routing In Fixed-Connection Networks: A Survey, 1998.

[HS98] Seungjae Han and Kang G. Shin. Experimental Evaluation of Failure-Detection Schemes in Real-time Communication Networks. In *in Proc. IEEE FTCS*, pages 122–131, 1998.

[JJK+11] D.R. Johnson, M.R. Johnson, J.H. Kelm, W. Tuohy, S.S. Lumetta, and S.J. Patel. Rigel: A 1,024-Core Single-Chip Accelerator Architecture. *Micro, IEEE*, 31(4):30 –41, july-aug. 2011.

[SFK+98] P. Stelling, I. Foster, C. Kesselman, C. Lee, and G. Von Laszewski. A fault detection service for wide area distributed computations. In *High Performance Distributed Computing, 1998. Proceedings. The Seventh International Symposium on*, pages 268 –278, jul 1998.

[SG10] R. Steinert and D. Gillblad. Towards Distributed and Adaptive Detection and Localisation of Network Faults. In *Telecommunications (AICT), 2010 Sixth Advanced International Conference on*, pages 384 –389, may 2010.

[SND11] A. Shikri, M. Noor, and M. Deris. Dynamic-Hybrid Fault Detection Methodology. In *Journal of Computing*, volume 3, pages 58–64, 2011.

[SPTU07] Benjamin Satzger, Andreas Pietzowski, Wolfgang Trumler, and Theo Ungerer. A new Adaptive Accrual Failure Detector for Dependable Distributed Systems. In *ACM Symposium on Applied Computing (SAC 2007)*, pages 551–555, 2007.

[WGSU11] Sebastian Weis, Arne Garbade, Sebastian Schlingmann, and Theo Ungerer. Towards Fault Detection Units as an Autonomous Fault Detection Approach for Future Many-Cores. In *ARCS 2011 Workshop Proceedings (SCAFT Workshop)*, pages 20–23. VDE Verlag, February 2011.

FAIL*: Towards a Versatile Fault-Injection Experiment Framework*

Horst Schirmeier[1], Martin Hoffmann[2],
Rüdiger Kapitza[3], Daniel Lohmann[2], and Olaf Spinczyk[1]

[1]Department of Computer Science 12, Technische Universität Dortmund,
e-mail: {horst.schirmeier, olaf.spinczyk}@tu-dortmund.de
[2]Department of Computer Science 4, Friedrich-Alexander-Universität Erlangen-Nürnberg,
e-mail: {hoffmann, lohmann}@cs.fau.de
[3]Institute of Operating Systems and Computer Networks, TU Braunschweig,
e-mail: kapitza@ibr.cs.tu-bs.de

Abstract: Many years of research on dependable, fault-tolerant software systems yielded many tool implementations for vulnerability analysis and experimental validation of resilience measures. We identify two disjoint classes of fault-injection (FI) experiment tools in the field, and argue that both are plagued by inherent deficiencies, such as insufficient target state access, little or no means to switch to another target system, and non-reusable experiment code.

In this article, we present a novel design approach for a FI infrastructure that aims at combining the strengths of both classes. Our FAIL* experiment framework provides carefully-chosen abstractions simplifying both the implementation of different simulator/hardware target backends and the reuse of experiment code, while retaining the ability for deep target-state access for specialized FI experiments. An exemplary report on first experiences with a prototype implementation based on existing x86 and ARM simulators demonstrates the tool's versatility.

1 Motivation and State of the Art

Recent technology roadmaps [Bor05, DYdS+10, NX06] suggest that future hardware designs for embedded systems will exhibit an increasing rate of intermittent errors in exchange for a life extension for Moore's Law—in terms of even smaller device sizes, lower energy consumption, decreased per-transistor costs, and more performance. This bears new challenges for software developers, which must incorporate software fault-tolerance measures to compensate for unreliable hardware while still benefitting from these new designs: An application-specific resource-efficiency/dependability tradeoff must be made, only hardening mission-critical parts of the software stack against hardware faults. The remaining components must economize resource consumption and are endorsed to yield wrong results, or fail in other modes.

*This work was partly supported by the German Research Foundation (DFG) priority program SPP 1500 under grant no. KA 3171/2-1, LO 1719/1-1 and SP 968/5-1.

Fault-injection (FI) experiments and dynamic trace analyses are common means to analyze a complex software-stack's susceptibility to hardware faults, and to assess the effectivity of previously applied software fault-tolerance measures [BP03]. Repeating analysis/evaluation and software-hardening steps allows system designers to converge to an application-specific tradeoff eligible for their product.

In this context, often an ad-hoc solution—highly specific to the assessed software, the current target platform, and a particular fault model—is chosen, resulting in non-reusable tools. As an unfortunate side effect, such tools—although non-negligible efforts were spent on them—are rarely published themselves, hindering experiment reproduction and forcing the community to consistently reinvent the wheel.

Over the last decades, this situation was improved by a multitude of dedicated FI experiment tool suites, each targeting different development phases and fault models, based on a zoo of hardware simulators at varying levels of simulation accuracy, or on physical prototype hardware accessed through debugging interfaces [ZAV04]. These tools can be partitioned into *generalists* and *specialists*:

The *generalists* claim a certain level of flexibility regarding the target-platform backend. Among the benefits of this approach is that experiments can more easily be reused on a different platform—e.g., for gaining evidence the tested fault-tolerance measure is not platform-specific, or to move from a simulator backend to a real hardware prototype in later development phases. With GOOFI, Aidemark, Skarin et al. presented such a generic FI framework, abstracting away target systems in a plugin-based architecture [AVFK01, SBK10], and additionally providing extensive pre- and post-experiment analysis methods [BVFK05]. Fidalgo et al. [FGAF06] describe a generic tool addressing FI via the NEXUS on-chip debugger interface. Another example is QINJECT (David et al., [DCCC08]), injecting faults into a target backend utilizing the GDB debugger interface. These approaches have the common disadvantage that the chosen interface between experiment engine and target backend heavily limits access to target-system state, and narrows the possibilities for FI—e.g., obstructing the possibility to inject networking-device–specific faults into QEMU in the latter example.

In contrast, the *specialist* tools are highly specific to a single target. An example is FAUMACHINE (Sieh et al., [SPS09]), which provides access to a large part of its x86 simulator's state, and enables various FI methods, including, e.g., hard-disk faults. David et al. modified QEMU in [DC07], which also allows for deep simulator state access. But despite the advantage of providing access to the backend's full capabilities, this class of tools is characterized by severe maintainability issues: Deep state-access usually results in deep intrusion into the backend's code-base. The resulting *tight coupling* between simulator and FI code often complicates or even inhibits exchanging the tool's target backend later on; in the case of tools that were forked from an existing hardware simulator, such as QEMU, keeping in sync with the simulator's evolution is often too arduous, soon resulting in an outdated FI platform. From an experimenter's point of view, the most notable side effect is the fact that his/her experiment setups are bound to the chosen specialist/backend couple and completely unportable, e.g., to re-run the same experiment with prototype hardware or another target CPU.

A compromise between generalists and specialists—combining their strengths regarding target-backend flexibility, experiment reuse, and deep target-state access—seems desirable. The contribution of this article therefore is:

- A novel design approach for a fault-injection experiment framework that allows *switching target backends* with little effort (Sec. 2),
- a framework API abstracting away target-backend details and thereby fostering *experiment code reuse* (Sec. 2),
- and a first *experience report* from our FAIL* tool prototype implementation (Sec. 3).

The paper concludes with a discussion and an outlook on future work in Sec. 4 and 5.

2 FAIL*: Design and Implementation

Based on the needs emerging in our DANCEOS project—a research endeavor in the context of fault-tolerant embedded operating systems—and the state of the art described in the previous section, we are developing FAIL*[1] aiming at combining the advantages of generalists and specialists while avoiding their drawbacks. In the following, we elaborate on design decisions regarding the tool's architecture and its API, and give some details on our prototype implementation based on existing X86 and ARM simulators.

2.1 Architecture

Two main ideas stand behind the architecture of FAIL*: A modularization scheme chosen specifically for a flexible interchangeability of target backends and for distributing experiments in a parallel environment, and an experiment API designed with the right choice of abstractions in mind for experiment portability and implementation ease-of-use.

Fig. 1 gives an overview of FAIL*'s architecture. User-defined experiments (green) are split up by the user in a *Campaign* and a *Fault Injection* part, which communicate by means of *parameter sets*: A FI campaign typically consists of a potentially large amount of independent single experiments that only differ in the specific fault vector, which can be described in a parameter set. At this point, it should be noted that FAIL* is not limited to fault injection. Other possible parameter sets can be series of input vectors of software components allowing extensive integration testing.

The campaign generates a series of parameter sets that are distributed among several (possibly distributed) FAIL* client instances, each iteratively running FI experiments, consuming parameter sets, and communicating back results to the *Campaign Controller*. The FI experiment controls its local target backend through a *Simulator Abstraction* layer,

[1] Fault Injection Leveraged; the wildcard operator * stands for exchangeable target backends, e.g., FailBochs representing an instantiation with the Bochs x86 simulator as the backend.

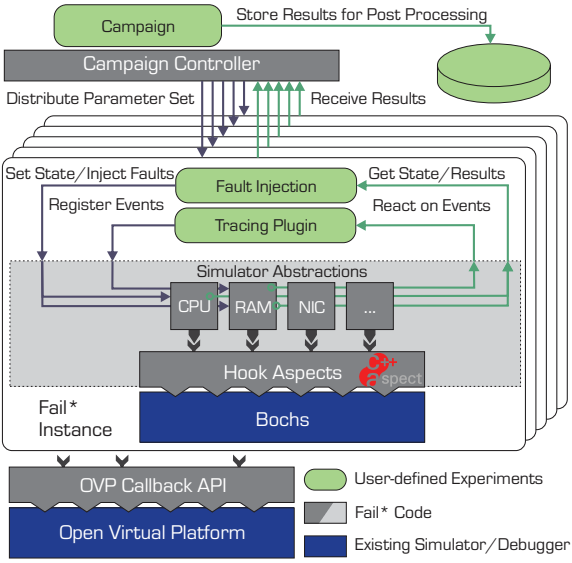


Figure 1: Architecture overview: The *Campaign Controller* distributes parameter sets from a user-defined *Campaign* throughout the FAIL* instances. Each single experiment ("Fault Injection") consumes a parameter set, and controls its target backend through a *Simulator Abstraction* layer. Actual target backends (simulators, but also real prototype hardware) can be exchanged by providing an interfacing module to this abstraction.

and can be assisted by, e.g., a memory access tracing plugin. Actual target backends (system simulators, or real prototype hardware in later development phases) can be exchanged by providing an interfacing module to this abstraction. The diagram shows a FAIL* instance interfacing with the popular x86 simulator Bochs [Law96] by means of Aspect-Oriented Programming, a technique we apply to retain a maintainable, loosely-coupled code base while still being able to gain deep access to the simulator's state; though, the details of this method are out of this article's scope and have partially been outlined in earlier work [SHK+11].

2.2 API Design

User-defined campaigns and FI experiments are implemented against a C++ API offering access to both target backend meta-information and current state. The interface is designed to abstract away machine-specific details such as the register set or events occurring during an experiment run. The FAIL* API currently provides abstractions for:

- **Machine registers:** Both meta information (e.g., number of registers, platform-independent naming of the program counter or stack pointer registers, bit widths and byte order, an iterator interface) and read/write state access is provided.

- **Memory:** Access to meta data (size, memory type) and state (read/write) is provided.
- **Events:** A set of system events an experiment or plugin may register for, e.g., a specific program address is reached, memory is being written to, or a trap has been generated.
- The **target system** as a whole: Mostly state access is provided, including backend state save/restore for deterministic repeatability of experiments, and a means to reset the system.

Each target backend may additionally introduce interfaces to target-specific state, e.g., a means to manipulate a network device; experiments utilizing these are naturally not portable anymore, unless an adequate abstraction is added to the generic API.

FI experiments usually follow a simple, sequential scheme:

As experiments are event-driven (e.g., wait for reaching the specified position in space/time to inject the fault) but need to retain a substantial amount of internal state between the sequential steps, we chose not to provide a register/callback API (that would force the experiment developers to explicitly carry state from one callback to the next) but an API with blocking calls that return to a sequentially written experiment flow upon event activation (see Sec. 3 for an example).

Additionally, we currently consider to introduce a classic register/callback API for experiments or companion plugins with little or no state to keep between incoming events; the aforementioned memory-access tracing plugin seems to be such a case and could possibly be formulated even more concisely with callbacks.

2.3 Tool Prototype

The current prototype implementation of FAIL* provides a target backend for the BOCHS x86 simulator (version 2.4.6) accompanied with all previously described backend abstractions (implemented in C++ and AspectC++ [SLU05], an Aspect-Oriented Programming extension to C++), with an alternative ARM backend (OVP, [Bai08]) currently under development. The campaign parameter set distribution utilizes the Google Protocol Buffer (PB) library for lightweight communication and efficient parallelization. Analogously all result sets are represented as PB messages simplifying post-processing, with the help of PB's versatile language support. As a proof-of-concept, a companion Memory Access Tracing plugin has been implemented.

```
1 // campaign: iterate over all backend machine registers
2 RegisterManager& rm = simulator.getRegisterManager();
3 for (fi::RegisterManager::iterator it = rm.begin();
4      it != rm.end(); ++it) {
5    Register *reg = &(*it);
6    // iterate over all bit positions within this register
7    for (int bitpos = 0; bitpos < reg->getWidth(); ++bitpos) {
8       // iterate over all instr. offsets in the target function
9       for (int instr_offset = 0; instr_offset < COVERAGE_NUMINSTR;
10           ++instr_offset) {
11          // encapsulate parameter set for a single experiment
12          FaultCoverageParam *p = new FaultCoverageParam;
13          d->msg.set_instr_offset(instr_offset);
14          d->msg.set_bitpos(bitpos);
15          d->msg.set_inject_register(reg->getId());
16
17          // enqueue the parameter set for retrieval by a client
18          fi::campaignmanager.addParam(d);
19       }
20    }
21 }
```

Listing 1: (Simplified) Fault coverage campaign implementation: The code excerpt shows the parameter set generation.

3 An Example Experiment

In the following we describe a straight-forward implementation of a fault-coverage campaign using the FAIL* API. The campaign implementation (Listing 1) uses the machine register abstraction to iterate over all registers, every single bit in each register, and all possible instruction offsets within a C function—the analysis subject—running in the target system. For each point in this parameter space, a parameter set is generated (lines **12–18**) and communicated to an available FAIL* instance, which executes the experiment shown in Listing 2.

The experiment is implemented in a concise, reusable and portable way, based on the FAIL* API. The first action is to request a parameter set for the current experiment from the Campaign Controller (line **2**), holding the aforementioned parameter sets in a job queue. Then we restore a system snapshot taken at the exact point when the function under evaluation is being entered (line **5**). This ensures each run starts under the exact same conditions.

The next steps involve enqueuing a `BPEvent` (BP abbreviates *breakpoint*) that normally fires when a specific address has been reached. In this case, though, the address is not really "specific": `ev_fi_instr` (line **7**) is configured to fire at the wildcard address `ANY_ADDR`, but not on its first occurrence; the optional second parameter (shared by all event types) introduces an event *count*, letting the event only fire after it occurred *count* times, instead

```
// retrieve parameter set from campaign
jc.getParam(par);
// restore previously saved simulator state:
// we're now at the entry of the analyzed func.
simulator.restore("sav/p_entry.sav");
// breakpoint n instructions (def. in parameter set) in the future
BPEvent ev_fi_instr(ANY_ADDR, par.instr_offset());
addEventAndWait(&ev_fi_instr);

// FI: single bit-flip in register specified in parameter set
Register r = simulator.getRegisterManager().
             getRegister(par.inject_register());
r.setData(r.getData() ^ (1 << par.bitpos()));

// Aftermath: traps, timeout, or normal exit
TrapEvent ev_trap(ANY_TRAP);
addEvent(&ev_trap);
BPEvent ev_timeout(ANY_ADDR, 1000);
addEvent(&ev_timeout);
BPEvent ev_func_end(ADDR_FUNC_END);
addEvent(&ev_timeout);
// wait for function exit, trap or timeout
BaseEvent *ev = waitAny();
// store experiment result in parameter set object ...
if (ev == &id_func_end) {
   int result = simulator.abi_func_retval();
   par.set_resulttype(LOG_NORMAL);
   par.set_result(result);
} else if (ev == &ev_trap) {
   par.set_resulttype(LOG_TRAP);
} else if (ev == &ev_timeout) {
   par.set_resulttype(LOG_TIMEOUT);
}
// ... and communicate it back to the campaign controller
jc.sendResult(par);
```

Listing 2: (Simplified) Fault coverage experiment implementation: The code excerpt shows the FI part, parametrized by the register, the bit to flip, and the code offset for injection.

of firing at its first occurrence. In effect, this allows us to count down instructions until the point within the evaluated function we want to inject the register bit-flip fault at (the "Instr" element in the parameter set). The blocking `addEventAndWait()` call (line **8**) combines registering as a listener for this event, and waiting for it to fire.

Once we reach line **11**, the event must have fired, and we go for the fault injection. Through the register abstraction and the parameter set received from the campaign, we grab the register we are supposed to inject the bit-flip into (line **11**) and modify its current state in line **13**.

Having injected the fault, we want to observe the outcome of this run: failure-free returning from the function (with a correct or faulty return value—this will be determined offline by evaluating the log files), a hardware trap (MMU violation, division by zero, ...), or a timeout. Lines **16–21** register three more events for catching these cases (without already resuming the simulator: `addEvent()` is non-blocking). The remaining code waits for one of the three events to fire (line **23**: `waitAny()` continues the simulator execution and blocks), and accordingly reports the result back to the campaign controller (lines **25–33**) by storing it in the parameter set (which subsequently is being transmitted in line **34**). Note the abstraction for a target system's ABI convention to store return values (line **26**).

This consequent usage of target backend abstractions allows to carry out the experiment with another simulator or hardware backend, once an abstraction library has been provided for it. Companion plugins, such as the memory-access tracing plugin, allow for a more coarse-grained reuse.

4 Discussion

We believe FAIL* will achieve the claimed low-effort switching of target backends by its explicit modular design, separating campaign descriptions, experiment instances and the associated target backends. The aforementioned Aspect-Oriented Programming techniques will—at least in the case of the Bochs variant—alleviate the task of updating to newer backend versions: they allow us to reuse traditionally very tightly-coupled modules, such as, e.g., the implementation of FI in the data bus for memory reads. Other systems, such as for example OVP, might already provide distinct callback interfaces which can be utilized directly.

The experiment API (as outlined in Subsec. 2.2) and its underlying abstractions for target backend commonalities such as machine registers, memory, or system events was explicitly designed to foster experiment code reuse. The exemplary FI campaign shown in the previous section (Listings 1 and 2) illustrates this quite clearly: it could be reused with another target backend *without modification*, even if FAIL* would be configured for a platform with a completely different instruction set, a reversed byte-order, or another set of general- and special-purpose registers. We are confident to confirm this educated guess once more target backends are implemented.

5 Conclusions and Future Work

We presented a novel concept for a versatile fault-injection framework, aiming at supporting large-scale dependability evaluation and system analysis campaigns on various target-platform backends. Our FAIL* framework provides abstractions supporting portable experiment implementations, fostering code reuse, and reducing the familiarization efforts for new simulator or hardware backends.

Currently our framework implementation is at a relatively early stage, providing a complete interface layer for the Bochs simulator, with OVP interfacing currently in development. Our next steps include advancing the simulator abstraction API, utilizing the available tracing capabilities for conducting pre-injection analyses similar to [BVFK05], and implementing an interface to a real hardware platform to evaluate the flexibility of our infrastructure.

References

[AVFK01] Joakim Aidemark, Jonny Vinter, Peter Folkesson, and Johan Karlsson. GOOFI: Generic Object-Oriented Fault Injection Tool. In *Proceedings of the 31st IEEE/IFIP International Conference on Dependable Systems and Networks (DSN '01)*, pages 83–88, Los Alamitos, CA, USA, 2001. IEEE Computer Society Press.

[Bai08] Brian Bailey. System Level Virtual Prototyping becomes a reality with OVP donation from Imperas. *White Paper*, (June), 2008.

[Bor05] Shekhar Y. Borkar. Designing reliable systems from unreliable components: the challenges of transistor variability and degradation. *IEEE Micro*, 25(6):10–16, 2005.

[BP03] Alfredo Benso and Paolo Prinetto, editors. *Fault Injection Techniques and Tools for Embedded Systems Reliability Evaluation (Frontiers in Electronic Testing)*. Springer-Verlag, Boston, 1st edition, October 2003.

[BVFK05] Raul Barbosa, Jonny Vinter, Peter Folkesson, and Johan Karlsson. Assembly-Level Pre-injection Analysis for Improving Fault Injection Efficiency. In *Proceedings of the 5th European Dependable Computing Conference (EDCC 2005)*, volume 3463, page 246. Springer-Verlag, April 2005.

[DC07] Francis M. David and Roy H. Campbell. Building a Self-Healing Operating System. In *Proceedings of the 3rd IEEE International Symposium on Dependable, Autonomic and Secure Computing*, pages 3–10, Washington, DC, USA, 2007. IEEE Computer Society Press.

[DCCC08] Francis M. David, Ellick Chan, Jeffrey Carlyle, and Roy H. Campbell. Qinject: A virtual-machine based fault injection framework. In *Proceedings of the 13th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '08)*, 2008. (Poster Presentation).

[DYdS+10] Marc Duranton, Sami Yehia, Bjorn de Sutter, Koen de Bosschere, Albert Cohen, Babak Falsafi, Georgi Gaydadjiev, Manolis Katevenis, Jonas Maebe, Harm Munk, Nacho Navarro, Alex Ramirez, Olivier Temam, and Mateo Valero. The HiPEAC Vision. Technical report, Network of Excellence on High Performance and Embedded Architecture and Compilation, 2010.

[FGAF06] André Fidalgo, Manuel Gericota, Gustavo Alves, and José Ferreira. Using NEXUS compliant debuggers for real time fault injection on microprocessors. In *Proceedings of the 19th Annual Symposium on Integrated Circuits and Systems Design*, pages 214–219. ACM Press, 2006.

[Law96] Kevin P. Lawton. Bochs: A Portable PC Emulator for Unix/X. *Linux Journal*, September 1996.

[NX06] Vijaykrishnan Narayanan and Yuan Xie. Reliability concerns in embedded system designs. *IEEE Computer*, 39(1):118–120, 2006.

[SBK10] Daniel Skarin, Raul Barbosa, and Johan Karlsson. GOOFI-2: A tool for experimental dependability assessment. In *Proceedings of the 40th IEEE/IFIP International Conference on Dependable Systems and Networks (DSN '10)*, pages 557–562, Los Alamitos, CA, USA, July 2010. IEEE Computer Society Press.

[SHK+11] Horst Schirmeier, Martin Hoffmann, Rüdiger Kapitza, Daniel Lohmann, and Olaf Spinczyk. Revisiting Fault-Injection Experiment-Platform Architectures. In *Proceedings of the 17th IEEE Pacific Rim International Symposium on Dependable Computing (PRDC '11)*, Pasadena, USA, December 2011. IEEE Computer Society Press. Fast abstract.

[SLU05] Olaf Spinczyk, Daniel Lohmann, and Matthias Urban. Advances in AOP with AspectC++. In Hamido Fujita and Mohamed Mejri, editors, *New Trends in Software Methodologies, Tools and Techniques (SoMeT '05)*, number 129 in Frontiers in Artificial Intelligence and Applications, pages 33–53, Tokyo, Japan, September 2005. IOS Press.

[SPS09] Matthias Sand, Stefan Potyra, and Volkmar Sieh. Deterministic high-speed simulation of complex systems including fault-injection. In *Proceedings of the 39th IEEE/IFIP International Conference on Dependable Systems and Networks (DSN '09)*, pages 211–216. IEEE Computer Society Press, July 2009.

[ZAV04] Haissam Ziade, Rafic A. Ayoubi, and Raoul Velazco. A Survey on Fault Injection Techniques. *The International Arab Journal of Information Technology*, 1(2):171–186, July 2004.

Correction of Faulty Signal Transmission for Resilient Designs of Signed-Digit Arithmetic

David Neuhäuser and Eberhard Zehendner

Institute of Computer Science
Friedrich Schiller University
D-07737 Jena, Germany
{david.neuhaeuser,nez}@uni-jena.de

Abstract: When arithmetic components are parallelized, fault-prone interconnections can tamper results significantly. Advances in feature size shrinking lead to a steady increase of errors caused by faulty transmission. We suggest to employ resilient data encoding schemes to offset these negative effects. Focusing on parallel signed-digit based arithmetic, frequently used in high-speed systems, we found that a suitable data encoding can reduce error rates by about 25% when using 2-bit encoding and about 62% when using 3-bit encoding. Data encoding should be driven by symbol occurrence probabilities. We develop a methodology to obtain these probabilities, show example fault-tolerant encodings, and discuss the impact on communicating parallel arithmetic circuits in example error scenarios.

1 Introduction

In times of billion-transistor processors being commercially available and transistors being processed in 22 nanometer CMOS process [ITR11] and beyond [Iwa09], it becomes more and more difficult to design fault tolerant [NSF01, RSKW07] or mixed critical systems [PMN+09]. More complex circuits require increased inter- and intra-circuit connections which become increasingly fault-prone.

Focusing on fast, parallelized, signed-digit based arithmetic, used extensively for instance in CORDIC arithmetic processors, we propose data encodings that can significantly reduce transmission error rates. When bit-level transmission is fault-prone, the received symbols might be faulty. To minimize the symbolic level error rate, we apply redundancy at bit-level by encoding more often used symbols by several bit combinations, using three steps:

- Obtain bit-level error rates.
- Obtain symbol occurrence probabilities.
- Given $R \in \mathbb{N}$, map bit patterns with R bits to symbols so that:
  - every symbol is represented by at least one bit pattern,
  - the overall symbol error rate is minimized.

The reduction of the symbol error rate depends on the available R bits per symbol. We apply our methodology exemplarily to 2-bit and 3-bit encodings and provide an error rate optimal encoding.

Alternative approaches for fault tolerant CORDIC are using TMR. They either require increased hardware [WWY06] or increased latency when reusing hardware [KPS01]. Alternative approaches for signed-digit like using check symbols have been proposed [COP+06], which are less efficient in terms of latency, since every arithmetic operation has to be done multiple times to obtain error information.

In the following section we discuss possible communication error scenarios. In Section 3 we discuss the used signed-digit arithmetic and show our methodology to obtain digit probabilities for signed-digit encoded data. In Section 4 we propose an algorithm to find optimal fault tolerant data encoding, and give recommendations for error resilient encoding. Applying our methodology, we provide accurate data word probabilities for common signed-digit adder cell implementations in Section 5 and present error rates for different encoding schemes. We conclude in Section 6 and give an outlook to future work.

2 Bit-Level Error Rates

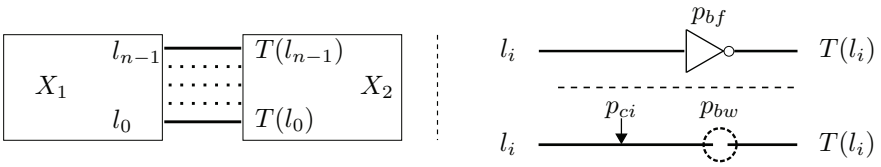


Figure 1: Left: Circuit X_1 communicates with circuit X_2 through signal lines s_0 to s_{n-1}. Right top: Simple error model of possible bit flip with probability p_{bf}. Right bottom: Complex error model with a broken wire (p_{bw}) and/or (radiation caused) electric charge insertion (p_{ci}).

Figure 1 shows two circuits exchanging data by signal lines 0 through $n-1$. On each line, the signal is sent as $l_i \in \{0,1\}$ and received as $T(l_i) \in \{0,1\}$. The data received may differ from the data sent due to imperfect wiring [SOHH07, KPKJ07].

Simple error model In our simple error model, p_{bf} denotes the probability, that one bit is inverted. The possibility of a bit flip leads to

$$\begin{aligned} T(l_i) &= \begin{cases} l_i & \text{when no bit flip occurred,} \\ 1-l_i & \text{else.} \end{cases} \\ P(T(l_i)=l_i) &= 1-p_{bf} \end{aligned}$$

Inverting encoding schemes by flipping 0 and 1 does not change the achieved error rate.

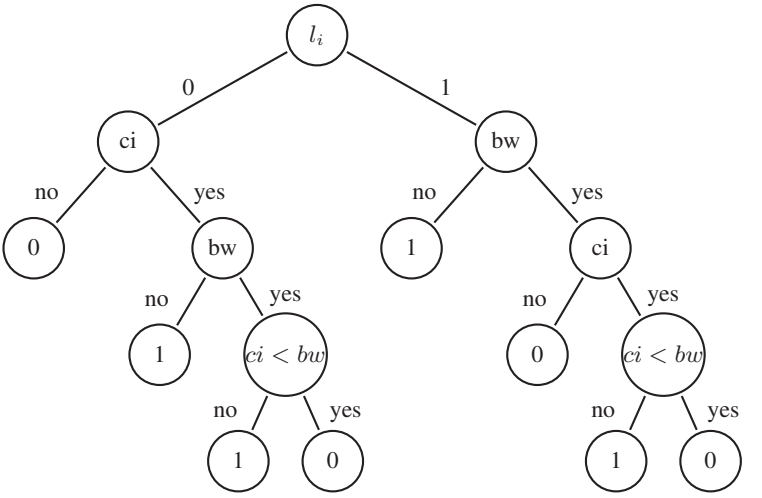


Figure 2: Complex error model decision graph. "ci" denotes current insertion, "bw" denotes broken wire, "$ci < bw$" denotes "ci" occurs close to X_1 and "bw" occurs closer to X_2. Leafs are $T(l_i)$.

Complex error model In a more complex error model, p_{bw} denotes the probability of a broken wire ($T(l_i) = 0$), p_{ci} denotes electric charge insertion ($T(l_i) = 1$), i.e. radiation caused.

$$P(T(0) = 0) = 1 - p_{ci} + p_{ci} \cdot p_{bw} \cdot p_{ci<bw} \quad (1)$$

$$P(T(1) = 1) = 1 - p_{bw} + p_{bw} \cdot p_{ci} \cdot (1 - p_{ci<bw}) \quad (2)$$

For the simple model holds $P(T(0) = 0) = P(T(1) = 1)$. The complex model differs to the simple model by the influence of p_{ci}, p_{bw}, and $p_{ci<bw}$ on $P(T(0) = 0)$ and $P(T(1) = 1)$ (see Equ. 1 and 2). Inverting encoding schemes by flipping 0 and 1 can change the resulting error rate.

3 Symbol Occurrence Probabilities

We use signed-digit arithmetic as an example to derive symbol occurrence probabilities. A special case of a signed-digit [Avi61] number system is a signed-binary number system, where each digit is limited to $\{-1, 0, 1\}$. In the following we focus on signed-binary number systems.

A signed-binary adder (SBA) calculates $S_{sb} = A_{sb} + B_{sb}$, where S_{sb},A_{sb}, and B_{sb} are signed binary numbers. This operation is decomposed into digit operations by signed-binary adder cells (SBAC). Figure 3 shows this decomposition. One operation at digit i calculates $s_i = a_i + b_i$. Since $a_i + b_i \in \{-1, 0, 1\} + \{-1, 0, 1\} = \{-2, -1, 0, 1, 2\}$, but $s_i \in \{-1, 0, 1\}$, we need some carry to propagate $\{-2, 2\}$ to the digit at $i+1$. Focusing on

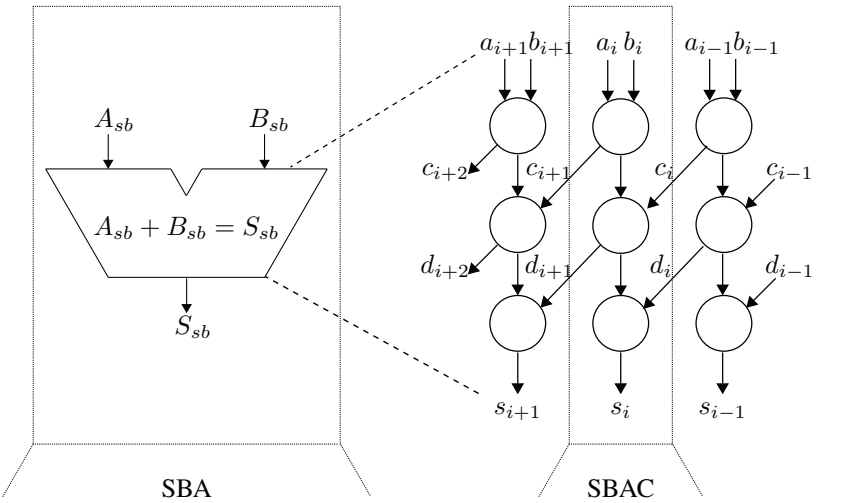


Figure 3: Signed-binary adder (SBA) consisting of three level signed-binary adder cells (SBAC) shown at numerical level, see [CR78, S.111].

a 3-level design as in Chow and Robertson [CR78], we describe a SBAC through atomic operations that are in accordance to Equation 8.

$$e_i = a_i + b_i \tag{3}$$

$$c_{i+1}(t, a_i, b_i) = \begin{cases} 0 & \text{when } e_i > 0, \\ -1 & \text{when } e_i < 0, \\ \gamma(t, a_i, b_i) & \text{when } e_i = 0. \end{cases} \tag{4}$$

$$f_i = e_i - 2 \cdot c_{i+1}(t, a_i, b_i) \tag{5}$$

$$g_i = f_i + c_i \tag{6}$$

$$d_{i+1} = \begin{cases} 0 & \text{when } g_i \leq 0, \\ +1 & \text{when } g_i > 0. \end{cases} \tag{7}$$

$$h_i = g_i - 2 \cdot d_{i+1} \tag{8}$$

$$s_i = h_i + d_i = a_i + b_i + c_i + d_i - 2 \cdot c_{i+1} - 2 \cdot d_{i+1} \tag{9}$$

The calculation of c_{i+1} must be independent from c_i and d_i, the calculation of d_{i+1} must be independent from d_i. The resulting carry chain is locally constraint, the calculation of any s_i depends only on a_i, b_i, a_{i-1}, b_{i-1}, a_{i-2}, and b_{i-2} [Zeh92].

We construct a decision graph, see Figure 4, that shows all possible degrees of freedom when constructing a functionally correct SBAC that is constrained by the formal model described above. In Figure 4, all impossible choices of c_{i+1} and d_{i+1} have already been removed. We illustrated the choice of $c_{i+1} = -1$ by dotted arrows and of $c_{i+1} = 0$ by dashed arrows. This choice also fixes the decision of d_{i+1}.

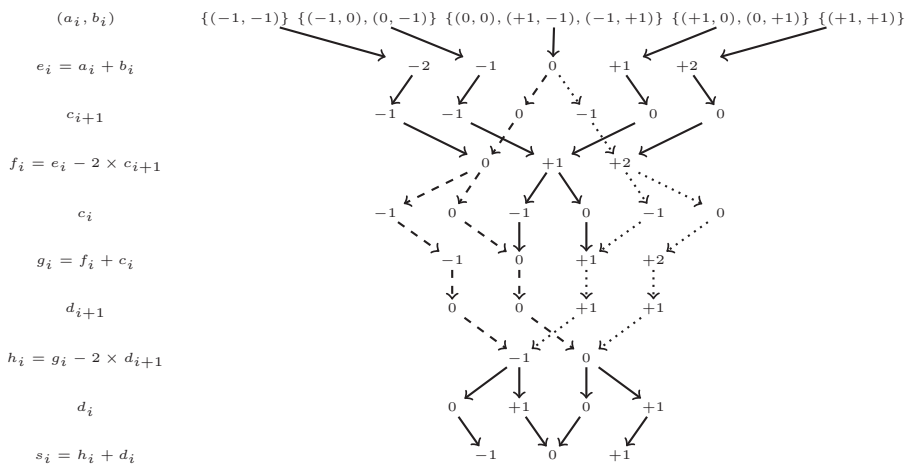


Figure 4: Signed-binary adder cell decision graph. For $a_i + b_i = 0$, the dashed graph denotes a choice of $c_{i+1} = 0$, the dotted graph a choice of $c_{i+1} = -1$. For $a_i + b_i = 0$ it is obvious, that d_{i+1} depends on the choice of c_{i+1} but not on c_i. Furthermore, for $a_i + b_i = 0$,s_i does not depend on the choice of c_{i+1}, but s_{i+1} does depend on the choice of c_{i+1} through d_{i+1}.

Our SBAC model offers $2^3 = 8$ different signed-binary adder cells at the numerical level. Let t be the *type id* of the design choice, $0 \leq t < 2^3$. All possible design choices are shown in Table 1. Note that the formula for calculating c_{i+1} depends on the input digits (a_i, b_i) and the chosen design parameter t. Let $SBAC_t$ be the design using choice t to calculate c_{i+1}.

Assigning probability information to the symbols in Equations 3 through 7 we are able to calculate the digit probabilities. As an example we give the calculation of c_{i+1}:

$$\begin{aligned} P(c_{i+1} = -1) &= P(e_i = -2) + P(e_i = -1) + P(\gamma(t, a_i, b_i) = -1) \\ P(c_{i+1} = 0) &= P(e_i = +1) + P(e_i = +2) + P(\gamma(t, a_i, b_i) = 0) \end{aligned}$$

$P(\gamma(t, a_i, b_i) = 0)$ and $P(\gamma(t, a_i, b_i) = -1)$ are calculated in accordance to Table 1 as

$$\begin{aligned} P(\gamma(t, a_i, b_i) = 0) &= P(a_i = \ \ 0) \cdot P(b_i = \ \ 0) \cdot P(t \in \{0, 1, 2, 3\}) + \\ &\quad P(a_i = +1) \cdot P(b_i = -1) \cdot P(t \in \{0, 1, 4, 5\}) + \\ &\quad P(a_i = -1) \cdot P(b_i = +1) \cdot P(t \in \{0, 2, 4, 6\}) \\ P(\gamma(t, a_i, b_i) = -1) &= P(a_i = \ \ 0) \cdot P(b_i = \ \ 0) \cdot P(t \in \{4, 5, 6, 7\}) + \\ &\quad P(a_i = +1) \cdot P(b_i = -1) \cdot P(t \in \{2, 3, 6, 7\}) + \\ &\quad P(a_i = -1) \cdot P(b_i = +1) \cdot P(t \in \{1, 3, 5, 6\}) \end{aligned}$$

For any given input digit probabilities, we are now able to calculate the corresponding output digit probabilities.

	(a_i, b_i)		
	$(0,0)$	$(+1,-1)$	$(-1,+1)$
t	$\gamma(t, a_i, b_i)$		
0	0	0	0
1	0	0	-1
2	0	-1	0
3	0	-1	-1
4	-1	0	0
5	-1	0	-1
6	-1	-1	0
7	-1	-1	-1

Table 1: Meaning of parameter t in description of SBAC.

4 Mapping Bit Patterns to Symbols

When encoding signed-binary digit symbols $\{-1, 0, 1\}$ with bit strings of a fixed length, we either can encode every symbol with exactly one bit pattern (non-redundant encoding scheme), or we may opt to encode one or more symbols with several bit patterns (redundant encoding scheme). For any symbol, we choose one distinct encoding as the **base encoding**. When using a redundant encoding scheme, all other encodings are called **secondary encodings**. In our approach, each symbol should always be provided as its base encoding. However, when modified to some corresponding secondary encoding by faulty transmission, the latter should also be regarded as correct. We get the following constraints

- any arithmetic operation produces base encoded symbols
- any transmission can corrupt base encoded symbols, producing:
  - correctly base or correctly secondary encoded symbols
  - wrongly base or wrongly secondary encoded symbols
- any arithmetic operation accepts base and secondary encoded symbols

Given the bit-level error rates, the symbol occurrence probabilities and the number R of bits available per symbol, we can map bit patterns to symbols in way that every symbol is represented by at least one bit pattern and overall symbol error rate is minimized. We propose the following brute force algorithm to find an encoding scheme with minimal error rate:

```
1. start with one symbol
2. define a base encoding for this symbol
   out of all remaining encodings
3. if it is the last symbol, use all remaining encodings
   as secondary encodings
4. else: for any subset of all now remaining encodings
   - all encodings of this subset are secondary encodings
```

```
      to the base encoding
    - recurse into 2 with an unencoded symbol
4. calculate overall error rate
5. track minimal error rate and according encoding scheme
```

5 Results

At first we discuss the probabilities of signed-digit symbols when applying SBAC operations repeatedly in Subsection 5.1. In Subsection 5.2 we proposed example encodings obtained by the algorithm of Section 4 and discuss the resulting reduced error rates.

5.1 Signed-Digit Symbol Occurrence Probabilities

For any trivial digit probability, where one symbol out of $\{-1, 0, 1\}$ has the probability of 1, and the others have 0, the probabilities of the output symbols are either 0 or 1.

If any other, non trivial data probability is applied to a_i, b_i, and the probability distribution of s_i is looped back to the $SBAC_t$ inputs a_i, b_i, the probabilities converge, see example Figure 5, where $t = 5$ and initial $P(a_i = 0) = P(b_i = 0) = 0.1$ and $P(a_i = 1) = P(b_i = 1) = 0.9$. Since the calculation of c_{i+1} (and indirectly d_{i+1}) depends on t, the converg also depends on t, see hollow symbols in Figure 6.

A sample application for signed-digit arithmetic could be a CORDIC based algorithm. CORDIC [Vol59] transforms initial data iteratively with predefined coefficients. Let A_0 be the initial N-bit data and B_i the predefined coefficients:

$$A_{i+1} = f_{cordic}(A_i, B_i) \text{ , with } 0 \leq i \leq N - 1 \tag{10}$$

f_{cordic} is the CORDIC function for processing A_i and B_i by an adder/subtracter and

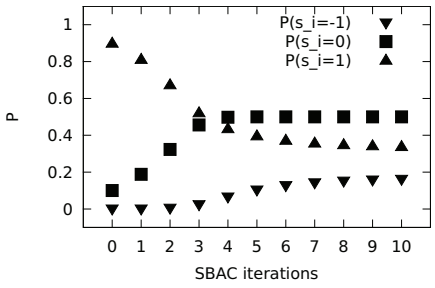


Figure 5: SBAC adder operation data probability for $t = 5$ and initial $P(a_i = 0) = P(b_i = 0) = 0.1$ and $P(a_i = 1) = P(b_i = 1) = 0.9$.

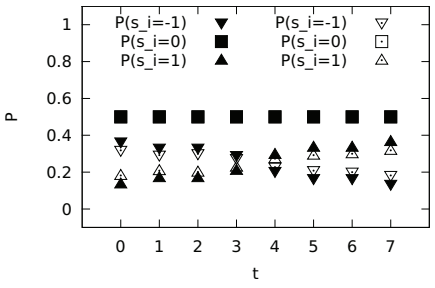


Figure 6: SBAC adder/subtracter data probability depending on type t. Solid symbols represent $+$ operation, hollow symbols represent a probability of 50% for $+/-$ alternation.

shifter. A_N is the final result, A_0 the input data to be processed. To simulate the impact of subtraction, we assume a probability of 50%, that input for b_i has opposite signs. The solid symbols in Figure 6 correspond to this more realistic use case.

5.2 Example Encodings and According Symbol Error Rates

Table 2 (3) shows a 2-bit (3-bit) redundant encoding of signed-binary digit symbols with a minimal error rate when using a SBAC type $t = 7$, non trivial initial digit symbol occurrence, and a probability of $+/-$ alternation of 50%.

symbol	0	-1	1
encoding	**00** 10	**11**	**01**
error prob.	p_{bf}	$2 \cdot p_{bf} - p_{bf}^2$	$2 \cdot p_{bf} - p_{bf}^2$

Table 2: Example encoding scheme for $R = 2$. Base encodings are marked bold.

symbol	0				-1			1
encoding	**000**	001	010	100	**111**	101	110	**011**

Table 3: Example encoding scheme for $R = 3$. Base encodings are marked bold.

The error ratio of any redundant encoding scheme to the non-redundant 2-bit encoding can be calculated by

$$\text{error ratio} \quad = \quad \frac{\text{error rate redundant encoding}}{\text{error rate of 2-bit non-redundant encoding}} \tag{11}$$

For the two bit redundant encoding fo Table 2 we investigate the digit error depending on the bit flip error and the encoding, as shown in Figure 7. The error ratio is 75% for small p_{bf} and increases to expected 100% for $p_{bf} = 1$. This means by using error correction and redundant encoding for digit 0 instead of no error correction for any digit, when using SBAC type $t = 7$ in a CORDIC like arithmetic with switching sign possibility of one operand of 50%, error rate drops by 25%.

The three bit redundant encoding of Table 3 reduces the error ratio further to 38%, as can be seen in Figure 8. This means by using error correction and redundant encoding for digit 0 instead of no error correction for any digit, when using SBAC type $t = 7$ in a CORDIC like arithmetic with switching sign possibility of one operand of 50%, error rate drops by 62%.

Comparing the two bit redundant and three bit redundant encoding, the error ratio is

$$\text{redundant error ratio} \quad = \quad \frac{\text{error ratio of 3-bit red. encoding}}{\text{error ratio of 2-bit red. encoding}} = \frac{0.38}{0.75} \approx 51\% \tag{12}$$

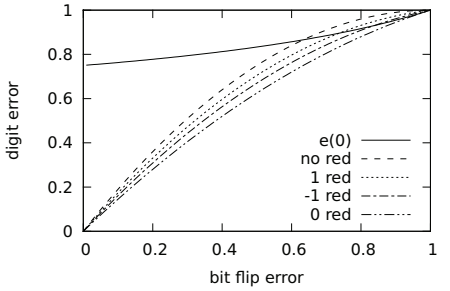


Figure 7: SBAC adder/subtracter digit error probability for $t = 7$ and non trivial initial digit occurrence. Probability of $+/-$ alternation is 50%. "no red" denotes no redundant encoding, "-1 (0,1) red" denotes redundant encoding and error correction for -1 (0,1), "$e(0)$" denotes the error ratio of redundant encoding and error correction of digit "0", see equation 11.

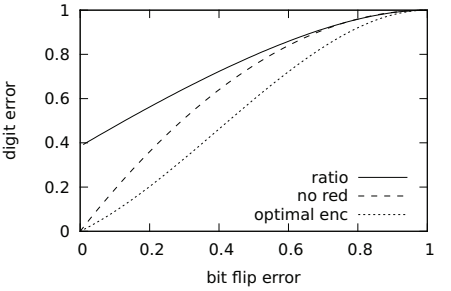


Figure 8: SBAC adder/subtracter digit error probability for $t = 7$ and non trivial initial digit occurrence. Probability of $+/-$ alternation is 50%. "no red" denotes a 2-bit non redundant encoding (enc 1), "optimal enc" denotes 3-bit optimal redundant encoding and error correction (enc2, see Table 3), "ratio" denotes the error ratio enc2 normed to enc1, see equation 11.

Using redundant encoding, increasing the minimal required encoding bits (2 bits) by 50% (3 bits) reduces error rate by $\approx 49\%$.

6 Conclusion and future work

When reliable signed-binary adder cells are connected to each other or to further reliable system components via highly unreliable switching networks,the knowledge of symbol occurrence probabilities offers a chance to use a data encoding scheme that provides some implicit fault tolerance. We have shown a methodology to gain data flow probability information for signed-binary based arithmetic operations and have proposed a data encoding scheme that provides advanced fault tolerance properties.

To validate these results, we intend to implement a complete arithmetic processor based on the chosen signed-binary adder structure and run some benchmarks on it. For 2-bit encodings, alternative signed-binary adder structures have been described, that might be worth further investigation with our methodology; cf. in particular [GHM89], [Zeh92], and the references in the latter. We will also try to improve the run-time complexity of our exploration algorithms to gain predictions for encodings with considerably higher redundancy.

References

[Avi61] A. A. Avižienis. Signed-Digit Number Representations for Fast Parallel Arithmetic. *IRE Transactions on Electronic Computers*, 10(3):389–400, Sep. 1961.

[COP+06] G. C. Cardarilli, M. Ottavi, S. Pontarelli, M. Re, and A. Salsano. Localization of Faults in Radix-n Signed Digit Adders. In *Proceedings of the 12th IEEE International On-Line Testing Symposium*, IOLTS, pages 178–180, Washington, DC, USA, 10–12 Jul 2006. IEEE Computer Society.

[CR78] C. Y. Chow and J. E. Robertson. Logical Design of a Redundant Binary Adder. *Proc. 4th Symposium on Computer Arithmetic*, pages 109–115, 1978.

[GHM89] A. Guyot, Y. Herreros, and J.M. Muller. JANUS, an On-line Multiplier/divider for manipulating large numbers. *Proceedings of the 9th Symposium on Computer Arithmetic, Santa Monica*, pages 106–111, 1989.

[ITR11] ITRS. *International Technology Roadmap for Semiconductors. 2011 Edition. Emerging Research Devices*. http://www.itrs.net/links/2011itrs/2011Chapters/2011ERD.doc, 2011.

[Iwa09] H. Iwai. Roadmap for 22 nm and beyond (Invited Paper). *Microelectronic Engineering*, 86(7–9):1520 – 1528, 2009.

[KPKJ07] A. Kumar, L.-S. Peh, P. Kundu, and N. K. Jha. Express virtual channels: towards the ideal interconnection fabric. In *Proceedings of the 34th annual international symposium on Computer architecture*, ISCA '07, pages 150–161, New York, NY, USA, 2007. ACM.

[KPS01] J.-H. Kwak, V. Piuri, and E. E. Swartzlander, Jr. Time-shared TMR for fault-tolerant CORDIC processors. In *Proc. ICASSP '01*, volume 2 of *IEEE International Conference on Acoustics, Speech, and Signal Processing*, pages 1241–1244, 7–11 May 2001.

[NSF01] K. Nikolic, A. Sadek, and M. Forshaw. Architectures for Reliable Computing with Unreliable Nanodevices. In *Proc. 1st IEEE Conference on Nanotechnology*, pages 254–259, 2001.

[PMN+09] R. Pellizzoni, P. Meredith, M. Nam, M. Sun, M. Caccamo, and L. Sha. Handling mixed-criticality in SoC-based real-time embedded systems. In *Proceedings of the seventh ACM international conference on Embedded software*, EMSOFT '09, pages 235–244, New York, NY, USA, 2009. ACM.

[RSKW07] W. Robinett, G. S. Snider, P. J. Kuekes, and R. S. Williams. Computing with a trillion crummy components. *Commun. ACM*, 50(9):35–39, Sep 2007.

[SOHH07] M. Stahl-Offergeld, H.-P. Hohe, and M. Hackner. Spinning current offset in vertical Hall sensors caused by imperfect wiring. In *Proc. 13th International Sensor Conference*, volume 2 of *SENSOR 2007*, pages 211–216, 22–24 May 2007.

[Vol59] J. E. Volder. The CORDIC Trigonometric Computing Technique. *Ieee Transactions On Electronic Computers*, EC-8(3):330–334, 1959.

[WWY06] S. Wang, Z. Wen, and L. Yu. High-performance fault-tolerant CORDIC processor for space applications. In *Proc. ISSCAA '06*, volume 1 of *1st International Symposium on Systems and Control in Aerospace and Astronautics*, page 4 pp., 19–21 Jan 2006.

[Zeh92] E. Zehendner. Reguläre parallele Addierer für redundante binäre Zahlsysteme. Technical report, Report 255, Institut für Mathematik der Universität Augsburg, Juni 1992.

Fault Tolerant and Adaptive Path Planning in Crowded Environments for Mobile Robots Based on Hazard Estimation via Health Signals

Raphael Maas, Erik Maehle

Institute of Computer Engineering
University of Lübeck
Ratzeburger Allee 160
23562 Lübeck
{maas, maehle}@iti.uni-luebeck.de

Abstract: Mobile robots are complex systems that tend to become even more complex. Since these systems show often graceful degradation in case of hardware faults the applied control strategies should be taken into account. The overall fitness of a robot can be condensed into health signals. Depending on this signal the control of the robot can be adjusted to counteract reduced sensing or acting capabilities, by following the design principles of organic computing. This work introduces techniques to estimate possible hazards that are introduced by the presence of obstacles in the close proximity of a damaged robot and might threaten its task.

1 Introduction

Since robotic systems have increased their complexity over the last years, it is desirable to equip these systems with fault tolerance as well as adaptive control strategies to guarantee a high reliability and dependability even in unforeseen situations.

In the context of path planning, factors such as decreasing energy resources or failing hardware can lead to a variety of new constraints. A robot with low energy resources should move along a very energy-efficient path, while a defective robot with decreased maneuverability should avoid paths that require many course changes.

An example for the effects of hardware failures on the maneuverability of a robot is given in [ALM08]. It could be observed that the compensation of the leg loss introduces a drift to the side of the defective leg during the movement.

Similar effects can be observed on wheeled robots that feature omni-drives [GWT+12]. A four wheeled omni-drive may compensate the defect of one wheel, but loses propulsive power and agility.

The planning of paths can be achieved with a variety of different algorithms like trajectory-based [JC89], graph-based (A*, HPA*) [BMS04], artificial potential fields [Ark98] or wave front based approaches [NI09, NN09].

A path planning approach that uses stereo vision to derive a difficulty measure for the surrounding terrain based on its height can be found in [CH09].

In a previous work, the effect of decreasing capabilities of the moving unit has been considered in [MM11], where health signals were used to influence the path planning to avoid regions that have become too demanding for the remaining capabilities by placing virtual obstacles into these regions. An alternative setup was the adaptation to energy constraints during paths planning.

In addition to these problems and approaches, some faults may also influence the sensing capabilities of the moving unit and must be taken into account. For example, if the sensors of a robot are damaged the path planning should avoid regions with a lot of obstacles and should choose free/plain regions instead. This paper presents algorithms that approach these problems on a generic level.

This paper is organized as follows: First, the underlying architecture of the adaptive and fault tolerant path planning is introduced in section 2. Then the path planning and fault tolerance methods are presented in section 3. Section 4 describes the environment of the experimental setup and sections 5 and 6 conclude the paper with the results and an outlook.

2 Organic Robot Control Architecture (ORCA)

ORCA [ALM08, MBG+11] is a hard- and/or software approach to design systems that keep the robot in a safe and optimal state during its operation, even in the presence of faults. In general, an ORCA controlled system monitors its current state and counteracts unhealthy changes in the overall system state in the best still possible way, without an explicit fault model. To rate the performance of a system health signals are used as a measure of the system's fitness.

The following sections will give a more detailed overview over the general design of ORCA and its implementation in this work.

2.1 The Design of ORCA

Basically, there are two types of functional units within ORCA. The first one is the Basic Control Unit (BCU) and the second one is the Organic Control Unit (OCU). These units may be connected, grouped and hierarchically layered to generate the overall function of the system.

On the lowest level a BCU encapsulates a basic function of the system. A more complex functionality can be achieved via the combination of several fundamental BCUs.

A system consisting of layered and interconnected BCUs is already capable of performing an arbitrary task. The OCUs are not needed in the design, as long as no fault tolerance is desired.

The OCUs are introduced into the design of the system to ensure the flawless operation in case of faults. These units observe attached BCUs and may alter parameters within them to counteract undesired system states that are considered as faults. However, if the system runs smooth the OCUs adhere to passive monitoring.

ORCA can be seen as a distributed variant of the observer/controller architecture presented in [RMB+06].

As mentioned above the OCUs monitor the BCUs and counteract undesired system states. Therefore, the OCU has to rely on a measure that indicates the fitness of a BCU. In the context of ORCA such a characteristic value is called health signal [LJA+07] and is generated by the unit that is under observation.

2.2 Implementing ORCA

In context of this work the principles of ORCA have been implemented as a software framework. For this approach the BCUs and OCUs are defined as (typed) interfaces. The general layout can be seen in Figure 1. Each BCU can have an OCU and vice versa. BCUs and OCUs can also have superordinated or subordinated variants on different hierarchical layers. A unit on a higher or lower layer is addressed via the *Parent* or *Child* property. In addition, both interfaces are inherited by typed variants of their corresponding interfaces. These typed interfaces offer a method to parameterize a class of the given type. (This concept will be described with the help of an example in the next paragraphs.) Finally, as defined in the previous section, the BCU has an instance of a health signal.

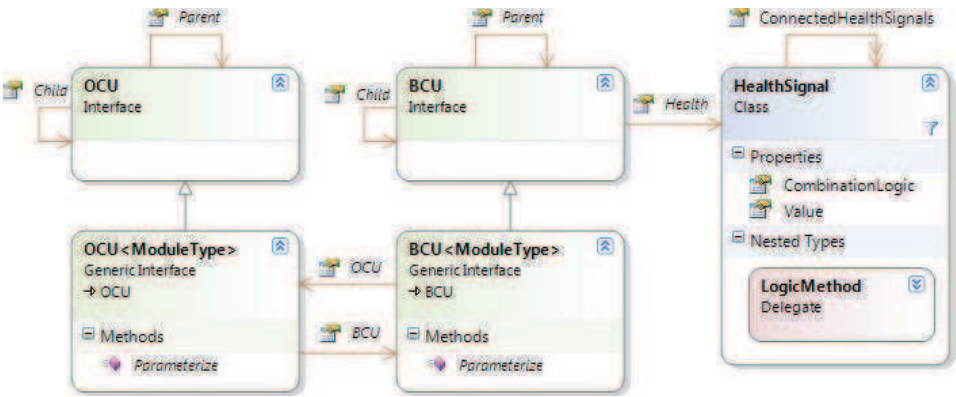


Figure 1: The class diagram of the chosen ORCA approach.

In case of adaptive path planning the planner BCU is defined as a new class that implements the BCU interface of its own type. Thereby the planner becomes a BCU of the type ”planner”. This guarantees that the planner can only be connected to OCUs, which are appropriate to observe a planner, because a typed BCU can only be connected to OCUs of

the same type. Therefore a planner OCU is capable of observing a planner BCU, where an OCU of the type engine for example is not, because of the different types.

As can be seen in Figure 1 each BCU and OCU must implement a method called *Parameterize*. This method expects a typed BCU of the same type as the implementing class and returns a modified version of this BCU. In the context of the planner it would accept an instance of the planner class, which is a BCU and return a modified version of this instance of a planner.

The *Parameterize* method of a BCU is only active, if no OCU is attached. Otherwise, it will hand over all control to the *Parameterize* method of the OCU. The *HealthSignal* class allows the hierarchical organization of health signals by offering the possibility to connect one health signal to other health signals that are vital to the functionality of its associated BCU. For example the health signal of a planner has to be connected to the health signals of the robot, because the planner cannot function normally if the robot is faulty. The internal combination logic of the different health signals can be adjusted via the definition of custom made, suitable *LogicMethod* delegate that fits the specific needs of a certain task.

3 Adaptive Path Planning

This section offers an overview over the algorithmic approach towards adaptive path planning. The path planning itself is carried out on a two dimensional (2D) grid of the environment, representing the terrain. The smallest unit within the 2D grid is a single cell that holds information like an abstract measure of the difficulty to cross the represented area, the distance to a goal or a measure that can be used to indicate possible hazards.

The use of abstract measures is on one hand a generalization, but offers on the other hand the possibility to represent different situations at once. Climbing a certain slope is a different task than walking over varying types of subsoil, but both can be described in terms of difficulty that they cause to the robot.

The path planner follows the ORCA design and is divided into BCU and OCU in a similar fashion as described in the examples in subsection 2.2. The BCU performs a shortest path planning between the robot and the goal, while avoiding obstacles. The associated OCU observes the health of the BCU, which is influenced by the health of the controlled robot. Basically, the OCU parameterizes the path planner to consider certain cells as blocked or hazardous, thereby counteracting paths that are not suitable for a robot in the current health state and might threaten the robot's mission.

3.1 Path Planning BCU

The shortest path planning is performed via wave front propagation. This algorithm can be best paraphrased by the effect that may be observed when an object plunges into water,

creating ripples that expand over the whole surface.

The algorithm that serves as basis for this work can be found in [MM11, MBG+11]. In addition to this approach each cell now features a value to indicate possible hazards within this cell. The value depends on the presence of obstacles in the near surrounding, thereby virtually increasing the area of an obstacle.

A cell at position (x,y) is denoted as c_{xy}. The hazard h of each cell c_{xy} is defined as

$$h_{c_{xy}} = \begin{cases} r - \lceil d - 1 \rceil & \text{for } d \leq r \\ 0 & \text{for } d > r \end{cases} \tag{1}$$

with $d = \sqrt{\triangle\ x^2 + \triangle\ y^2}$, r being the desired radius of effect and $\triangle\ x^2$ as well as $\triangle\ y^2$ being the distance of c_{xy} to the next obstacle.

3.2 Path Planning OCU

As mentioned in the previous subsection the path planner BCU generates a value to indicate areas that might become hazardous to a damaged robot. This information is not taken into account during normal operation.

As shown in [ALM08] the defect of a leg in a six-legged walking robot might introduce a drift to the side of the affected leg. Therefore, a path with many turns to different sides or obstacles on both sides of the path is mostly unfavorable. Hence, if the robot gets damaged the OCU is able to modify the path planning by increasing r in dependency of the current health signal and persuading the BCU interpret cells with $h > 0$ as obstacles.

This approach works well if the robot has enough space to avoid hazardous areas, but needs some additional interventions by the OCU, if a hazardous area has to be crossed in order to reach the goal. This is due to the fact that a goal cannot be reached if it is surrounded by hazards that are strictly avoided. Hence, if the OCU recognizes that the goal is surrounded and cannot be reached it modifies the path planning such that the robot will only traverse the safest cells in a hazardous area, while allowing cells with $t > h > 0$ to be traversed.

If c_{xy} holds no obstacles and is not part of a path it is termed as $\hat{c}_{xy}$. The adaption performed by the OCU, if no path is possible (while avoiding all hazards), searches path element by considering all cells that belong to the set

$$A_{pq} := \{\hat{c}_{xy} \mid\mid \hat{c}_{xy} - g \parallel_2 \leq \parallel c_{pq} - g \parallel_2\}, \tag{2}$$

with $\text{abs}(p - x) \leq 1$ and $\text{abs}(q - y) \leq 1$ for further analysis. In this case g denotes the position of the goal. As a next step the set with the lowest hazard is defined as

$$H_{\text{min}} := \{\alpha \in A_{pq} \mid \min(h_\alpha)\} \tag{3}$$

and finally the algorithm chooses the next cell of the path p_i by choosing an element from

the set H_{min} according to the definition

$$p_i := h \in H_{\text{min}} \tag{4}$$

with $\min(\| h - g \|_2)$.

The approach described above guarantees that the OCU parameterizes the planner in a way that it will ultimately find a static path through hazardous areas by adjusting the parameter t. The term static means that the path is planed once and the robot follows this path until it reaches the goal. The problem of a dynamic path planning, where the robot has to replan at each new cell, is the fact that the path planning prefers cells with a minimal hazard (see Equation (3)). This effect is illustrated in Figure 2. The path of the robot is indicated by arrows and hazardous cells are marked in gray. The left subfigure shows the initial path of the robot and the right subfigure shows the resulting path after a replan at the cell marked with an x. If the robot follows the new path the replan at the next cell will lead it back to the cell marked with an x and so on.

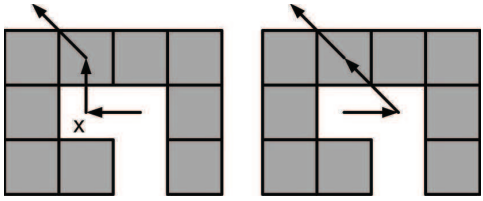


Figure 2: Due to the fact that the path planner avoids hazardous cells (shown in gray) as long as possible a robot might get stuck in a livelock during path planning, if the path is replanned at each cell. The initial path is shown in the left subfigure and the new path that occurs once the robot reaches the cell marked with an x is shown in the right subfigure.

Luckily the problem stated above can be easily overcome, because the shortest path to a given goal contains per definition no cycles. Therefore, each cell within any valid path can only be traversed once and the planner can mark traversed cells and avoid them during path planning, so that dynamic and static path planning result in the same path.

It has to be mentioned that one possible drawback of the approach given by Equation (2) to (4) is the fact that the overall quality of the resulting path is bound by the local maximum of h of all hazardous areas that have to be crossed. Therefore, the OCU has to increase t in order to mark hazardous areas as traversable. This fact in combination with certain spacial conditions can lead to paths that cross hazardous areas that could be avoided. Figure 3 shows this effect. The left subfigure depicts the path of a damaged robot. Hazardous areas are only crossed if no other choice is left. When the health signal drops lower the OCU has to adjust t even further, while hazardous areas are enlarged. This may lead to the effect that is shown in the right subfigure. The avoidance of hazardous areas is now bound by Equation (2), allowing only cells that do not increase the distance to the goal, while the OCU marks hazardous cells as traversable in order to reach the goal.

To overcome the problem stated above the robot can increase the quality of an obtained path by the iterative insertion of waypoints at the borders of hazardous areas that have to be

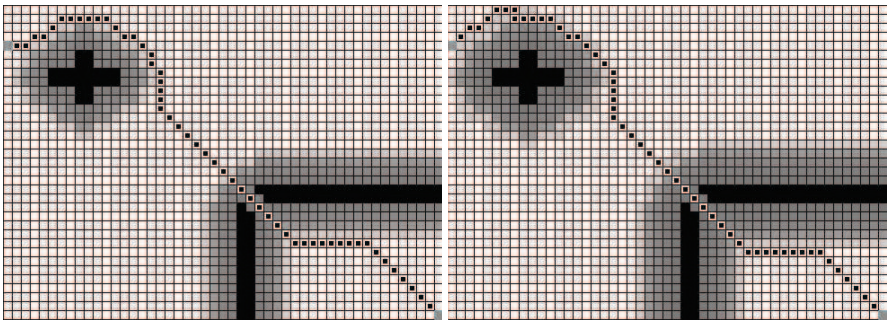


Figure 3: The left subfigure shows the path (dotted cells) of a damaged robot, while the right subfigure shows the path for an even further degraded health signal. A combination of the health state and certain spatial condition lead to the effect that the path planner chooses a path through hazardous areas that could have been avoided. Different levels of hazards are shown as shades of gray.

traversed. Figure 4 shows a segmented path with four waypoints. Basically the robot uses the path from Figure 3 (right) and inserts the two waypoints (cells marked with a cross) that are closest to the goal. The combination of two waypoints serves as a passage from a hazard free area to a hazardous area. Now the path is replaned to the waypoint within the hazardous area. Again two waypoints are added to allow the passage back to a hazard free area and a last replanning delivers the last part of the path that now avoids the hazardous cells that were traversed before.

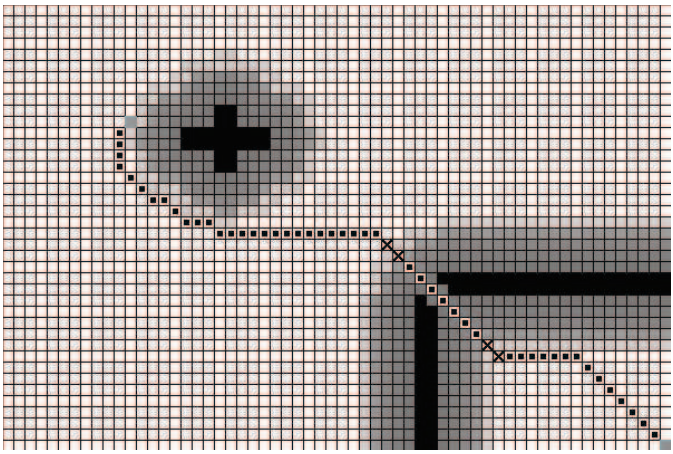


Figure 4: The figure shows a path that has been improved by the insertion of waypoints. Hazardous areas are only traversed if there is no other choice left. The path is depicted as dotted cells and the waypoints are shown as cells marked with a cross.

4 Simulation Environment

With respect to the suggestions presented in [ARS10] the experiments have been carried out in simulation to achieve a fast and reproducible evaluation of the algorithmic approach.

The implementation of ORCA described in subsection 2.2 is used in a custom made simulator written in C#. It allows the simulation of the approaches presented in [MM11] as well as this work. A simplified and adapted overview can be found in Figure 5. *GUI* and *Simulator* are decoupled, meaning that the *Simulator* is capable of running without the GUI to allow fast, automated simulations without user interaction. ORCA specific connections are depicted as dashed lines.

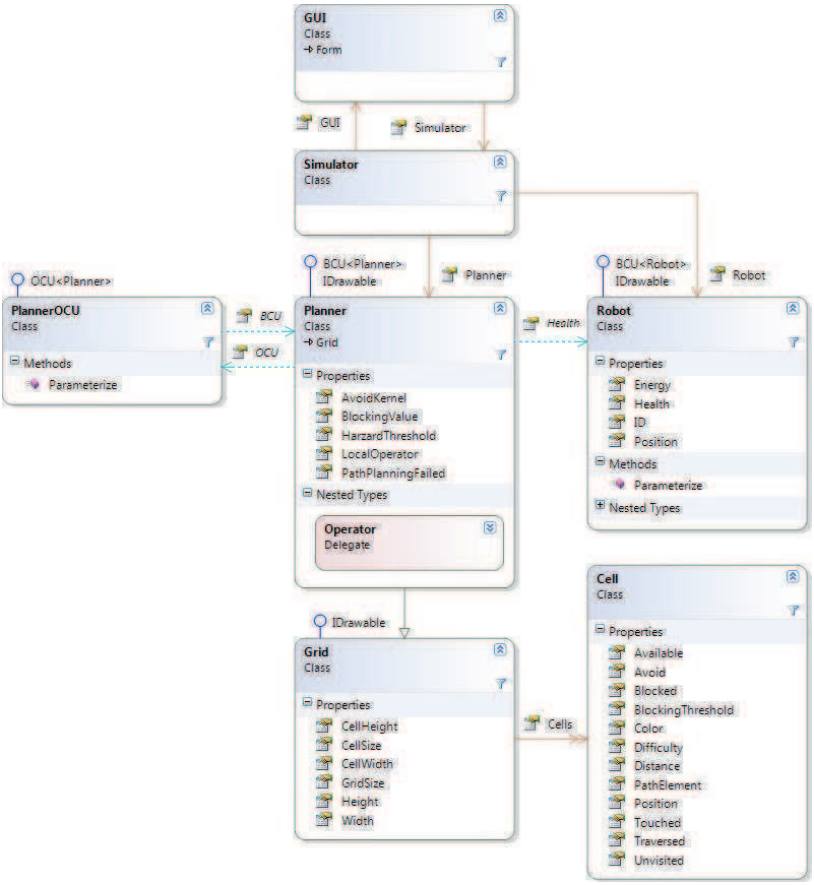


Figure 5: A simplified overview of the simulator's structure. As an adaption the ORCA specific connections are shown as dashed lines.

5 Results

In the following the behavior of the proposed approaches for path planning in crowded environments with OCU adjusted hazard estimation will be presented. The first two subsections show exemplary setups for certain path planning aspects, while the third subsection presents results for a path planning scenario on ten arbitrary maps.

5.1 Avoidance of Crowded Areas

A basic variant of the modified path planning, where the OCU modifies r in respect to the health signal was simulated in the environment shown in Figure 6. In this case each hazard is treated as an obstacle. The left subfigure shows the path for a healthy robot. As can be seen the robot travels through very narrow corridors and has to perform several turns.

In the right subfigure of Figure 6 the resulting path avoids narrow corridors and a lot of turns to different sides, because the narrow spaces between obstacles have been interpreted as virtual obstacles.

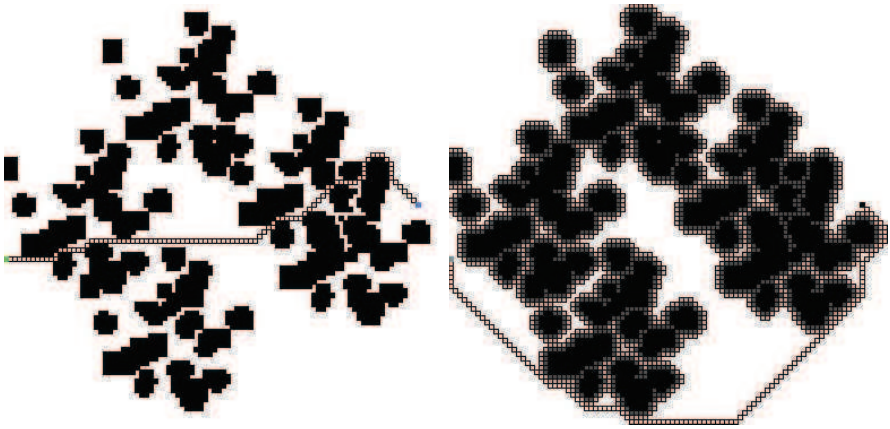


Figure 6: The effect of different health signals (left: 100%, right: 63%) on the resulting path. Fixed obstacles are depicted in black, hazardous areas are colored in shades of gray and the path is shown as a interconnected line of white cells.

5.2 Hazard Adaptation

For a better understanding of the result it has to be recalled that the path planning via a wave front expansion works in two major steps. First, the whole terrain is flooded from the goal by the wave front to generate distance measures and additional information like hazard estimations. Secondly the path is chosen by a local operator starting at the position of the robot. An exemplary result of this approach is shown in Figure 7.

The left subfigure in Figure 7 illustrates the effect of OCU interventions in case of very lose hazard avoidance during the wave front expansion, by accepting all hazardous cells as possible path cells, but a strict avoidance (if possible) in the local operator. As one can see the resulting path avoids hazardous cells as long as possible while still moving toward the goal. At a certain point the local operator has no other choice than to enter possibly hazardous cells in order to get closer to the goal, but stays on cells that are the least dangerous in the local surrounding. It has to be mentioned that this is a very extreme example to show pure performance of the local operator without hazard adjustment of the OCU (all cells without obstacles are ultimately traversable).

A better solution is a stricter avoidance of hazards during the wave front expansion that threads areas with large hazard index as fixed obstacles, which should never be traversed. Then the local operator can choose a more suitable path for the robot. The result of such an approach under same conditions is shown in the right subfigure of Figure 7.

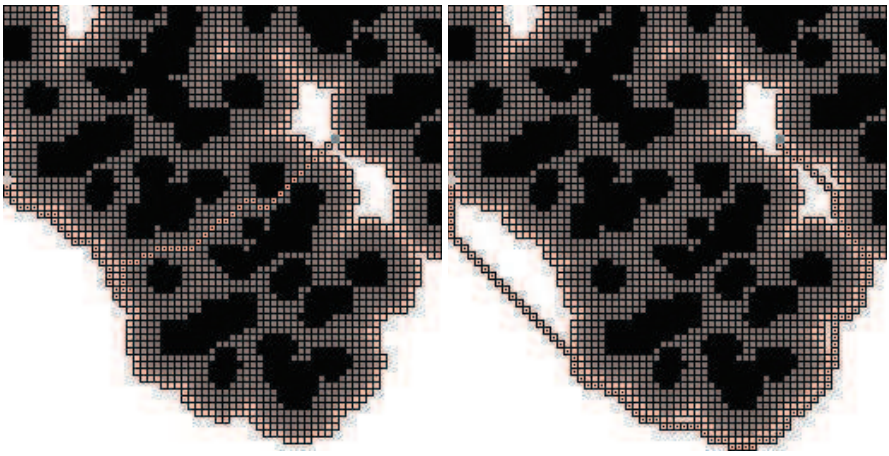


Figure 7: The effect of different OCU adjustments during the path planning in case of a very severe damage (health signal = 6%). In the left subfigure all hazards are marked as ultimately traversable if needed, whereas in the right subfigure severe hazards are marked as virtual obstacles. Fixed obstacles are shown in black, hazardous areas are colored in shades of gray and path elements are shown as dotted cells.

5.3 Automated Test Set

For the automated test the robot had to traverse ten arbitrary grids of 600×400 cells with the top left cell as starting point and the bottom right cell as goal. The first five maps feature fixed obstacles on an evenly difficult terrain, while the other maps feature fixed obstacles on arbitrary difficult terrain. The path planning on this set of maps was performed with an OCU as described in subsection 3.2 (including path segmentation) and a linear dependency between the health signal and t. Each map had to be traversed twice.

First with a fully functional robot (health signal = 100%) and then with the lowest health signal possible that still allows the robot to reach the goal.

The results of the automated test with ten arbitrary maps can be found in Figure 8. The value next to the name of the map shows the minimal health signal value that guaranteed a valid path to the goal. The value *Minimal Distance (healthy)* describes the minimal distance between the nearest obstacle and the robot (health signal = 100%) while it traverses the map. Due to reasons of clarity this value is also shown as number. The next value (*Minimal Distance*) is the minimal distance between the nearest obstacle and a damaged robot (the corresponding health signal is shown next to the name of the map). The *Average Distance Ratio* is the average distance between obstacles and the damaged robot compared to the average distance between obstacles and a healthy robot. The last value (*Path Ratio*) rates the traveled distance of a damaged robot in comparison to a healthy robot.

As can be seen the damaged robot holds a greater safety distance to obstacles than a healthy robot and accepts detours to achieve this goal. On average a faulty robot keeps an 1.47 times greater distance to obstacles, while traveling a path that is 1.32 times longer.

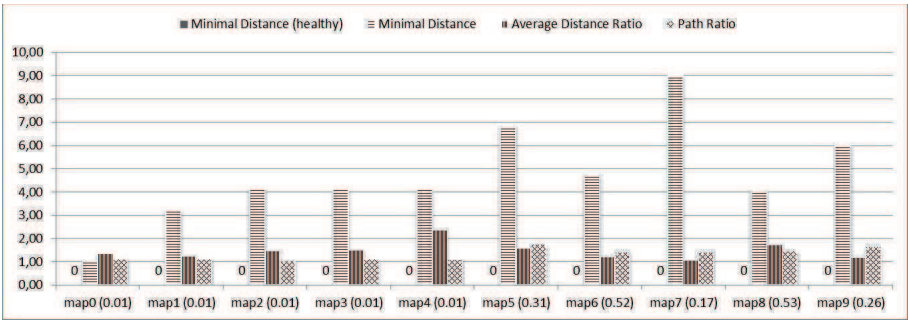


Figure 8: This figure shows the results of the automated test set. The term distance describes the minimal distance between the nearest obstacle and the robot during its movement. The first value is the minimal distance of a healthy robot. The other three values rate the performance of a damaged robot proportional to a healthy robot.

6 Conclusion and Outlook

It has been shown that adapting path planning is able to avoid possible hazards that are introduced by obstacles in the near surrounding of a faulty robot. This is done via the local calculation of possible hazards in dependence of the distance to an obstacle. With these hazard estimations the path planner is able to plan paths that keep a greater safety distance between the robot and fixed obstacles by accepting detours on its way towards the goal.

Future work on this topic will focus on the extension of the presented approach towards the support of multiple robots.

References

[ALM08] Adam El Sayed Auf, Marek Litza, and Erik Maehle. Distributed Fault-Tolerant Robot Control Architecture Based on Organic Computing Principles. In *Biologically Inspired Collaborative Computing, International Federation for Information Processing (IFIP)*, volume 268, pages 115–124, 2008.

[Ark98] Ronald C. Arkin. *Behavior-Based Robotics*. The MIT Press, 1998.

[ARS10] Francesco Amigoni, Monica Reggiani, and Viola Schiaffonati. Simulations Used as Experiments in Autonomous Mobile Robotics. In *ICRA2010 (International Conference on Robotics and Automation) Workshop on "The Role of Experiments in Robotics Research"*, 2010.

[BMS04] Adi Botea, Martin Müller, and Jonathan Schaeffer. Near Optimal Hierarchical Path-Finding. *Journal of Game Development*, 1:7–28, 2004.

[CH09] Annett Chilian and Heiko Hirschmüller. Stereo Camera Based Navigation of Mobile Robots on Rough Terrain. In *Intelligent Robots and Systems, 2009. IROS 2009. IEEE/RSJ International Conference on*, pages 4571–4576, October 2009.

[GWT+12] Alexander Gloye, Fabian Wiesel, Oliver Tenchio, Mark Simon, and Raúl Rojas. Robot Heal Thyself: Precise and Fault-Tolerant Control of Imprecise or Malfunctioning Robots. online, `http://www.inf.fu-berlin.de/inst/ag-ki/rojas_home/documents/2005/robotheal.pdf.pdf`, Accessed: January 2012.

[JC89] Paul Jacobs and John Canny. Planning Smooth Paths for Mobile Robots. In *Robotics and Automation, 1989. Proceedings., 1989 IEEE International Conference on*, volume 1, pages 2–7, May 1989.

[LJA+07] Svetlana Larionova, Bojan Jakimovski, Adam El Sayed Auf, Marek Litza, Florian Moesch, Erik Maehle, and Werner Brockmann. Towards a Fault Tolerant Mobile Robot: Mutual Information for Monitoring of the Robot Health Status. In *International Workshop on Technical Challenges for Dependable Robots in Human Environments, IARP, EURON, IEEE/RAS*, 2007.

[MBG+11] Erik Maehle, Werner Brockmann, Karl-Erwin Grosspietsch, Adam El Sayed Auf, Bojan Jakimovski, Stephan Krannich, Marek Litza, Raphael Maas, and Ahmad Al-Homsy. Application of the Organic Robot Control Architecture ORCA to the Six-legged Walking Robot OSCAR. In *Organic Computing - A Paradigm Shift for Complex Systems*, pages 517–530. Birkhäuser-Springer, 2011.

[MM11] Raphael Maas and Erik Maehle. Fault Tolerant and Adaptive Path Planning for Mobile Robots Based on Health Signals. In *Architecture of Computer Systems, 2011. ARCS 2011. 24th International Conference on*, pages 58–63, February 2011.

[NI09] Amir Nooraliei and R. Iraji. Robot Path Planning using Wavefront Approach with Wall-Following. In *Computer Science and Information Technology, 2009. ICCSIT 2009. 2nd IEEE International Conference on*, pages 417–420, August 2009.

[NN09] Amir Nooraliei and Hamed Nooraliei. Path Planning Using Wave Front's Improvement Methods. In *Computer Technology and Development, 2009. ICCTD '09. International Conference on*, volume 1, pages 259–264, November 2009.

[RMB+06] Urban Richter, Moes Mnif, Jürgen Branke, Christian Mller-Schloer, and Hartmut Schmeck. Towards a generic observer/controller architecture for organic computing. In *Workshop Organic Computing-Status and Outlook, INFORMATIK 2006, Proc. 36. GI-Jahrestagung. Lecture Notes in Informatics*, October 2006.

ART Networks as Flexible Means to Implement Dependability Properties in Autonomous Systems

K.-E. Großpietsch
EUROMICRO, P.O. Box 2043
Sankt Augustin, Germany
karl-erwin.grosspietsch@online.de

Tanya A. Silayeva
Moscow Aviation Institute, Volokolamsk Highway 4
125871 Moscow, Russia
ta.silaeva@mail.ru

Abstract: In this paper, the potential of adaptive resonance theory (ART) networks for dependability issues is considered. The basic properties of ART architectures are described, and some strategies are discussed to enable a balanced combination of performance and dependability requirements by these networks.

1 Introduction

The development of IT systems in general is mainly realized under observing constraints as costs and resulting perfomance. Dependability issues usually have to be brought into compliance with these two major requirements. In real-time computing, this need is explicitly shining up in the requirement that the system is not only correctly working, but guarantees to fulfil tasks within given upper time bounds: I.e. functioning alone is not enough, sufficiently quick functioning is necessary.

In many applications, the relation between performance orientation and dependability orientation must not be fixed, as e.g dependability-critical and -uncritical situations may quickly follow each other. Consider e.g. a robot moving in unknown territory. The cautious approach, always to move very slowly, always to perfectly check for all possible dangers might imply too much loss of velocity, to reach the goal location in the required time. So, what we would need is a flexible strategy, which also considers performance issues, while not neglecting some background cautiousness.

Here I would like to propose to consider adaptive resonance theory (ART) [CG88], a subfield of the theory of neural networks, as an interesting offer, albeit not yet investigated in more detail with regard to application in the dependability community. In section 2 the main characteristics of ART networks will shortly be exhibited.

Subsequently, in section 3 some ideas about exploiting the ART structure for dependability issues are discussed.

2 Basic Structure and Properties of ART Networks

Standard neural networks, as e.g. backpropagation networks, work in two different modes: the training phase and the recognition phase. In the training phase, given pairs of an input pattern and an output pattern are offered to the – arbitrarily initialized – network. The network tries to systematically minimize the error in reconstructing the output from the input pattern, by corresponding changes of the weight factors of the neurons, until the error reaches 0 or a sufficiently small value. After the end of the training period, these values are frozen so that no change of the learned experience is possible any more.

Then, in the recognition phase the network is to classify unknown input patterns as similar to certain learned inputs, and, thus, to sufficiently exactly reproduce the corresponding output patterns.

In many applications, however, as e.g. for the movement of autonomous robots in unknown territory, it would be desirable to adapt again the experience of the network to the changing environment. However, simply extending the training to the operational phase of the system causes the tradeoff that this treatment would destroy part of the experience learned during the initial training phase.

Here, ART networks have been proposed as a remedy [CG88]. In the basic ART architecture, the entire recognition process mainly proceeds as follows (see also Fig. 1): The input pattern inp is implemented as a vector of Boolean numbers. It activates the neurons i (i=1,...,m) of the so-called comparison layer F1. The output of the layer neuron i is a function $s_i = f(inp_i)$; often simply the identical reproduction of the input vector inp is assumed: $s_i = inp_i$ (i=1,...m). For the subsequent discussion, we shall also adopt this assumption s=inp.

The vector s is multiplied by the so-called bottom-up matrix B_{ij} the elements of which are real numbers. This produces a real number vector t comprising as components the weighted sums

$$t_j = \sum_{i=1}^{m} B_{ij} * s_i = B_i * s \quad (j=1,...,n \; ; \; B_i \text{ being the row vector i of matrix } B_{ij})$$

The maximum of these sums is determined:

$$t_k = \max_j t_j$$

Neuron k of the recognition layer F2 is then set to 1; all other neurons of this layer are

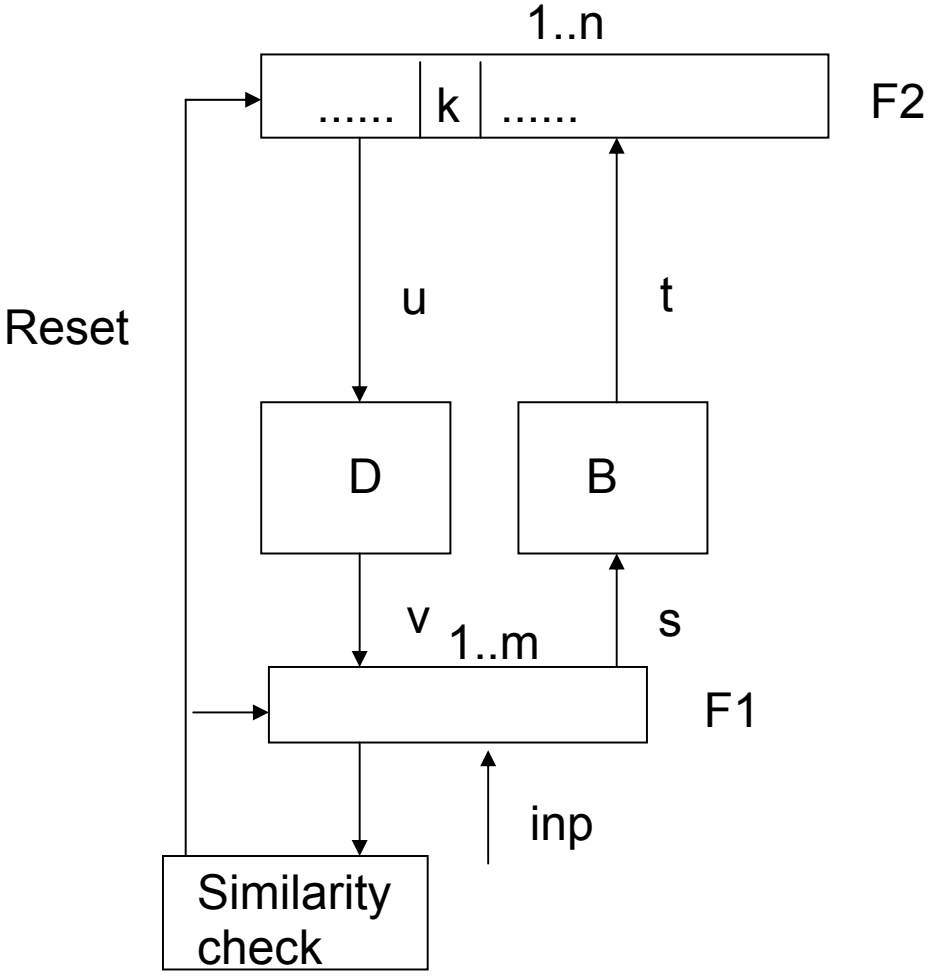


Fig. 1 Basic architecture of ART networks (according to [CG88, Ze 97])
F1 comparison layer, F2 recognition layer, B bottom-up matrix, D top-down matrix, Reset reset line, inp input vector; s, t, u,v generated vectors (see text), k winning neuron of layer F2

set to 0 (neuron k is the „winner neuron“). I.e. the neuron k of F2 represents the class of patterns, to which the input pattern inp is, in this first selection, estimated to belong.

Subsequently, this decision is checked by a control computation. To do so, the vector u of the Boolean values of the recognition layer neurons is multiplied by a second matrix, the so-called top-down matrix D_{ij} , producing a real number vector v:

$$v_i = \sum_{j=1}^{n} D_{ij}\, u_j = d_{ik} * u_k = d_{ik} \quad \text{for } i=1,...,m$$

By AND ing the components of vector v and of the input vector inp of layer F1, a check vector c is formed:

$c_i = v_i$ AND inp_i for $i=1,...,m$

Finally, the similarity of c and input vector inp is compared. This similarity is measured by counting the numbers n_c and n_{inp} , respectively, of 1s in both vectors, and forming their quotient

$q = n_s / n_e$.

If q is larger than a previously selected value of a so-called tolerance parameter p, the input inp is assumed to be sufficiently close, „in resonance", to the column vector D_i of matrix D. If this is not the case, the classification approach is decided to be not fitting. Then the entire recognition process is repeated, with the previous winner neuron k being excluded from the selection process. This causes that the new maximum

$$t_l = \max_{j=1,...,k-1,k+1,...,n)} t_j$$

is etstablished, so that now another neuron l is the winner neuron and the subsequent check for its resonance with the input vector inp is carried out.

This search loop is repeated until a resonant solution is found. If all the actual n class representations of the neural network do not fit to the input vector, an additional neuron n+1 is created in layer F2, and is set to 1. Correspondingly, the number of pattern classes distinguishable by the ART network, increases by one. So, the network is able to respond to the appearance of unidentified patterns, by the creation of new classes.

After the search process for the given input vector inp has been completed, the elements of the matrices B and D are updated. Updating of bottom-up matrix B is done by changing the row vector which had, in the procedure described previously, produced the maximum sum; this row vector is torn towards the input vector inp. All other rows of matrix elements remain unchanged. Updating of the top-down matrix D is done by component-wise ANDing by the vector s. For the mathematics of the update formulas, and also the influence of some additional so-called gain factors see [CG88, Ze97] which provide a comprehensive discussion of these details.

Also, in this position paper, for ease of understanding we confine the discussion to considering the basic ART architecture ART-1 described in this section; upgrading modifications of this architecture allow e.g. the use of real number input vectors [CG91a, CG91b] or combinations with fuzzy logic [CG91c].

3 Application of ART Networks for the Dependability of Systems

As the main advantage of ART networks their „plasticity“ is claimed, i.e. the ability to integrate new knowledge into the network without destroying old one. Moreover, this is done in a balanced way where still the attempted changes are checked aginst the memorized knowledge, stored in the top-down matrix. So, the network emulates, to a certain degree, the cooperation between short term storage and long term storage known from biological systems: Actual on-the-fly experience or situation estimation might flow into the current input vector inp, and then can be integrated into memory under the surveillance of the long-term knowledge stored in the top-down matrix D.

This scheme might be applied to many applications, especially of autonomous systems as, e.g., robots: In easily-manageable landscapes like plains, the robot without risk could be allowed to move quicker. On the other hand, a situation where none of the stored strategies (each represented by one neuron of the recognition layer) is acceptable, would indicate an extraordinary situation where a high degree of cautiousness has to be paid while exploring this situation and its potential consequences; i.e. where the robot should be in a „high alert“ state.

So, the use of an ART network for control would enable a a three-stage scheme of adaptivity:

- on the one hand, we have individual strategies (each represented by one neuron of the recognition layer) which have learned to adapt to a class of input patterns;

- the network provides switching from one strategy to another one in a quite simple way;

- moreover, for extraordinary situations, considering a new, additional strategy is possible.

We have pointed out that the ART network very flexibly enables the implementation of new distiguishable decision classses. In the basic ART approach, however, it is left open, how, as response to the input, a certain output pattern o, e.g. comprising control signals, is to be generated.

Here we propose to adopt the solution used e.g. in counter-propagation networks [He88], to utilize, after having completed the search for the winner neuron, the recognition layer contents u, also for a matrix vector multiplication with an additional matrix, called here the output pattern matrix OP (see Fig. 2):

$$o_i = \sum_{j=1}^{n} OP_{ij} * u_j = OP_{ik} \; (i=1, ..., n^*),$$

k being the index of the winning neuron in F2, and n* the number of rows in matrix OP (i.e. n* is the width of the control pattern output).

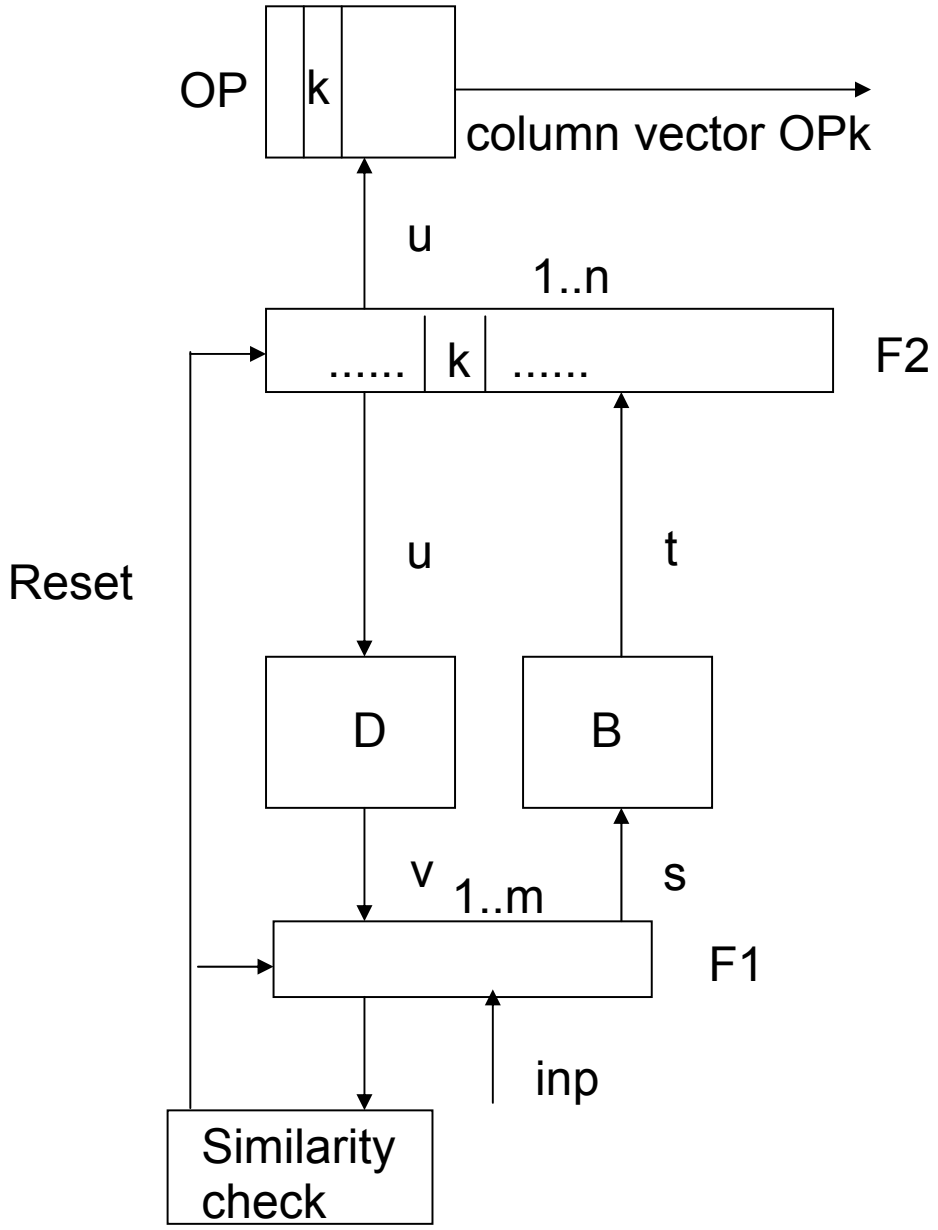


Fig. 2 Extension by an associative memory
F1 comparison layer, F2 recognition layer, B bottom-up matrix, D top-down matrix, Reset reset line, inp input vector; s, t, u,v generated vectors (see text); k winning neuron of layer F2. Additionally the matrix–vector multiplication OP * u produces as result the column vector OP_k of matrix OP, thus providing output of a data pattern associated to the winner neuron k of layer F2. That means the result vector o is just the column vector

OP_k of matrix OP. So, depending on the contents of the input vector inp (not via a memory address !) the output pattern is accessed from the „memory“ OP, i.e. OP functions like an associative memory.

To conclude, the ART network approach provides a promising strategy to implement control schemes especially for autonomous systems, where dependability issues as well as performance considerations can flexibly be combined in a balanced way. Of course, beyond the general concept outlined here, several detail problem areas are still open for further investigation, e.g

- efficient definition of the traning procedure for the matrix OP, to produce the contents of the associative memory;

- extending or modifying the strategy sketched here, to cope with the modifications of the basic ART network, mentioned in the literature.

For tackling such problems, we would welcome colleagues interested in this field, for some future cooperation.

4 Conclusion

In this paper, the potential of adaptive resonance theory (ART) networks for dependability issues has been discussed. The basic properties of ART architectures are outlined, and some strategies are shown to enable a balanced combination of performance and dependability requirements by these networks.

5 References

[CG88] Carpenter, G.A.; Grossberg, S.: The Art of Adaptive Pattern Recognition by a Self-Organizing Neural Network, Computer, March 1988, pp. 77-88

[CG91a] Carpenter, G.A.; Grossberg, S.; Rosen, D.B.: ART2-A: An Adaptive Resonance Algorithm for Rapid Category Learning and Recognition, Neural Networks, Vol. 4, 1991, pp. 493-504

[CG91b] Carpenter, G.A.; Grossberg, S.; Reynolds, J.H.: ARTMAP: Supervised Real-Time Learning and Classification of Nonstationary Data by a Self-Organizing Neural Network, Neural Networks, Vol. 4, 1991, pp. 565-588

[CG91c] Carpenter, G.A.; Grossberg, S.; Rosen, D.B.: Fuzzy ART: Fast Stable Learning and Categorization of Analog Patterns by an Adaptive Resonance System, Neural Networks, Vol. 4, 1991, pp. 759-771

[He88] Hecht-Nielsen, R.: Applications of Counterpropagation Networks, Neural Networks, Vol. 1, 1988, February, pp. 131-139

[Ze97] Zell, A.: Simulation neuronaler Netzwerke, R. Oldenbourg Verlag München 1997

Model-based Testing of Autonomous Systems based on Coloured Petri Nets

Raimar Lill, Francesca Saglietti

Chair of Software Engineering
University of Erlangen-Nuremberg
Martensstr. 3
91058 Erlangen, Germany
raimar.lill@informatik.uni-erlangen.de
saglietti@informatik.uni-erlangen.de

Abstract: The use of autonomous systems, including cooperating agents, is indispensable in certain fields of application. Nevertheless, the verification of autonomous systems still represents a challenge due to lack of suitable modelling languages and verification techniques. To address these difficulties, different modelling languages allowing concurrency are compared. Coloured Petri Nets (CPNs) are further analysed and illustrated by means of an example modelling autonomous systems. Finally, some existing structural coverage concepts for Petri Nets are presented and extended by further criteria tailored to the characteristics of CPNs.

1 Introduction

For reasons of flexibility and time efficiency modern software-based applications tend to decentralize the target functionality among a number of cooperating autonomous subsystems. This results in highly decoupled system units and an increasing multiplicity of interplay. Examples include mobile agents, as typical for robot applications.

In general, the term autonomy is taken to indicate entities (humans, populations or technical systems) capable of providing themselves with their own laws. This includes the moral responsibility of individuals for their actions, the self-government of human populations, as well as the capacity of technical systems to make rational and informed decisions.

The optimal degree of autonomy lies between the two extremes represented by fully decoupled agents and fully central controllers. It relies on essential rules of co-existence involving different communication patterns like the following ones:

- **Separation of concerns:** different agents may autonomously cooperate by carrying out parallel individual and independent sub-tasks.
- **Synchronization:** as soon as each independent sub-task has been separately and autonomously concluded, a synchronization mechanism establishes appropriate communication between the cooperating parts for the purpose of organizing the next common activity.
- **Coordination:** the communication of synchronized agents must be supported by a coordination mechanism determining a consensus on how to proceed. Typically, coordination tasks may be required to avoid or resolve conflicts due to concurrency, e.g. by mutual exclusion in case of shared resources.
- **Delegation:** for the purpose of carrying out a complex activity, a process may delegate a sub-task to another process by invoking its support through synchronous or asynchronous message passing.
- **Feedback:** agents may mutually influence their local or global behaviour by providing information on particular operating scenarios recently experienced, e.g. information on road traffic exchanged among communicating vehicles. The degree of influence exercised may depend on the amount and consistency of information broadcasted or by the number of agents broadcasting it.

From the perspective of verification, one of the major challenges posed by autonomous systems refers to their inherent lack of compositionality. In fact, understanding the local behaviour of each individual system part does not suffice to comprehend all potential implications on global behaviour. In other words, the classical verification strategy based on separation of concerns provides only limited support here, as it does not allow for the deduction of global properties by derivation or composition of local properties.

The observable *emergent behaviour* may reflect the intended target behaviour to be achieved by cooperation, but sometimes it may also reveal as surprising, undesired, or even unsafe by cascading of unpredicted side effects.

In order to model concurrent behaviour of cooperating autonomous systems capturing the interaction patterns mentioned above, an appropriate modelling notation is to be selected, capable of supporting both the representation and the analysis of the system considered. As the observation and measurement of system behaviour in real situations is crucial for testing autonomous systems, representative test scenarios have to be derived from the modelling notation selected.

The rest of the article is structured as follows: in section 2 different modelling notations are compared w.r.t. the considerations mentioned above. In section 3 CPNs are introduced by illustrating their syntactical and semantical highlights. In section 4 a CPN model of a system consisting of autonomous robots is introduced and illustrated in detail. This example confirms the decision taken in favour of CPNs and emphasizes the value of the corresponding modelling tool CPN Tools [JKW07]. In chapter 5 existing coverage criteria for Petri Nets are presented before introducing novel coverage criteria explicitly tailored to CPNs by focusing on token colours and on the occurrence of interleaving events in the net.

2 Comparison of Modelling Notations for Autonomous Systems

For the purpose of comparing different notations modelling autonomous systems, the following evaluation criteria were considered:

- **Understandability** concerns the clarity of a modelling representation, including the availability of graphical visualization techniques.
- **Well-definedness** ensures a unique interpretation of the underlying operational semantics as offered by formal languages and required for the analyzability of central properties like state reachability. A well-defined semantics is the basis for tool support concerning the analysis of model data.
- **Scalability** concerns the ease of widening the problem dimensions without prohibitively increasing problem complexity and representational size.
- **Testability** concerns the ease of capturing and visualizing the multiplicity of behaviour by adequate coverage concepts and metrics; central to these activities are tools supporting the editing, import and export of data.

In the light of the criteria mentioned above a number of well-known and widely used modelling languages allowing for concurrent behaviour are compared in the following.

- **Process algebras** like CCS [Mi80] or CSP [Ho78] provide a formal algebraic description of model concurrency aspects capturing communication by explicit algebraic *send* and *receive* operators.
- **UML activity diagrams** provide a semi-formal, graphical representation of activity flows which can be extended by dedicated profiles to include real-time concurrent behaviour; among them are the profile "Schedulability, Performance and Time" (SPT) [Om05], which allows for quantitative performance predictions, and its UML 2 successor "Modelling and Analysis of Real-Time and Embedded Systems" (MARTE) [Om11].
- **Petri Nets** [Mu89] formalize concurrent behaviour by graphical entities representing actions (so-called *transitions*), as well as pre- and post-conditions (so-called *places*). The fulfilment of conditions is represented by tokens marking corresponding places. In this way, Petri Nets succeed in capturing the interactions of autonomous systems by decentralizing the information referring to their states.
- **Coloured Petri Nets** [JKW07] are an extension of Petri Nets allowing for the refinement of pre- and post-conditions while supporting scalability by use of different token types (so-called *token colours*). Details are presented in section 3.

Process algebras possess a well-defined algebraic theory making them adequate for static analysis purposes. The lack of visual representation, however, does not offer sufficient support to intuitive comprehension and graphical coverage concepts.

UML activity diagrams are widely used and provide an intuitive visualization of concurrency aspects. They lack, however, a formal operational semantics [SH05], as required by rigorous analysis techniques.

Petri Nets are well suited for visualization and analysis purposes [Mu89]; unfortunately, the decentralized representation of states by generic tokens may lead to an exponential growth of places and transitions. In other words, their scalability may be severely limited.

Compared to the other approaches, Coloured Petri Nets reveal as a promising candidate for modelling autonomous systems: CPNs are based on a sound mathematical basis resulting in unambiguous models that can be analyzed by simulation or formal techniques. Furthermore, they benefit from the visual clarity of Petri Nets by sharing their graphical elements, while overcoming the Petri Nets limitations concerning scalability: in fact, the use of specific tokens supports a compact state representation. In other words, when compared with Petri Nets CPNs offer higher scalability thanks to the higher level of the language.

Consequently, CPNs are selected as a modelling notation to be further investigated.

3 Coloured Petri Nets

Coloured Petri Nets [Je94] differ from the original Petri Net notation by type-specific tokens. Depending on the colour set associated with a particular token, this token may assume different values denoted as the *token colours*. Additionally, transitions and arcs can be annotated by conditional expressions controlling the transition firing. In more detail, a CPN is a 9-tuple (P, T, A, Σ, V, C, G, E, I), where

- P denotes a finite set of nodes $p \in P$ denoted as *places*;
- T denotes a disjoint finite set of nodes $t \in T$ denoted as *transitions*, i.e. $P \cap T = \emptyset$;
- $A \subseteq P \times T \cup T \times P$ denotes a set of directed *arcs* connecting either places with transitions or transitions with places;
- Σ denotes a finite set of non-empty sets denoted as *colour sets*; the elements of each colour set are denoted as *colours*;
- V denotes a finite set of *variables*, each varying over a colour set, i.e. $type[v] \in \Sigma \; \forall \; v \in V$;
- $C: P \rightarrow \Sigma$ denotes a function (the so-called *colour function*) attaching to each place $p \in P$ a colour set $C(p) \in \Sigma$; $C(p)_{MS}$ denotes the multi-set over C(p) where each colour of the colour set C(p) may occur more than once;
- $G: T \rightarrow EXP_V$ is a function attaching to each transition $t \in T$ a *guard,* i.e. a conditional expression G(t) over variables $v \in V$ with type[G(t)] = Bool;
- $E: A \rightarrow EXP_V$ is a function attaching to each arc $a \in A$ an *expression* E(a) over variables $v \in V$, with $type[E(a)] \in C(p)_{MS}$, where p is a place connected with arc a;
- $I: P \rightarrow C(p)_{MS}$ denotes a function attaching to each place $p \in P$ a so-called *initial marking* M(p) of type $C(p)_{MS}$.

The dynamic behaviour of a CPN is given by state changes due to successive firings of transitions.

For a given transition $t \in T$ a *variable binding* associates each variable occurring in at least one input arc expression of t with one of the colours belonging to the colour set of the corresponding input place(s).

A transition $t \in T$ with input places p_i and input arcs a_i: $p_i \rightarrow t$, $i \in \{1,\ldots,k(t)\}$ is *enabled w.r.t. a particular variable binding* if and only if

- the value of G(t) w.r.t. the given variable binding is true and
- for each colour of an input place the colour multiplicity in the multi-set obtained by evaluating the input arc expression $E(a_i)$ w.r.t. the given variable binding is not higher than the number of tokens of the same colour in the corresponding input place.

After *firing* a transition t w.r.t. an *enabling variable binding*, a new marking is obtained from the previous marking by

- removing from each input place as many tokens for each of its colours as indicated by the colour multiplicity in the multi-set resulting by evaluating the corresponding input arc expression w.r.t. the enabling variable binding;
- adding to each output place as many tokens for each of its colours as indicated by the colour multiplicity in the multi-set resulting by evaluating the corresponding output arc expression w.r.t. the enabling variable binding.

4 The Robot Factory

In the following, a CPN model of an autonomous robot system is presented. The model was built using CPN Tools [JKW07]. Net annotations are expressed in the language CPN ML which extends the functional programming language SML [Mi97] by additional constructs for defining the CPN elements presented above. As commonly used in Petri Net visualizations, places and transitions are represented by circles and rectangles respectively. Guards are encapsulated in square brackets, arc inscriptions are annotated along the corresponding arcs and places are assigned corresponding colour sets by cross products of pre-defined basic colour sets.

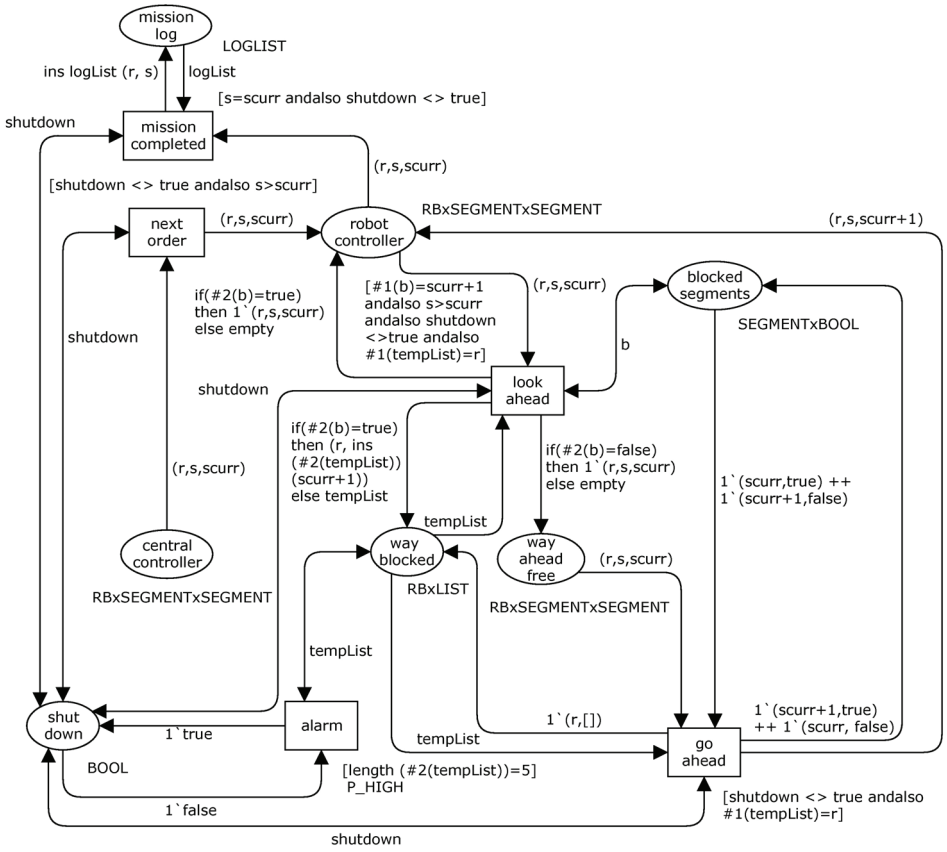


Figure 1: CPN Model of the Robot Factory

The example depicts a factory where robots of type *RB* move from one area – a so-called *SEGMENT* – to another one. A central controller sends orders *(r, s, scurr)* to the local robot controllers where

- *r* represents the *robot* addressed,
- *s* represents the *target segment* of robot *r*, and
- *scurr* represents the *current location* of robot *r*.

To limit the complexity of the example, the robots only move along narrow lanes and obstacle passing manoeuvres are not included, thus limiting the robot behaviour to forward movement. It should be noted that the central controller does not hinder system autonomy, as it only provides the robots with orders, without dictating their behaviour for fulfilling their missions. Robots autonomously check whether their way to the next segment is free or blocked (modelled by transition *look ahead*). A robot can access this information thanks to its sensors (e.g. by camera or laser techniques). The functioning of the sensors is not explicitly modelled here, but could be easily included, e.g. by a hierarchical design.

For the purpose of deriving valuable test scenarios from the modelled CPN, the necessary sensor data are simulated by the place *blocked segments* containing the local knowledge of each robot about the segment lying ahead.

If the sensors of a robot detect a blockade, the information about the target segment and the robot is logged in the place *way blocked* by maintaining a list of blockades for every robot. Five continuously detected blockades of a certain robot trying to access a specific segment raise an alarm (modelled by transition *alarm*). To ensure this, CPN Tools allows to assign different priorities to transitions. The alarm was prioritized to avoid the interference of other transitions resulting in a potential alarm delay. Raising an alarm prevents any further transition from firing, requiring the intervention of a human operator. If a blockade can be resolved before raising an alarm, the blockades are flushed from the corresponding list. This is realized by the use of list colour sets in analogy to abstract list data types in programming languages.

As soon as the way is not blocked by obstacles or other robots, a robot can move on (modelled by transition *go ahead*) resulting in an update of its current location *scurr*.

If a robot eventually detects that its current segment position *scurr* equals its target position *s*, the given order is completed (modelled by transition *mission completed*) and logged.

The strength of modelling the behaviour of autonomous systems by CPNs lies in their ability to capture a wide multiplicity of possible execution traces by a relatively compact representation. This results in a flexible format that can be easily adapted to application-specific scenarios by changing the configuration of the CPNs (e.g. by varying the number of segments or robots).

Possible execution traces regarding an initial net marking can be statically analyzed by exploring the reachability graph [Je94] or dynamically analyzed w.r.t. different test coverage criteria, as presented in the next chapter.

5 Testing Coverage Criteria for CPNs

Several CPN coverage criteria can be defined to determine appropriate scenarios during testing. For example, [ZH00] and [ZH02] introduce a number of coverage criteria originally defined for Predicate-Transition Petri Nets [GL81] which inspired the following classes of CPN testing criteria.

- **Transition-based coverage criteria** focus on the occurrences of transition firings, requiring individual transition firings as well as sequences of transition firings of given length.
- **State-based coverage criteria** focus on individual states or on state representatives of pre-defined state classes.

– **Flow-based coverage criteria** focus on the production or consumption of at least one token (regardless of its colour) by individual transition firings or by sequences of transition firings.

The above mentioned criteria can be extended to include the following coverage demands concerning CPN-specific entities, namely colour sets and variable bindings.

– **Colour-based coverage criteria** focus on the production or consumption of tokens belonging to pre-defined colour sets.
– **Event-based coverage criteria** focus on the individual or sequential occurrence of CPN events, where an *event* is defined as a transition together with an enabling variable binding.

The information needed to measure the coverage regarding the criteria presented above can be extracted from models created in CPN Tools by the framework Access/CPN [WK09] which permits to access the internal model data. It is planned to apply this framework in order to generate test cases fulfilling the above mentioned CPN coverage criteria by means of evolutionary techniques. Analogous approaches have already been successfully applied to structural code coverage [OS06], to integration testing [SP10], as well as to the filtering of operational experience for the purpose of reliability assessment [Sö10].

6 Conclusion

This article compares different modelling notations in terms of their expressive power and graphical support concerning structural testing of autonomous systems. The comparative evaluation of a number of alternative notations resulted in the selection of Coloured Petri Nets as the most promising option. As an example, a simple version of a robot factory was modelled by a CPN illustrating the benefits offered by its high scalability. Finally, a number of different CPN-based coverage criteria were introduced which will provide the basis for future research focused on the automatic generation of adequate testing scenarios.

Acknowledgement: It is gratefully acknowledged that part of the work reported was sponsored by the European Union Research Programme ARTEMIS (Advanced Research and Technology for Embedded Intelligence and Systems), project R3-COP (Resilient Reasoning Robotic Co-operating Systems).

References

[GL81] Genrich, H. J.; Lautenbauch, K.: System Modelling with High-Level Petri Nets. In: Theoretical Computer Science, Vol. 13, Issue 1. Elsevier, 1981; pp. 109-136.

[Ho78] Hoare, C. A. R.: Communicating Sequential Processes. In: Communications of the ACM, Vol. 21, No. 8. ACM Digital Library, 1978; pp. 666-677.

[Je94] Jensen, K.: An Introduction to the Theoretical Aspects of Coloured Petri Nets. In (de Bakker, J. W.; de Roever, W.-P.; Rozenberg, G. Eds.): A Decade of Concurrency - Reflections and Perspectives, Proc. REX School/Symposium, Noordwijkerhout, the Netherlands, 1993. Vol. LNCS 803, Springer-Verlag, 1994; pp. 230-272.

[JKW07] Jensen, K.; Kristensen, L. M.; Wells, L.: Coloured Petri Nets and CPN Tools for Modelling and Validation of Concurrent Systems. In: International Journal on Software Tools for Technology Transfer (STTT), Vol. 9, No. 3-4. Springer-Verlag, 2007; pp. 213-254.

[Mi80] Milner, R.: A Calculus of Communicating Systems. Vol. LNCS 92, Springer-Verlag, 1980.

[Mi97] Milner, R. et al.: The Definition of Standard ML (Revised). MIT Press, 1997.

[Mu89] Murata, T.: Petri Nets: Properties, Analysis and Applications. In: Proc. IEEE, Vol. 77, No. 4. IEEE, 1989; pp. 541-580.

[Om05] Object Management Group: UML Profile for Schedulability, Performance and Time Specification. Version 1.1, formal/05-01-02, 2005.

[Om11] Object Management Group: UML Profile for MARTE: Modeling and Analysis of Real-Time Embedded Systems. Version 1.1, formal/2011-06-02, 2011.

[OS06] Oster, N.; Saglietti, F.: Automatic Test Data Generation by Multi-Objective Optimisation. In (Górski, J. Ed.): Proc. 25th Int. Conf. on Computer Safety, Reliability, and Security, SAFECOMP 2006, Gdanks, Poland, 2006. Vol. LNCS 4166, Springer-Verlag, 2006; pp. 426-438.

[SH05] Störrle, H.; Hausmann, J. H.: Towards a Formal Semantics of UML 2.0 Activities. In (Liggesmeyer, P.; Pohl, K.; Goedicke, M. Eds.): Proc. Software Engineering 2005, Essen, Germany, 2005. Vol. LNI 64, Köllen Verlag, 2005; pp. 117-128.

[SP10] Saglietti, F.; Pinte, F.: Automated Unit and Integration Testing for Component-based Software Systems. In: Proc. Workshop on Dependability and Security for Resource Constrained Embedded Systems (D&S4RCES'10), Vienna, Austria, 2010. ACM Digital Library, 2010.

[Sö10] Söhnlein, S. et al.: Software Reliability Assessment based on the Evaluation of Operational Experience. In (Müller-Clostermann, B.; Echtle, K.; Rathgeb, E. P. Eds.): Proc. 15th International GI/ITG Conference on Measurement, Modelling, and Evaluation of Computing Systems and Dependability in Fault Tolerance, MMB&DFT 2010, Essen, Germany, 2010. Vol. LNCS 5987, Springer-Verlag, 2010; pp. 24-38.

[WK09] Westergaard, M.; Kristensen, L. M.: The Access/CPN Framework: A Tool for Interacting with the CPN Tools Simulator. In (Franceschinis, G.; Wolf, K. Eds.): Proc. 30th Int. Conf. on Applications and Theory of Petri Nets, PETRI NETS 2009, Paris, France, 2009. Vol. LNCS 5606, Springer-Verlag, 2009; pp. 313-322.

[ZH00] Zhu, H.; He, X.: A Theory of Testing High Level Petri Nets. In (Feng, Y.; Notkin, D.; Gaudel, M.-C. Eds.): Proc. 16th Int. Conf. on Software - Theory and Practice, IFIP World Computer Congress 2000, Beijing, China, 2000. Publishing House of Electronics Industry, 2000; pp. 443-450.

[ZH02] Zhu, H.; He, X.: A Methodology of Testing High-Level Petri Nets. In: Information and Software Technology, Vol. 44, No. 8. Elsevier, 2002; pp. 473-489.

Hierarchical Self-repair in Heterogenous Multi-core Systems by Means of a Software-based Reconfiguration

Sebastian Müller, Mario Schölzel, Heinrich Theodor Vierhaus

Institut für Informatik
BTU-Cottbus
Walther-Pauer-Str. 2
03046 Cottbus
{smuelle2 |mas |htv}@informatik.tu-cottbus.de

Abstract: This paper deals with the problem of a software-based self-repair in a statically scheduled multi-core system in the presence of permanent faults. The basic idea is to adapt the application in a way that the use of faulty components is avoided. This goal is achieved by re-compiling the program-code via an off-line repair process in the field. The repair process is organized in a hierarchical manner. At the beginning a local repair is applied considering only one core. If the local repair fails, a retry at a higher system-level is performed. For that purpose, the local repair techniques are re-used in a global context. The repair at a global system level results in some specific system constraints and properties, which are investigated in this work. Due to the use of pure software-based methods one gains the possibility to repair defects in different components (even multi errors) or defects in spare components. The presented approach is not bounded to a concrete architecture and is therefore adaptable to systems like MPSoCs or NoCs, if these systems provide some basic functionality.

1 Introduction

The on-going downscaling of the feature-size for integrated circuits (currently 32nm) provides the possibility to combine a higher amount of transistors with a decreased switching time and a decreased energy consumption. But these positive effects also come along with negative ones, e.g. a higher vulnerability to faults and early life failures due to wear-out effects. The reasons are, on the one hand, higher stress density [BGM04] and, on the other hand, deviations in the production [MG04]. To lower the production costs and to guarantee certain emergency properties it is necessary to develop fault tolerant systems, which can cope with such permanent faults. Due to the possibility of integrating an increasing amount of transistor in an steadily decreasing chip area, we can observe the trend to multi-core systems [OH05]. To design such systems as fault tolerant, it can be helpful to combine techniques, which were developed isolated, in a hierarchical approach.

Statically scheduled VLIW-processors (very long instruction word) are easily scalable and therefore well adaptable to different requirements of certain applications. In addition, they provide a feasible degree of performance, due to the super-scalar architecture, for signal-

and video-processing in an automotive or avionic environment. A crucial advantage of a VLIW-architecture is the small controlling logic. In contrast, the control logic in dynamically scheduled processors makes up a major portion of the overall gate count. These dynamic scheduled processors are therefore hardly to repair.

2 Related Work

In the past decades, several hardware-based as well as software-based self-repair approaches have been proposed for processor based systems. An established approach is the re-configuration of a FPGA in the presence of permanent defects [MHS+04]. In [MHS+04] the re-configuration is done by a micro-controller which is implemented in a second FPGA. The disadvantage of this approach is the increase of the necessary memory size (configuration data) and the fact that an FPGA-implementation, in contrast to an ASIC-implementation, tends to have a higher consumption of area and energy.

To replace logic blocks in non-FPGA systems it is possible to use switches, which can shutdown blocks and activates spare-blocks [KSV09]. Such an approach is only feasibly for large logic blocks, since the administration of the repair requires additional hardware which is error prone itself. To gain an actual improvement of such a fault-tolerant system, it is necessary that the portion of the administrative logic is relatively small compared with the overall chip area [KV11].

There exist other approaches, which focus on a software-based self-repair. In [KKP00] it is proposed to calculate all necessary schedules for every fault situation in advance. These schedules are stored in the memory and the appropriate schedule is executed at run-time, according to the detected faults. The obvious disadvantage of this method is the extra cost in memory for storing all these additional schedules.

Another approach is presented in [MS08]. It is proposed to implement most of the hardware operations in a software routine. If an operator is defective, its function is assumed by the associated software implementation. In case of an error the performance degradation might be substantial due the overhead of the implementation in software.

The presented solutions so far are all bounded to the repair of at least one single core or only a few components in a single core. But just with the upcoming of multi-core systems further techniques for handling permanent faults in such an environment have been proposed. In [ARJS07] and [RS08] similar approaches are presented. The idea is that a defective core can borrow resources from other cores in the system; e.g. a core $C1$ can use the execution stage of a different core $C2$ to finish the execution of one of its operations. The major disadvantage of this approach can be found in the administration. For this purpose it is necessary to extend the control logic and to enlarge the connection between the cores.

There exist further publications, in which the repair of permanent faults is done with the help of virtualization in homogenous multi-core systems [Jos06, CASP10]. In [Jos06] a hypervisor is introduced between the hardware and the operating system. The hypervisor can e.g. emulate operators in software, and it can shift threads from a faulty core to

a non-fault one. In [CASP10] an additional layer is introduced, which maps defective components of a certain core to a fault-free core.

All these methods are bounded to homogenous multi-core processors. Moreover, all the presented methods have a low system performance in case that an error is present. This performance degradation arises from the virtualization and therefore an additional runtime due to the necessary software layers.

3 Description of the Basic System

The multi-core system, depicted in fig. 1, consists of several VLIW-Processors, the necessary program memories, one common used data memory, and a connection network. All components are clocked synchronous via a global clock. The processors can be heterogeneous, if they belong to a family of processors. To every processor C_i exists one program memory P_i. The cores are not connected directly to their ROM. Instead, they are connected to the connection network, which implements the external interface for every processor. Via the connection network every access to the program and data memories is handled as well as the communication between the cores. The controller of the connection network guarantees a mutual exclusion regarding the access to resources of the system.

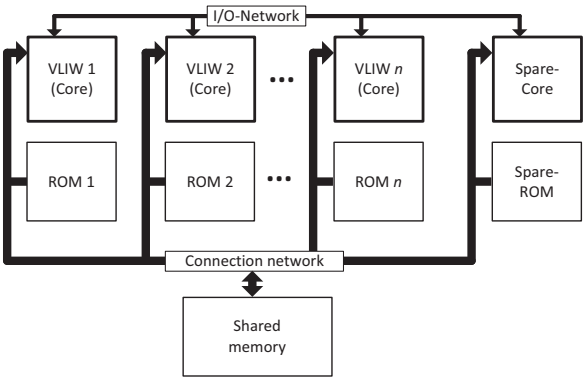


Figure 1: Scalable MPSoC with several cores, ROMs and one shared memory

A single VLIW-core has no information about the global design of the system. Every core addresses a local program memory and a common used data memory, which seems to be exclusive for every core. Furthermore, there are I/O-ports available for every core to communicate with external components. The cores are implemented in a VLIW-architecture. The advantage of such an architecture is its scalability. With such architecture a varying range of several heterogeneous systems can be configured in an easy manner. A VLIW-core consists typically of multiple execution slots. The calculation in one slot is done

by an FU (functional unit), whereas those can perform different operations. A statically scheduled instruction word controls in every cycle, which operation is calculated in which slot. There is no dynamic scheduling within the processor. The scheduling is done at the compile time of the system. The given processor has a 4-stage pipeline (FE, DE, EXE and WB) as well as a bypass network to avoid hazards.

In the program memory of every core exists a program for a self-test. The software-based self-test is executed at the system start-up or during short breaks of the system runtime. Even the spare cores executes this self-test. The results of the self-test are stored in a dedicated part of the data memory. In the presence of any error an appropriate self-repair routine is executed. The self-repair is software-based and can cover local repair techniques as well as global ones. The major goal of the repair is to re-configure the application in a way that the use of faulty components is avoided.

4 Local Software-based Self-repair

This section presents two repairing techniques which can be applied as a local repair in a VLIW processor. The first method implements a re-binding, whereas the second one executes a re-scheduling based on entire basic blocks. Both techniques re-configure the application to avoid the use of defective components. This adaption is done as an off-line repair, and the repair process itself is a pure software-based solution.

The software-based re-binding, developed in [Sch09, SM10], binds the operations of an instruction word to a different hardware resource in the case that the original binding plans the use of a defective component. The algorithm iterates over the program memory and transfers in every step initially one instruction into the data memory. After that, every operation is bound to an unused non-defective resource. Then the new created instruction word replaces the old instruction in the program memory. The software-based re-binding is a fast technique in respect to the execution time, which can only handle single cycle operations.

A more powerful technique is called Scoreboarding, which can handle multi-errors for any number of components. The method is described in detail in [SM10]. The basic idea is the following. The program code of an application is re-compiled stepwise at the granularity of basic blocks. The algorithm transfers at the beginning an entire basic block from the program memory into the data memory. The machine code is then inserted into a priority list. Based on the priority list, a re-scheduling is executed, which generates a new schedule for the basic block. For re-scheduling a simple list-scheduling algorithm is used. The schedule is converted into machine code and written back into the program memory, where it replaces the old machine code. The Scoreboarding algorithm takes into account that several components can be faulty. Due to that a fine grained repair of the system is possible with this technique. The repairable defects can be located in different system components like single operators, FUs, register of the pipeline, regular registers or read- and write-ports. The presented method is a pure software-based technique, which is executed at the system start-up.

5 Global Self-repair

If a repair at the local level fails, the next objective is to apply a global repair strategy. A local repair can fail, if there are too many defects present or if a critical component (e.g. all components of the control path) fails. This can lead to severe problems during the repair process. A first scenario could be that the repair algorithm is not able to determine a valid schedule. In another scenario, it is possible that the software-based self-repair itself is not correctly executable. This problem can arise if the repair algorithm uses a defective component (e.g. a faulty branch unit). In the first case, a spare core C_s shall overtake the task of the defective core. In the latter case it is sufficient to execute an appropriate repair algorithm, which does not use any defective component. This can be achieved by e.g. repairing the repair algorithm itself.

Case 1: Compilation to a spare core
Figure 2 shows the corresponding process for the global repair strategy. In this example it is assumed that core C_x is faulty and its task shall be overtaken by core C_s. To achieve this it is necessary to transfer all relevant parts of the program memory of C_x into the memory of C_s. Meanwhile it is possible to replace the program code of C_x with nop instructions, so that C_x is denied from any writing access to system resources. Due to the fact that C_x might differ in its architecture from C_s, it can be necessary to adapt the program code to the architecture of C_s. For that the scheduling method, which was described in the last chapter, will be applied. With the help of the Scoreboarding algorithm the application of C_x is re-compiled to the architecture of C_s, whereas the algorithm consideres parameters like FU amount and operator configuration. After completing the re-compilation of the application the repair process is done and the regular system execution can be started.

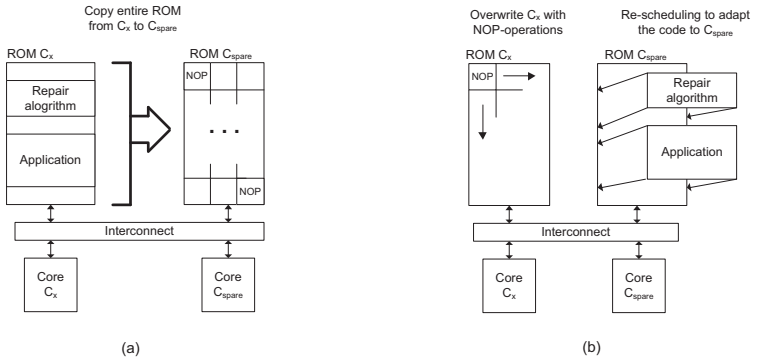


Figure 2: (a) Transferring the program code from C_x into the ROM of C_s; (b) Overwriting the ROM of C_x with *NOP*-Instructions and adapting the program code to C_s

Case 2: Repair of the Repair Algorithm
This section presents two strategies of how a repair algorithm can be adapted to a certain defect situation. The first method adapts the program code of the repair algorithm at the

compile time of the system. In comparison, a second method will execute the adaption in the field as a off-line repair procedure.

In the first approach the decision, which version of a repair algorithm has to be executed, is based on differently compiled versions of this algorithm. At system compile time, several versions v_i of the same algorithm are generated. Every version v_i uses only operators of an corresponding FU f_i. As long as there exist on fault-free FU f_j in the datapath, it also exists a working version v_j of the repair algorithm. An external core C_e can determine at repair time an appropriate algorithm version, based on the results of the self-test. After determining a version v, it is necessary to start the execution of v on the defective core. To do so C_e manipulates a jump instruction in a way that the target of the instruction is the start address of v. Figure 3 shows the procedure of launching an appropriate algorithm version.

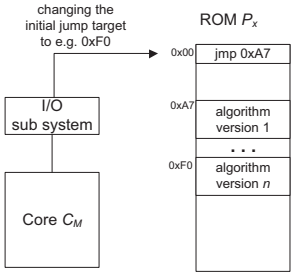


Figure 3: Launching an appropriate algorithm version by changing a jump address

In the second alternative, we propose to adapt the repair algorithm in the field in an off-line repair process. The advantage is that no additional schedules have to be stored in the memory. This approach is especially effective, if defects in the regular registers or the read- and write-ports of the register file can occur. In such a case, it is not practicable to calculate several schedules in advance for all possible combinations of faults. For that purpose it is more feasible to adapt the local repair algorithm in the field to the current fault situation. Thereby the procedure is quit similar to the procedure in the first alternative. The difference is that no spare core is necessary, and instead only the repair algorithm in the ROM of a defective core C_x is changed. The residue application of C_x is self-repaired by C_x after its repair algorithm is re-compiled. To support this repair strategy, it is helpful to compile the local repair algorithm only with a few or even with no parallelism at instruction word level. This can be done by the compiler at the compile time of the system.

6 Extension of the System

The proposed repair strategy has some consequences, which affect the system architecture. The first issue to mention is that it is necessary to organize and administrate the repair in an appropriate manner. A next important point is that the requirements concerning the

connection network has changed; e.g. access to the program memories. Other points are the extensions regarding the spare core and creating necessary meta-data for supporting the repair. These meta-data can be generated and gathered during the compile time of the system.

6.1 Organisation and Administration of the Repair

The system start-up is divided into two phases. At the end of a phase, every core has to be in a valid state. In the first phase, every core (even spare cores) execute a software-based self-test followed by a local software-based self-repair if necessary. At the end of every sub phase, the result (e.g. fault detected, repair successful) is written into the shared memory. For coordinating the organisation and administration of the repair, a designated core of the system (referenced as C_M) is determined. The core which finishes first successfully the first phase of the system start-up declares itself as master core C_M. After that C_M informs the other cores about this fact via the shared program variable mID. The evaluation of all of the results generated in the first phase is done by C_M. The possible situations are the following:

- Case 1: No defects respectivle successfully repaired
- Case 2: One or more cores are faulty but not repaired yet (Repair failed or did not stop)
- Case 3: One or more cores are without result (Self-test did not stop)

After determining the master C_M, the second phase is started. First of all C_M waits until the necessary execution time of the several self-tests is over. Thereafter C_M knows if a local self-repair is executed and waits for the end of it as well. Now C_M can determine, which of the latter described cases is present. If case one is present, the regular system execution can be started. The other two cases will be handled as follows:

Case 2:
In this case a faulty core C_x could successfully determine its fault situation. However, the local repair was not able to generate a valid schedule. C_M determines, based on the results in the shared memory, that the execution of the self-test was finished successfully, but that the self-repair failed. There are two reasons why a local self-repair might fail:

- 1. If the local self-repair algorithm uses defective components it can not be executed correctly. Therefore the repair algorithm might not stop (e.g. defective branch unit) or it calculates wrong results (e.g. defect in a register).
- 2. Due to many faults or a lack of resources, it is not possible to determine a valid schedule. A possible scenario might be that all operators of a certain type are faulty. In certain circumstances it is also thinkable that undetected errors influence the repair algorithm.

C_M can in the first case adapt the local repair algorithm with the strategy described in case 2 of the global repair. After that C_x could re-executed the local repair algorithm with the appropriate algorithm. Alternatively C_M can execute the local repair and re-schedule the application of C_x. As already mentioned, the second strategy is useful, if registers are faulty.

In the second case, core C_x has to be replaced by a spare core due to many or critical faults. It is assumed that undetected errors can occur. Therefore an external software-based test is executed to re-investigate C_x for undetected errors before a spare core is used in the end. If any differences between the results are identified, the self-repair is re-executed with the new information.

The last thing to mention is how C_M can decide whether a local repair algorithm uses a faulty component or not. It would normally imply that every instruction word has to be checked for a certain binding to a faulty component. As solution it is proposed that the compiler (at system compile time) compiles the algorithm in a way that only the first slot is used. If a fault in the first slot is present, C_M now knows that the local repair algorithm is not executable. A similar agreement can be made regarding the registers by declaring a fixed range of register as usable.

Case 3:
In this case, no result of the self-test of core C_x is present. C_M can check this by evaluating the shared memory. In this scenario C_M externally re-tests C_x. After that C_x will be repaired with respect to the detected faults. This can be done externally or locally. It is marked that C_x has been successfully tested. If now a repair fails, the situation changes into the second case, which was discussed before.

6.2 Requirements for the Connection Network

The access to the program memories is managed by the connection network. Every core can communicate with the connection network via certain I/O-ports. Three different ports are necessary. The first port specifies the address in the program memory, the second port declares the address in the data memory, and the third port specifies the required command word. In the command word is the access mode (read or write) and the target core coded. The connection network ensures a mutual exclusion during the access to the program memories. Every core, which requests a further access during a granted access, will be set into a stall-mode.

During system start-up no write access to the program memories will be granted by the network. Therefore it is avoided that a faulty core accidently writes to system resources. At the global system repair phase, write access has to be explicitly activated by sending a certain command via an I/O-port to the controller of the connection network.

In the last section it was described how the core C_M is determined. After C_M has declared it self as master, this fact is communicated to the other cores via the global variable mID. The access to this variable has to take place exclusively. Therefore the system is extended with a global instrument for synchronisation. To do so, a dedicated port p_m is declared,

which has to be accessed before an access to mID is issued. The synchronisation via port p_m is managed by the controller of the connection network. After an access to p_m, no further access will be granted for the next 10 clock cycles. If the controller registers a further access within this 10 cycles, the corresponding core will be stalled.

6.3 Metadata for the Administration

For coordinating the repair, it is necessary to provide some information about the system to the core C_M. These data are stored in a data structure, to which every core has access during the runtime of the system. The necessary data are the amount and id of the cores, the runtime of the self-tests, start addresses and size of the repair algorithm, memory address of the self-test results, and addresses of the jump instructions into the local self-repair routines. The determination of the metadata takes place at the compile time of the system. This task can be done automated by the compiler. The runtimes of the local self-tests can be determined by simulation and profiling of the algorithm. At the end, all data are gathered in one data structure. This data structure is stored in the shared memory.

7 Results

The presented multi-core system has been implemented in VHDL (and synthesized) and, furthermore, a scalable multi-core simulation environment has been developed in C++. The simulator emulates for every VLIW-core in the MPSoC one instruction set simulator. The proposed software-based self-repair algorithms have been developed in assembler, translated into binary code, and tested on the VDHL-model and in the simulation environment.

7.1 Synthese Results

Two VHDL-Implementations have been developed. The first implementation is the non fault-tolerant system with four cores. In a next step this system was extended with a spare core and the necessary functionality for our fault-tolerant strategies. Both systems were synthesized using a 45nm library. The delay on the longest critical path amounts to 2.9 ns. Therefore the resulting system clock is about 340 MHz. Table 1 shows additional results of synthesise for some selected components and its area dimensions. The major portion of the overall chip area belongs to the data path, whereas the control path makes up only a few percent. This is the major advantage of such a VLIW-Architecture. The probability that an error in the control path occurs, is decisively lower compared to the occurrence of an error in the data path. Furthermore our repair strategy concentrates on components like the FUs, registers and ports, which define the major part of the data path. The necessary extensions of the connection network which are crucial for our software-based repair are,

compared to the overall chip area, relatively small with only 2.8 percent. This is an another advantage in contrast to a fault-tolerant hardware solution.

Table 1: Synthese results (relative area dimensions) of the original system (4 cores) and the fault-tolerant system ($4 + 1$ cores) - values in Nand2-aquivalents

Component	Original System	Fault-tolerant System
MPSoC overall	92240	118510
Connection network	1274	2844
1 core	22869	
1 slot	965	
1 FU	748	
Register file	14835	
Control path	270	

7.2 Runtime of the Software-based Re-binding

The following scenario (Case 2 of the global repair) presents the necessary execution time of a global repair strategy if the local repair algorithm is adapted to the current fault situation. The program code of a core C_x is adapted by the help of a software-based Re-binding executed on core C_M. The code size, which has to be repaired, amounts to about 500 instruction words. The Re-binding algorithm reads in every step one instruction word from the program memory, disassembles it, creates a new binding, assembles it, and overwrites the old instruction word in the program memory with the newly created one. The necessary overall execution time for repairing a single instruction word is 306 cycles. To adapt the Scoreboarding algorithm (around 500 instruction words), the repair will need 0.72 ms of execution time in respect to a system clock of 340 MHz.

7.3 System Reliability

Based on the synthesis results the overall system reliablity of the original system and the fault-tolerant system has been determined. Figure 4 compares the reliabilities of both systems against each other. The reliability was calculated in respect to a constant rate of failure. For the calculation of the reliability of the original system, the reliabilities of the four cores and the non-fault tolerant connection network have been used. For the fault-tolerant system, the reliabilities of the spare core and of the extended connection network were used. The fault-tolerant system is modelled as 4-out-of-5 system due to the fact that a failing of an entire core can be tolerated. Figure 4 shows that the system with a software-based repair (Rft) provides a higher system reliability then the original system.

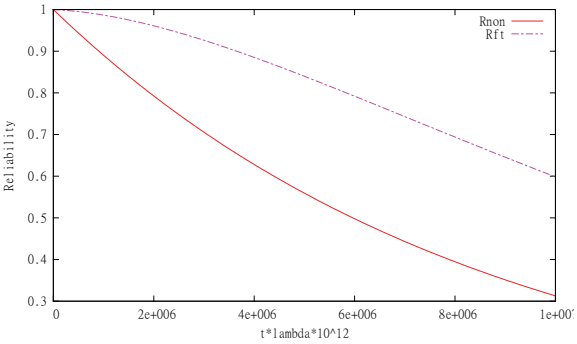


Figure 4: Overall system reliability of the original system (Rnon) and the fault-tolerant system (Rft)

8 Summary

We presented a fault-tolerant heterogeneous multi-core system. The applied repair strategies are organized in a hierarchical manner, and the main focus lays on a software-based self-repair. The proposed method abstracts from a certain architecture and can be employed on a varying set of platforms (MPSoCs, NoCs). A prototypical system has been implemented and synthesized in VHDL. The results of the synthesis process were used to determine the reliability of different system configurations. Further investigations have been done with a scalable simulation environment.

The main advantage of combining different repair techniques in one hierarchical strategy is that a local fine grained repair is possible as well as a repair at system level. A system stays functional even if critical components of a core are faulty. Due to the hierarchical approach the amount of critical components, which causes a total breakdown of a multi-core system, is further reduced.

References

[ARJS07] Nidhi Aggarwal, Parthasarathy Ranganathan, Norman P. Jouppi, and James E. Smith. Configurable isolation: building high availability systems with commodity multi-core processors. In *ISCA '07: Proceedings of the 34th annual international symposium on Computer architecture*, pages 470–481, New York, NY, USA, 2007. ACM.

[BGM04] Melvin A. Breuer, Sandeep K. Gupta, and T.M. Mak. Defect and Error Tolerance in the Presence of Massive Numbers of Defects. *IEEE Design and Test of Computers*, 21:216–227, 2004.

[CASP10] Kavitha Chandrasekar, Revathi Ananthachari, Sangeetha Seshadri, and Ranjani Parthasarathi. Fault Tolerance in OpenSPARC Multicore Architecture Using Core Virtualization. 2010.

[Jos06] Russ Joseph. Exploring salvage techniques for multi-core architectures. In *In Proceedings of the 2nd Workshop on High Performance Computing Reliability Issues*, 2006.

[KKP00] Ramesh Karri, Kyosun Kim, and Miodrag Potkonjak. Computer Aided Design of Fault-Tolerant Application Specific Programmable Processors. *IEEE Transactions on Computers*, 49:1272–1284, 2000.

[KSV09] T. Koal, D. Scheit, and H. T. Vierhaus. A scheme of logic self repair including local interconnects. *Design and Diagnostics of Electronic Circuits and Systems*, 0:8–11, 2009.

[KV11] T. Koal and H.T. Vierhaus. Optimal spare utilization for reliability and mean lifetime improvement of logic built-in self-repair. In *Design and Diagnostics of Electronic Circuits Systems (DDECS), 2011 IEEE 14th International Symposium on*, pages 219 –224, april 2011.

[MG04] Mahim Mishra and Seth C. Goldstein. Defect tolerance at the end of the roadmap. pages 73–108, 2004.

[MHS+04] Subhasish Mitra, Wei-Je Huang, Nirmal R. Saxena, Shu-Yi Yu, and Edward J. McCluskey. Reconfigurable Architecture for Autonomous Self-Repair. *IEEE Design and Test of Computers*, 21:228–240, 2004.

[MS08] Albert Meixner and Daniel J. Sorin. Detouring: Translating Software to Circumvent Hard Faults in Simple Cores. In *38th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, June 2008.

[OH05] Kunle Olukotun and Lance Hammond. The Future of Microprocessors. *Queue*, 3(7):26–29, 2005.

[RS08] Bogdan F. Romanescu and Daniel J. Sorin. Core cannibalization architecture: improving lifetime chip performance for multicore processors in the presence of hard faults. In *PACT '08: Proceedings of the 17th international conference on Parallel architectures and compilation techniques*, pages 43–51, New York, NY, USA, 2008. ACM.

[Sch09] Mario Schölzel. A Delay Estimation of Rescheduling Schemes for Static Scheduled Processor Architectures. *22th International Conference on Architecture of Computing Systems*, 2009.

[SM10] Mario Schölzel and Sebastian Müller. Combining Hardware- and Software-Based Self-Repair Methods for Statically Scheduled Data Paths. *Defect and Fault-Tolerance in VLSI Systems, IEEE International Symposium on*, 0:90–98, 2010.

Enhanced Reliability in Tiled Manycore Architectures through Transparent Task Relocation

Holm Rauchfuss, Thomas Wild, Andreas Herkersdorf

Technische Universität München
Lehrstuhl für Integrierte Systeme
Arcisstrasse 21
80290 München, Germany
holm.rauchfuss@tum.de
thomas.wild@tum.de
herkersdorf@tum.de

Abstract: Manycore platforms with tens and even up to hundreds of processing cores per chip are becoming a commercial reality and are subject of intensified research. This concept paper describes work in progress on the applicability of HW supported communication and processing virtualization on regular structured, tiled manycore architectures for the benefit of improved fault tolerance against transient and permanent perturbations. Temporarily unused, naturally redundant tiles are dynamically occupied during run time via transparent task relocation. This means, the execution of a task can pro-actively and transparently for the application be switched by distributed system management and virtualization services from a tile, which is considered unreliable, to a more reliable tile. In order to support different requirements regarding safety, timing integrity and minimized overhead for the relocation services, several established strategies can be enacted by the system management. The migration protocol for signaling during run configuration and actual relocation allows migration with minimal downtime and no communication loss. The actual migration is triggered by a configurable threshold on critical system parameters on a per task basis.

1 Introduction

The growing transistor integration densities following Moore's Law enabled manycore platforms with tens and even up to hundreds of processing cores per chip. So-called tiled architectures, where compute, IO and memory resources are structured in individual tiles interconnected by a packet-based Network-on-Chip (NoC), are a particularly promising set-up for scalable manycores (see Fig. 1 for a generic example of a tiled multicore). Existing implementations include Intel's "Single-Chip Cloud Computer" (SCC) [GHKR11] or the TILEPro100 platform by Tilera [Aga07], containing 24 and 100 tiles, respectively.

However, technological progress is not the only driver towards manycore architectures. The trend is to consolidate multiple applications, each consisting of several tasks with individual safety, security and real-time requirements onto a single shared processing platform.

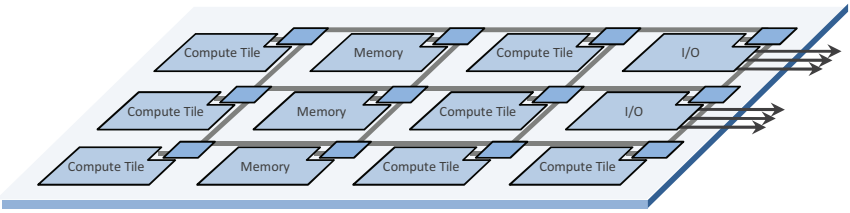


Figure 1: Generic tiled manycore architecture

Examples are sensor or media processing applications, augmented reality and robotic control or other recognition, mining and synthesis (RMS) applications [Dub05], all running on the same multi-/manycore processing platform. Those applications have in common that they require high performance and parallel execution.

The aforementioned platforms are subject to dependability issues as they favor latest technology implementation and 3D integration with increased potential error rates, either transient or permanent [B+04]. One example is stress by thermal hotspots resulting in intermittent errors in signal integrity or run time behavior (short term effects) as well as in physical damages in form of transistor aging or even electromigration [NX06]. To guarantee the operativeness of the applications, counter measures must be taken at all abstraction levels of integrated circuit and system design.

In this concept paper, we propose to enhance regular structured manycore architectures with HW supported processing and communication virtualization techniques in order to increase the fault tolerance of the applications. This is centered around the dynamic re-use of temporarily unused or weakly loaded tiles by transparent task relocation. The execution of a task is switched over from a tile marked as unreliable to a more reliable tile as a service by the underlying distributed system management. In order to support different requirements regarding safety, timing integrity and minimal relocation overhead, several established strategies can be enacted by the system management. tiles by

The remainder of the paper is structured as follows: In chapter 2 the platform with its basic building blocks and supporting extensions for virtualization is described. Chapter 3 explains the different error and failure concepts on this platform and the protocols to signal migration, triggered by task specific thresholds violations. The paper concludes with a summary in chapter 4.

2 Tiled Manycore Platform

2.1 Basic Building Blocks

Fig. 2 depicts a common tiled manycore platform constructed from a limited set of building blocks.

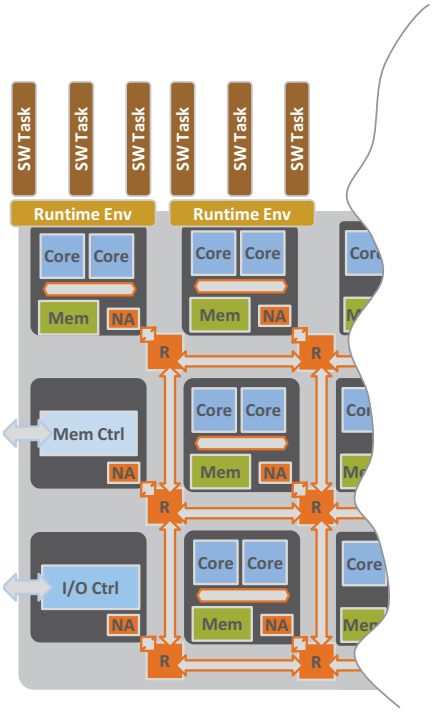


Figure 2: Tiled manycore architecture in detail

Processing is performed in compute tiles, consisting in general of one or few small RISC cores and local memory connected via a shared bus. Due to resource constraints and the overall design decisions compute tiles only have small feature set. To avoid complex cache coherence algorithms and synchronization, the basic runtime environment usually spans only individual compute tiles. It provides SW tasks running on top of it with limited services.

A NoC with router nodes and links acts as a generic communication infrastructure to connect individual tiles. Every tile has at least one network adapter (NA) providing access to the NoC. All interaction and data exchange between tiles is done via explicit communication, i.e. message passing, over the NoC. In order to prevent message loss or blockage in case of broken NoC routers or links, the NoC has to flexibly provision alternative routes.

IO, e.g. network interfaces or high-speed links, is encapsulated in own tiles and can is shared via the NoC by a large number of compute tiles. Memory resources are either on-chip, dedicated SRAM based memory tiles, or for larger memory ranges, off-chip DDR SDRAM modules accessible through memory controllers. Local memories within the compute tiles, on-chip SRAM tiles and off-chip memory form the memory hierarchy of the manycore.

The regular structure of such an architecture and the high number of available homoge-

neous (compute) tiles enables approaches to enhance the reliability by using (temporarily) unused tiles as fallback alternatives in case of failures and errors within the designated tiles. In general, limiting factor is the static configuration/deployment of the architecture and (up to now) missing methods and strategies to migrate tasks in a transparent way, i.e. without modifications and side-effects for the tasks and their runtime environment.

2.2 Extensions for Virtualization

As tiled manycore architecture provide a consolidating environment for hundreds or even thousands of tasks, those can have different requirements towards their running environments in terms of real-time, required operating system and performance/throughput. The system needs to be provisioned, scheduled and partitioned for those concurrently running tasks and environments.

Virtualization of the physical HW by a hypervisor or virtual machine monitor (VMM) is an established concept for providing such a function ([BDF+03], [Hei09]). Different domains are running in separate virtual machines (VM). A domain can include a single task with only a rudimentary operating system up to a complex general operating system with several tasks. The underlying platform resources are shared and compartmentalized by the hypervisor.

To avoid a single point of failure a hypervisor instance should be only controlling one or a few cores ([lin]) and communicate/coordinate with the the other hypervisor instances via a distributed system management and their local slaves.

To reduce the overhead for processing virtualization and enhance compartmentalization HW support for this is preferred. Similar approaches exist already for High Performance Computing (HPC), e.g. VT-X on Intel CPUs [NSL+06] and can be scaled down for cores in compute tiles. The explicit communication between compute, memory and IO tiles is highly performance critical. Here, dedicated HW support is also required to eliminate overhead otherwise occurring in software to provide virtualization for communication. To avoid changes to the NoC and and enable re-use of existing implementation this HW support should be added to the edges of it, i.e. within the network adapters in the tiles. As NoC is is packet-based concepts from the HPC network and IO virtualization can be adapted for this ([WSC+07]). IO tiles share the most resemblance to normal virtualized network interfaces and therefore have an extensive virtual network interface controller (VNIC). Virtual Network Adapter (VNA) for the compute tiles are a subset of the features of VNIC as their have a only a limited number of connections to share and to service.

Fig. 3 highlights the differences between a common tiled manycore architecture and one with virtualization extensions described above.

In order to achieve a performing and cost efficient realization for the VNIC/VNA entities, NoC packet queues, caches for individual communication configurations and packet buffer management should not be stored entirely in the respective entity. Although this would be advantageous from throughout and real-time aspects it essentially duplicates memory requirements. With the assumption that only a limited number of real-time and high-

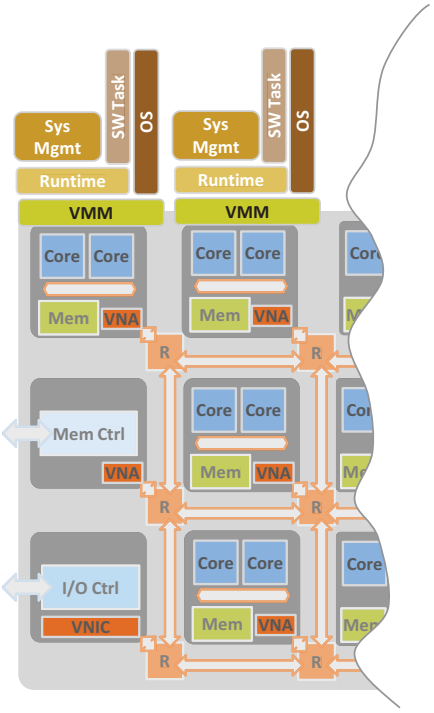


Figure 3: Tiled Manycore Architecture with virtualization extensions

priority connections are active at any point of time, it should be possible that a bounded number of queues/caches/buffers can be shared between those connections in a dynamic way [RWH10]. By managing those elements for real-time/high-priority tasks and for best-effort tasks by scheduling and (de-)multiplexing the associated resources, different levels of services can be provided (see Fig. 4.

Because of the incooperation of communication and processing virtualization it is now possible have transparent task relocation. Furthermore, hypervisors with their small footprint provide a minimal trusted computing base and can be used to enhance overall reliability even further via monitoring of runtime environments and their tasks [DKR08], via check-pointing complete runtime environments or via loose-lock-stepping ([TSKM08], [CLO+08]). This can be used to provide flexible (dual) modular redundancy for cores in intra and inter tile scope but without requiring special or dedicated HW. Drawback is a higher overhead due to handling it mostly in SW via hypervisor.

Results reported in the literature ([CCS10], [JRK10]) show only minimal downtimes during live migration of complete operating systems in HPC environment. We expect that for tightly integrated tiled manycores architecture with HW enablement for processing and communication virtualization with only small runtime environments live migration without communication data loss under real-time constraints should be possible.

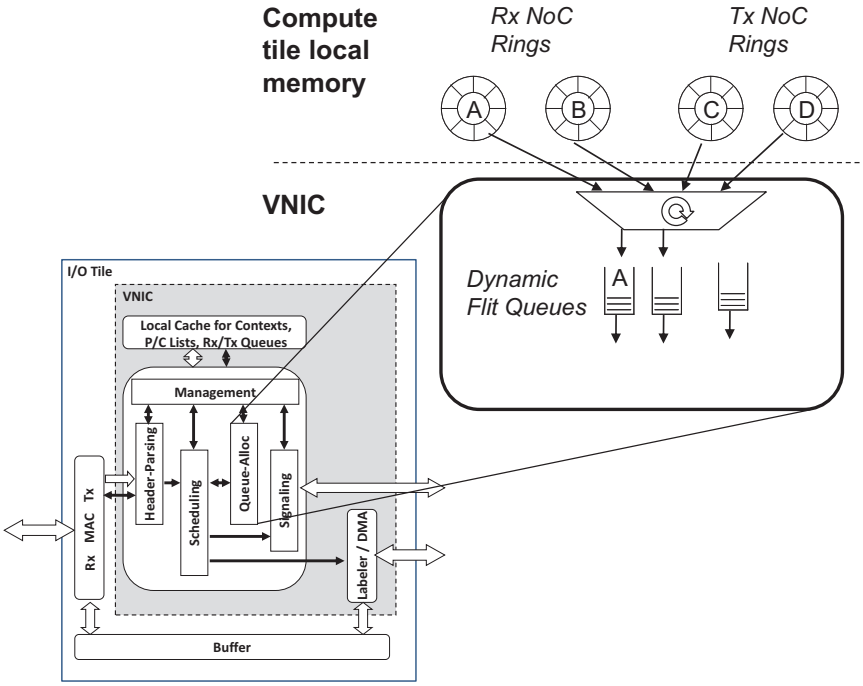


Figure 4: VNIC with dynamically shared queues for HRT and Best Effort communications

3 Error/Failure and Migration Handling

3.1 Classification

To decide if and when a task relocation is needed the components required for this task, e.g. runtime system, compute tile, cores, NoC or IO, have to be categorized by their operativeness. The following states are possible:

1. **No Error, No Failure**: The component is operating normally.
2. **Some Errors, No Failure**: The component is not operating error-free, but there is no failure yet. Errors include parameters exceeding a defined threshold, e.g. a too high temperature in a tile.
3. **Significant Errors, No Failure**: The component experiences errors over a defined rate, but this still does not result in a failure.
4. **Failure**: There is a failure and the component's behavior is corrupted.

The first three states can be handled by task relocation without information loss. The last state requires additional mechanisms to return to a valid state. According to varying

safety criticality and timeliness requirements, task relocation can be divided into different strategies based on existing protection methods [VPD04].

- **Cold Task Relocation**: No alternative set-up has been configured. The system management has to create this alternative from scratch. This involves reserving compute tile resources and NoC routes and activating them. This approach does not bind additional resources beforehand, but requires the most actions straight before and during the relocation. See Fig. 5a) for an example involving tasks on a compute tile processing IO data coming from an IO tile and and storing data on a memory tile.
- **Warm Task Relocation (1:1, N:1)**: An alternative set-up has been pre-configured. The basic system and task are existing, but not running. The state of the task has be transferred and the communication routes have to be activated. Overcommitting the alternative compute tile for different tasks is possible. This strategy should provide a good trade-off between increased reliability and utilized resources. See Fig. 5b) for an example.
- **Hot Task Relocation (1+1)**: An alternative location has been pre-configured and is running. All communication is already up both for the original and alternative location. In case of a relocation only the active setting is switched over. In this variant the task itself can be allowed to trigger its relocation and inform the system management afterwards to eliminate it from the critical decision path. Here, low latency, minimal downtime and high fault tolerance is achieved with high resource utilization. See Fig. 5c) for an example.

According to the criticality for an individual task, a particular strategy for task relocation is pre-defined during design time and potentially pre-configured by the system management during run time. This strategy is triggered when the task is under a dependability threat. Sensors within the tiled manycore architecture monitor the components and report failures and errors to the system management. This uses the aggregated information to generate a model of the system. See chapter 3.2 for the triggering and chapter 3.3 for the used protocol.

For cold and warm task relocation a chain of strategies can exist as to prior relocation of other tasks the first fallback is already impossible. Set-ups must be constructed and chosen carefully to prevent overload in the system in case of a migration.

A special case is task migration within a tile. This is only possible if this tile contains two or more cores. Such a migration is preferred due to the limited configuration effort in opposition to inter tile migration. No changes or reconfiguration other than within the tile have to be performed and no complex migration protocol is required (see below). Nevertheless, inter tile task migration is needed if the reason for the compute tile becoming unreliable are errors or failures in required components, e.g. the shared bus. Another reason is too strained resources in this tile to support a consolidation of all tasks on a reduced number of cores. Furthermore, task migration to another tile can be required because of errors and failures in the vicinity of the tile, e.g. existence of a thermal hotspot or problems with routers or network adapters nearby.

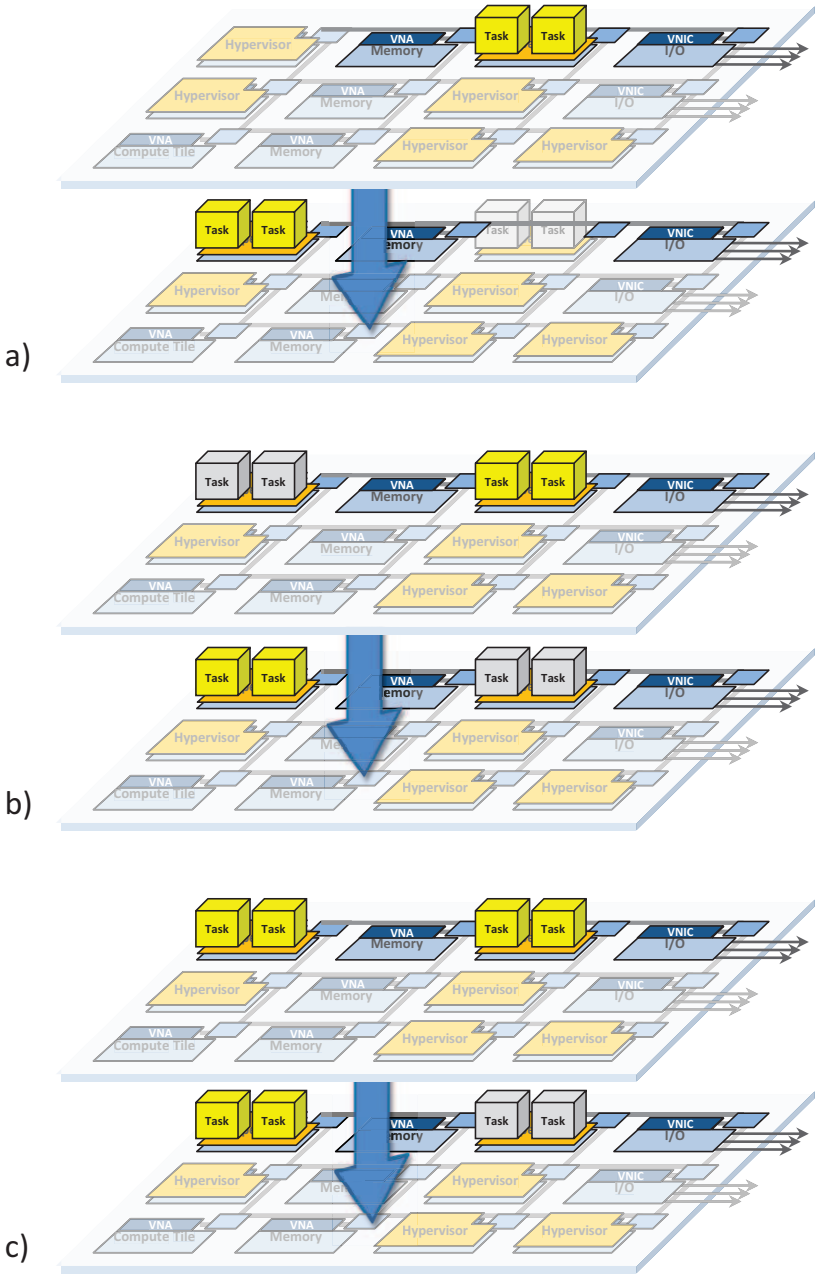


Figure 5: Task migration with different strategies a) cold task relocation, b) warm task relocation, c) hot task relocation involving two tasks on a compute tile processing IO data coming from an IO tile and and storing data on a memory tile

3.2 Threshold-based Migration

In the system management a pre-set matrix for each task exists with its sensitivity to error and failure states in a definable time slot and per individual components. Some tasks have a high sensitivity to errors and failures in certain components because this will almost certainly lead to a failure of the task, e.g. a wrong timer value for a real-time task. Other tasks are maybe more relaxed regarding errors or failures, for example small data corruption in video input data for a object recognition task can be ignored due to the noise robustness.

This matrix is weighted with factors decided during design time and used in the decision to relocate based on configurable threshold for this specific task. The criticality of the task is accounted for in the weighting, i.e. highly critical tasks have high weighting factors. The threshold is defined by the migration cost for this task. The relocation consists of two factors: The cost for the actual migration operation (the amount of to be transferred task data for this relocation between tiles in form of a memory snapshot, reconfiguration overhead) and the load increase of the system after migration in the new set-up (e.g. traffic over more NoC links). If the actual value generated by errors and failures for the weighted matrix is over the threshold the task relocation is triggered as configured by the respective system management slave. This concept is visualized in Fig. 6. This approach goes beyond Polze et. al [PTS11] which ignores the cost for migration and only triggers migration on a failure prediction.

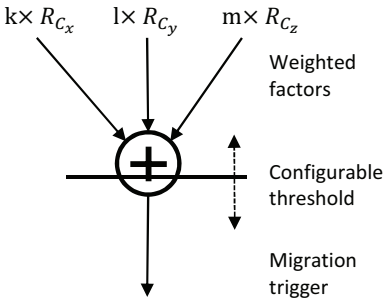


Figure 6: Visualized migration trigger calculation with configurable threshold

3.3 Protocol for Migration Management

As migration management has to act in a timely fashion we propose to integrate the critical steps of it as dedicated HW blocks and queues into the VNA and VNIC modules. The communication for the migration protocol has a higher priority and/or reserved communication resources in the NoC compared to the normal data communication. The system management can pre-configure alternative (i.e. shadow) set-ups. Such a set-up consists of a bundle of communication and processing configurations which have to exist for a task deployment. By deactivating the old set-up and activating a new set-up a task migration is

performed.

Each set-up has a global unique ID. This is used to locate the set-up (local memory in a tile or in a memory tile, for example by a VNIC). Linked to a set-up is a list of all involved tiles. Each tile has its own configuration linked to this set-up (e.g. the run time and hypervisor parameters and memory address ranges).

If triggered by a task threshold violation the responsible system management slave (SMS) initiates switching to the designated alternative set-up. The first step is writing the ID of the old set-up to be purged and the ID of the new set-up replacing it to a special configuration interface of the tile's VNA. Receiving this write the VNA locks itself against other task migration requests. It then transmits a request to VNAs and VNICs listed in the new set-up informing about the activation initiation of the new set-up.

VNAs receiving this request are locking themselves, are acknowledging it by replying with a response and are informing their respective SMS via notification. The SMS then starts reconfiguring its tile resources for this new set-up. After successful reconfiguration it then informs its VNA about it. The VNA sends then a second acknowledgment response to the triggering VNA.

VNICs receiving this request are locking themselves and are acknowledging it by replying with a response.

If all addressed VNAs and VNICs respond in a configurable time frame with all acknowledgment responses the migration protocol continue with the deactivation initiation of the old set-up by sending a request to the VNAs and VNICs listed in the old set-up. The VNAs and VNICs receiving this request are acknowledging it by replying with a response and locking themselves.

If all addressed VNAs and VNICs respond in a configurable time frame with the acknowledgment the switching is performed (including a downtime for the task). In this stage also running state information is exchanged in case of warm standby relocation.

The triggering VNA sends out a request for invalidating the old set-up. The addressed VNAs reply with an acknowledge response and inform their SMS about it which will then purge all related resources to the old set-up. For VNA which are both involved in the new and old set-up incoming messages are buffered for the time being. Otherwise communication is dropped.

VNICs involved reply with an acknowledgment response and if both active in the new and old set-up buffer communication during this downtime and drop them otherwise.

By adding an individual timer value per VNA/VNIC allows a tasteful shutdown of communication so that no communication will be dropped.

When receiving all the acknowledgment responses the triggering VNA sends out the activation for the new set-up. All VNAs and VNICs are using directly the new set-up. If communication is already incoming for a not yet activated set-up on a VNA or VNIC it is buffered until activated.

Sending out the deactivation of the old set-up and activation of the new set-up can be be done together if packet loss during the migration phase is acceptable.

Further requests for a different task migration are denied with a negative response message. Then the locking of the VNA/VNIC resource is acknowledged and also the system management part for this VNA is informed. It performs the needed resource requirements for task migration. If there is any time out or problem during the stages of the migration protocol, e.g. a VNA is already blocked for different and still running migration the process is stopped for now, the triggering VNA is informing all participant which roll back possible configuration changes.

A sequence of set-ups is possible to prevent deadlocks due to resource constraints, e.g. first a set-up is enacted to free up resources then the set-up for the actual task migration is done. Parallel task relocations are possible as long they do not share tiles with VNAs and VNIC to be reconfigured.

4 Summary

In this paper we have described how the regular structure of tiled manycore architectures can be utilized to enhance dependability of applications running on it. With HW supported communication and processing virtualization it is possible to migrate tasks transparent within such a platform under real-time and performance constraints. Different task migration strategies can be used based on the required reliability level of its application. For triggering the actual migration a threshold-based configuration on individual task basis is proposed. This concept would allow to run dependable applications on a platform with generally undependable components by preventing single points of failure, live substitution of failed components and virtual redundancy.

5 Acknowledgments

This work is supported in parts by the German Research Foundation (DFG) as part of the priority program “Dependable Embedded Systems” (SPP 1500 - spp1500.itec.kit.edu).

References

[Aga07] A. Agarwal. The Tile processor: A 64-core multicore for embedded processing. In *Proceedings of HPEC Workshop*, 2007.

[B+04] S. Borkar et al. Microarchitecture and design challenges for gigascale integration. In *Proceedings of the 37th annual IEEE/ACM International Symposium on Microarchitecture*, pages 3–3. IEEE Computer Society, 2004.

[BDF+03] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield. Xen and the art of virtualization. *ACM SIGOPS Operating Systems Review*, 37(5):164–177, 2003.

[CCS10] F. Checconi, T. Cucinotta, and M. Stein. Real-time issues in live migration of virtual machines. In *Euro-Par 2009–Parallel Processing Workshops*, pages 454–466. Springer, 2010.

[CLO+08] K. Chanchio, C. Leangsuksun, H. Ong, V. Ratanasamoot, and A. Shafi. An efficient virtual machine checkpointing mechanism for hypervisor-based HPC systems. In *Proc. of the High Availability and Performance Computing Workshop (HAPCW)*, 2008.

[DKR08] T. Distler, R. Kapitza, and H.P. Reiser. Efficient state transfer for hypervisor-based proactive recovery. In *Proceedings of the 2nd workshop on Recent advances on intrusiton-tolerant systems*, page 4. ACM, 2008.

[Dub05] P. Dubey. Recognition, mining and synthesis moves computers to the era of tera. *Technology@ Intel Magazine*, pages 1–10, 2005.

[GHKR11] M. Gries, U. Hoffmann, M. Konow, and M. Riepen. SCC: A Flexible Architecture for Many-Core Platform Research. *Computing in Science & Engineering*, pages 79–83, 2011.

[Hei09] G. Heiser. Hypervisors for consumer electronics. In *Consumer Communications and Networking Conference, 2009. CCNC 2009. 6th IEEE*, pages 1–5. IEEE, 2009.

[JRK10] B. Jiang, B. Ravindran, and C. Kim. Lightweight live migration for high availability cluster service. *Stabilization, Safety, and Security of Distributed Systems*, pages 420–434, 2010.

[lin] Hardware-Assisted Reliability Enhancement for Embedded Multi-core Virtualization Desig.

[NSL+06] G. Neiger, A. Santoni, F. Leung, D. Rodgers, and R. Uhlig. Intel virtualization technology: Hardware support for efficient processor virtualization. *Intel Technology Journal*, 10(3), 2006.

[NX06] V. Narayanan and Y. Xie. Reliability concerns in embedded system designs. *Computer*, 39(1):118–120, 2006.

[PTS11] A. Polze, P. Troger, and F. Salfner. Timely Virtual Machine Migration for Pro-Active Fault Tolerance. In *Object/Component/Service-Oriented Real-Time Distributed Computing Workshops (ISORCW), 2011 14th IEEE International Symposium on*, pages 234–243. IEEE, 2011.

[RWH10] H. Rauchfuss, T. Wild, and A. Herkersdorf. A network interface card architecture for I/O virtualization in embedded systems. In *Proceedings of the 2nd conference on I/O virtualization*, pages 2–2. USENIX Association, 2010.

[TSKM08] Y. Tamura, K. Sato, S. Kihara, and S. Moriai. Kemari: virtual machine synchronization for fault tolerance. In *Proceedings of the USENIX Annual Technical Conference 2008 (Poster Session)*, 2008.

[VPD04] J.P. Vasseur, M. Pickavet, and P. Demeester. *Network recovery: Protection and Restoration of Optical, SONET-SDH, IP, and MPLS*. Morgan Kaufmann Publishers, 2004.

[WSC+07] P. Willmann, J. Shafer, D. Carr, A. Menon, S. Rixner, A.L. Cox, and W. Zwaenepoel. Concurrent direct network access for virtual machine monitors. In *High Performance Computer Architecture, 2007. HPCA 2007. IEEE 13th International Symposium on*, pages 306–317. IEEE, 2007.

Two-Way-Compiler: Additional Data Saving for Generating the Original Source Code of a Binary Program

Dennis Obermann, Josef Börcsök

Department of Computer Architecture and System Programming
University Kassel
Wilhelmshöher Allee 71
34121 Kassel
obermann@uni-kassel.de

Abstract: The Two-Way-Compiler is an approach to show the equivalence between implemented source code and the generated binary program for safety-related software. A compiler which translates a source code into a binary program and restores the original source code out of the generated binary program exactly, like a decompiler, is described. Data that are required to build the original source code back again are especially examined in this paper. Some data are contained in the binary itself and other data lost during compilation. The lost data have to be collected and stored in the binary. With these additional data the decompiler can restore the binary program to the original source code.

1 Introduction

An implemented source code has to be translated by a compiler into a binary to get an executable program for safety-related systems. Frequently, the compiler is classified as not trustable. Therefore, intensive tests of the binary program against the requirements and the source code have to be performed. These tests are helpful to verify that the compiler does exactly what it should do. IEC 61508-3 mentions this in the development life cycle for software verification [IEC10]. A special test, which proves that the binary program corresponds exactly to the implemented source code, is not explicitly shown in this standard. But for the verification of safety-related software such a test is very important.

To verify safety-related software, it would be helpful to be able to translate the binary program back to its original representation. A simple comparison between the original source code and the decompiled source code will be possible.

A program that creates source code out of a binary program is called a decompiler. The first decompiler that translates binary programs from second generation computers to third generation computers was developed in the 1960s [Ha62]. In the following years, the techniques of decompiling were further developed and used to porting programs from one machine to another machine, to make documentation of assembler code, to rescue lost source code and to modify binary programs [Ci94]. Today, many decompilers are based on the phases of compilers and use identical techniques to analyze the input [Em07]. They analyze the binary program by graph theory, translate it into an intermediate representation and generate the source code. Control flow analysis and data flow analysis are very important [Ci94]. Human readability also plays a fundamental role [Ch10].

In some approaches the decompilers use assembler code [Sa66] or binary programs with debug informations as input. These inputs contain additional data that simplify the generation of the source code. There are symbolic informations about data segments, types, subroutine names, entry points and exit statements [Ci94]. Other decompilers are working on byte code like the java byte code [HD09]. Byte codes are executed by a virtual machine and contain more informations than binary programs. There are informations about data segments, types, method names, member names, entry points and exit statements [Vi03].

But all known decompilers are only able to generate code into one direction: From the binary program to the source code. They are working on the instructions of the binary program. Only a functional equivalence between the binary program and the decompiled source code can be achieved. In [PW93] such a decompiler is used to verify the equivalence between a binary program and the implemented source code of safety-related software. The decompiler translates the binary program and shows the functional equivalence by formal methods. But to compare the original source code and the decompiled source code they have to be translating both source codes into a formal representation.

One of the main problems is that many data are lost during compiling. But these data are required to restore the source code exactly. Therefore, this approach examines not only the decompiler phases, but also the compiler phases to gather the data that are normally lost.

Section 2 shows the symmetry between compilers and decompilers, while Section 3 gives an overview on this approach and describes the data that are required to restore the original source code. Section 4 concludes this paper.

2 Symmetry between a compiler and a decompiler

Modern compiler translates the source code in a binary program by a sequence of phases [Ah06]. The lexical analysis read the source code and split it into meaningful sequences called lexemes. For each lexeme it creates a token that contains the lexeme itself and an identifier of the token type. The sequence of tokens is handled by the syntax analysis. If it is possible to generate a syntax tree, the source code will be syntactical correct. The semantic analysis does verifications about types and language specific characteristics which cannot specify by a context free grammar. At the end of these frontend phases the source code is presented as an intermediate representation. All required symbolic informations are collected in the symbol table, which can access in every phase of the compiler. Optimizations will be performed on the intermediate representation and on the target code instructions before the binary program is generated.

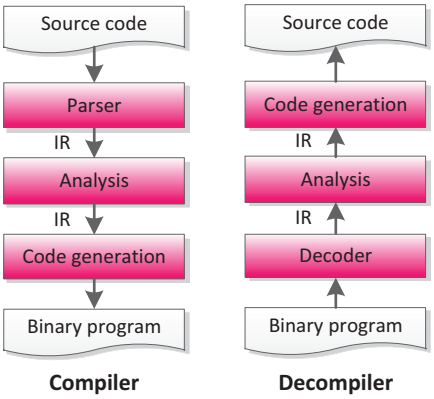


Figure 1: Symmetry between a compiler and a decompiler [Em07]

A decompiler generates the source code in a sequence of phases, too [Em07]. The decoder reads the binary program and splits the instructions from the data. It analyzes the control flow and the data flow and creates an intermediate representation of the binary program. Machine specific constructs will be replaced by corresponding constructs of higher programming languages and type informations are reconstructed. Programming language specific constructs will be recovered from the intermediate representation and the source code is generated.

Emmerik describes the symmetry between a compiler and a decompiler and consisders the similar techniques in the phases [Em07]. Figure 1 gives a short overview of the symmetry between a compiler and a decompiler. It shows that the decompiler is an inverse of a compiler.

3 Approach

In this approach the compiling and decompiling are considered together. Each phase of compilation has to be covered by a phase of decompilation. The symmetry between a compiler and a decompiler is the starting point of this approach. In contrast to a conventional compiler, the details of structure and format of the source code are not allowed to get lost during compile time. They have to be collected and stored in the binary program itself. Using this additional data, the original source code is reproducible.

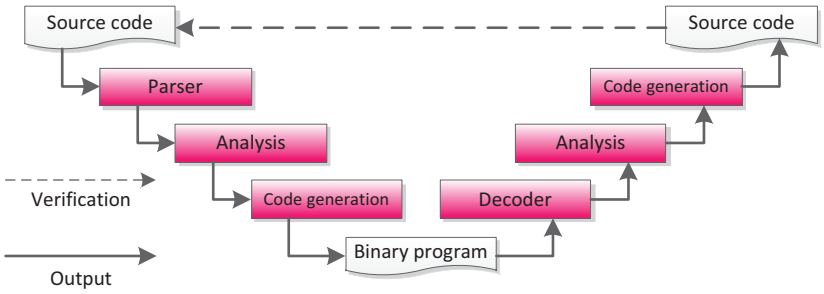


Figure 2: V-Model of the Two-Way-Compiler

3.1 Compiler phases

The compiler phases are used to transform a given source code to an executable binary program, which can be restored to the original source code back again. Figure 3 shows a simplified scheme of the compiler phases.

The lexical analysis reads the source code and generates a sequence of tokens. Normally, whitespaces and comments are ignored at this point of compilation. This approach uses an ignore list in addition to the symbol table. The ignore list collects all the ignored lexemes and their beginning positions. The syntax analysis works on a context free grammar and parses the token sequence into an abstract syntax tree. Special tokens that are lost during this phase are collected in the ignore list, too. The semantic analysis does type checking and fill the symbol table. It collects symbol informations like variable names and function names. Each entry holds the token informations and a data type at the end of this phase.

After the front end has collected the required data, the abstract syntax tree is transformed into a SSA form. This intermediate representation is used by many compilers [Cy89] and is qualified for decompilers, too [Em07]. The mapping between nodes in the abstract syntax tree and the SSA statements has to be reversible. Currently, there are not considered optimizations in this approach.

The back end generates the binary program out of the SSA form. In this approach the selection of target code instructions is very important, because the sequence of instructions has to identify the SSA form during the decompilation. Addresses for variables in the data sections are calculated and added to the corresponding symbol in the symbol table. The addresses of function entry points in the code sections are added to the symbol table, too. At least the binary is build and gets two additional sections. The first one contains the data from the ignore list and the second one holds the data from the symbol table.

The result of the compiler phases is an executable binary program that contains additional informations to restore the original source code back again. These informations are explicit collected data and informations contained in the structure of the binary program.

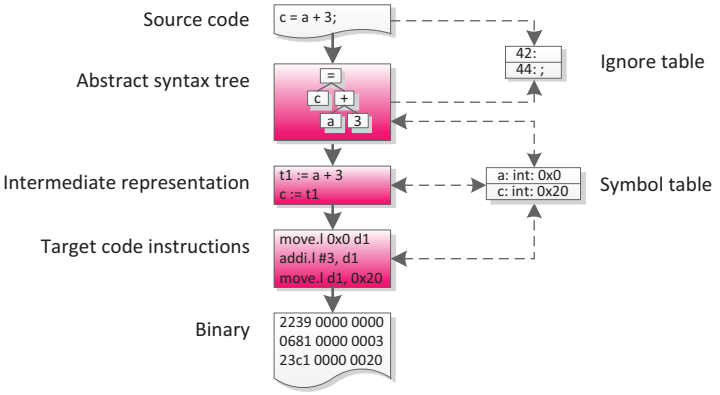


Figure 3: Simplified compiling scheme

3.2 Informations in binary programs

There are informations that can be gained from the binary program itself. These data can be evaluated by control flow and data flow analysis and this is also done by many decompilers [Vi03]. Control flow analysis [Al70] and data flow analysis [Ki73] are well known techniques and used for optimizations in a compiler, normally. Control structures, functions, function calls, operators and variables can be identified by using these techniques. The special selection of the target code instructions, the knowing of the programming language grammar and the knowledge of the mapping between the different representations make it possible to identify these informations and restore the corresponding lexemes in this approach.

To restore the exact source code, it is important that the compiler does not perform any optimizations in any phase. Optimizations would change the control flow and the data flow [SS08]. After optimizations the sequences of target instructions are modified. Thus, the decompiled control structures of the source code would not be the same as in the original source code. The reliability is more important than highly optimized programs in safety-related software. So, this restriction can be accepted for the compiler in this approach.

3.3 Lost data

Normally, many data are lost during the compiling process. In this approach these data are collected in every single phase of the compiling. The compiler saves whitespaces, formatting and comments of the source code in their ignore list during the lexical analysis. The names of functions and variables are lost, too. These data are collected in the symbol table of the compiler. In addition to the identifiers, the symbol table collects the memory addresses and the data types of the variables and functions. Typically, tokens like brackets are lost in the syntax analysis. These tokens are not in the abstract syntax tree and have to be stored in the ignore list, too. For example, this is necessary to restore arithmetical expressions that contain brackets exactly.

All collected data have to be included in two separate sections at the end of the binary program. The data from the symbol table and the informations of the ignore list are stored in these sections. Both sections are enclosed by special section markers to identify these sections during decompilation. To reduce storage space, the additional data sections are compressed and protected by a checksum.

3.4 Decompiler phases

The decompiler phases are used to restore the original source code out of the generated binary program exactly. Only binary programs can be handled that are generated from the compiler in this approach. Figure 4 shows a simplified scheme of the decompiler phases.

The decoding phase of the decompiler reads the binary program and separate instructions from the data. It identifies the symbol table section and the ignore table section. After decompressing these sections, the ignore list and the symbol table are filled with the existing data. The data from the restored symbol table and the restored ignore list are required in the following phases.

Because the decompiler has the same informations of the programming language grammar and the target code specification as the compiler, it knows the sequences of instructions and their parameters. Each identified sequence is transformed into the SSA form. The variable names and function names are obtained by the memory addresses and the data from the symbol table which were restored from the binary program. The symbol table contains the data types for variables, function parameters and function return values. It contains the names for variables and functions, too. After the transformation from the binary program to the intermediate representation, the control structures of higher programming languages are restored. The knowing of the programming language grammar and the knowledge of the mapping between the different representations makes it possible. A check for the syntax of the control structures are performed by transforming the SSA form into an abstract syntax tree. The syntax is correct, if the abstract syntax tree can be generated.

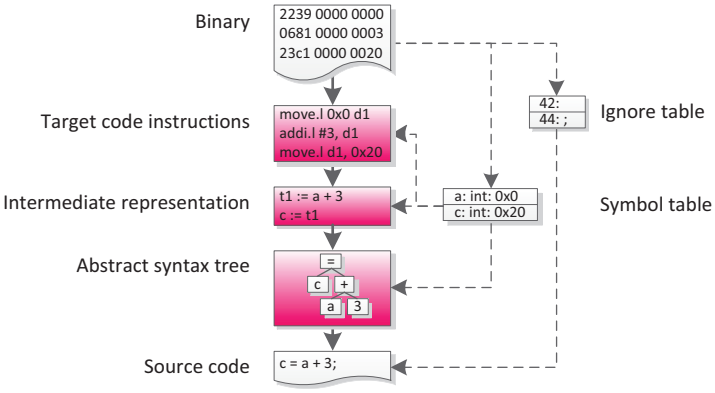


Figure 4: Simplified decompiling scheme

In the last phase the informations from the ignore list are used to restore the formatting and comments. The decompiler iterates the control structures and compares the positions in the ignore list. If the position is valid, it appends the ignored data to the decompiled source code. Inconsistent positions indicate an error during the decompilation. The decompiler uses the informations in the knowing programming language grammar to restore the identifiers for operators and control statements.

The result of the decompiler phases is the source code for the binary program that is the same as the original source code. It can be used to compare the decompiled source code with the original source code and shows the equivalence between the binary program and the source code.

4 Conclusion

This approach makes it possible to translate a source code into a binary program and back again. Through the simultaneous consideration of the compiler and the decompiler, as well as appending additional data to the binary program, it is possible to produce the original source code out of the binary program exactly.

One limitation of this approach is that no optimizations are done during compile time. This can be accepted, because the focus lies on safety-related software. A further disadvantage is the increased storage space of the generated binary program, which is created by the additional data.

However, the main advantage is that the verification between source code and binary program can be performed by a simple comparison between the original source code and the decompiled source code.

References

[Ah06] Aho, A. V.; Lam, M. S.; Sethi, R. and Ullman, J. D.: Compilers: Principles, Techniques, and Tools. Prentice Hall, 2nd edition, 2006.

[Al70] Allen, F. E.: Control flow analysis. In Proceedings of a symposium on Compiler optimization, 1970; pp. 1-19.

[Ci94] Cifuentes, C.: Reverse Compilation Techniques. PhD thesis, University of Queensland, 1994.

[Ch10] Chen, G.; Wang, Z.; Zhang, R.; Zhou, K.; Huang, S.; Ni, K.; Qi, Z.; Chen, K. and Guan, H.: A refined decompiler to generate c code with high readability. In 17th Working Conference on Reverse Engineering (WCRE), 2010; pp. 150-154.

[Cy89] Cytron, R.; Ferrante, J.; Rosen, B. K.; Wegman, M. N. and Zadeck, F. K.: An effcient method of computing static single assignment form. In Proceedings of the 16th ACM SIGPLAN-SIGACT symposium on Principles of programming languages (POPL), 1989; pp. 25-35.

[Em07] Emmerik, M. V.: Static single assignment for decompilation. PhD thesis, University of Queensland, 2007.

[Ha62] Halstead, M. H.: Machine-independent computer programming. Spartan Books, 1962.

[HD09] Hamilton, J. and Danicic, S.: An Evaluation of Current Java Bytecode Decompilers. In 9th IEEE International Working Conference on Source Code Analysis and Manipulation (SCAM), 2009; pp. 129-136.

[IEC10] IEC: Functional safety of electrical / electronic / programmable electronic safety-related systems - Part 3: Software requirements (IEC 61508-3:2010). International Electrotechnical Commission, 2010.

[Ki73] Kildall, G. A.: A unified approach to global program optimization. In Proceedings of the 1st annual ACM SIGACT-SIGPLAN symposium on Principles of programming languages (POPL), 1973; pp. 194-206.

[PW93] Pavey, D. J. and Winsborrow L. A.: Demonstrating equivalence of source code and prom contents. The Computer Journal, 36(7), 1993; pp. 654-667.

[Sa66] Sassaman, W. A.: A computer program to translate machine language into FORTRAN. Proceedings of the Spring joint computer conference (AFIPS), 1966; pp. 235-239.

[SS08] Srikant, Y.N. and Shankar, P.: The compiler design handbook: optimizations and machine code generation. CRC Press, 2nd edition, 2008.

[Vi03] Vinciguerra, L.; Wills, L.; Kejriwal, N.; Martino, P.; and Vinciguerra, R.: An experimentation framework for evaluating disassembly and decompilation tools for C++ and java. In Proceedings of the 10th Working Conference on Reverse Engineering (WCRE), 2003; pp 14-23.

Verlässlichkeit bei Wiederverwendung von IT-Komponenten – zum Stand der Normungsaktivitäten

Fevzi Belli

Fakultät für Elektrotechnik, Informatik und Mathematik
Universität Paderborn
Warburger Str. 100
33098Paderborn
belli@upb.de

Abstract: Die wirtschaftliche Bedeutung der Wiederverwendung von Komponenten ist evident: Je öfter eine bereits fertig gestellte Komponente in verschiedenen Systemen (wieder) verwendet wird, desto mehr wird an Herstellungskosten gespart. Bei aller Wirtschaftlichkeit darf jedoch nicht vergessen werden, dass die Verlässlichkeit einer Komponente vielfach von dem Verwendungszweck und dem Kontext abhängt, in dem diese Komponente (wieder) eingesetzt wird. Dieser Beitrag gibt einen Überblick über Normungsvorhaben bzgl. Anforderungen und Tests zur Sicherung der Verlässlichkeit bei Wiederverwendung bereits benutzter Hardware- und Software-Komponenten. Das Ziel ist, eine breite Diskussion zu initiieren und damit eine höhere Sensibilität für diesen Bereich zu schaffen.

1 Einleitung und Motivation

Bei physikalisch verschleißbaren Komponenten, z.B. in der Elektro-Industrie, ist die Wiederverwendung gebrauchter Komponenten in neuen Produkten noch immer ein heikles Thema, was leicht verständlich ist: Wenn Ihnen jemand ein neues Gerät „unter Verwendung von neuwertigen Teilen“ als neu verkaufen möchte, würden Sie es kaufen? Sie zögern? Tatsächlich gelangen jedoch immer mehr Geräte samt ihren neuwertigen Komponenten in den Abfallstrom, die noch weitere „Leben“ haben könnten. Zum Glück wurde das immens große Nutzungspotential der Wiederverwendung zur Schonung unserer Umwelt bereits erkannt und wird auch, allerdings im Augenblick noch von nur einigen wenigen Anbietern, genutzt, beispielsweise von Herstellern der Kopier- und Medizingeräten. Jüngere Industrie- und Rechtsnormen auf nationaler und europäischer Ebene haben zum Ziel, dieses Potential im großen Stil besser auszunutzen, unter Beachtung legaler, ökologischer und Verlässlichkeit-Aspekte.

Software kann dagegen nicht verschleißen, daher ist ihre Wiederverwendung unproblematischer und wird auch in der Software-Industrie gern und oft praktiziert. Die Aspekte der Verlässlichkeit und Ökologie wurden jedoch bisher nicht explizit manifestiert.

Sowohl bei Komponenten der Hardware als auch der Software ist zu berücksichtigen, dass die Qualität einer Komponente vielfach von dem Verwendungszweck und dem Kontext (Neu-Deutsch auch „Domain“ genannt) abhängt, in dem sie (wieder) verwendet wird. Mit anderen Worten, eine für einen bestimmten Einsatz bereits „qualitätsgesicherte“ Komponente kann bei einem erneuten, anders gearteten Einsatz eine Minder-Qualität haben, was u.U. zum Versagen des Systems führt. Daher ist eine adäquate Qualitätssicherung unter Berücksichtigung unterschiedlicher Verwendungszwecke und Einsatz-Konfigurationen eine unabdingbare Voraussetzung für die Wiederverwendung vorbenutzter Komponenten.

Diese Tatsache wurde von der Industrie zuerst für den Bereich elektrischer und elektronischer Bauteile (was hier als „elektr(on)isch“ abgekürzt wird) erkannt; ein Normungsvorhaben zur Sicherung der Verlässlichkeit durch geeignete Maßnahmen initiiert und mit Erfolg abgeschlossen; dies ist die internationale Norm IEC 62309 (*Dependability of products containing reused parts – Requirements for functionality and tests*), bekannt auch unter der Bezeichnung DIN EN 62309 (VDE 0050) (*Zuverlässigkeit von Produkten mit wieder verwendeten Teilen – Anforderungen an Funktionalität und Prüfungen*).

Augenblicklich beschäftigt sich die Arbeitsgruppe K 134 (*Gebrauchsfähigkeit und Qualität bei erneut verwendeten Teilen und Geräten der Elektrotechnik*) der DKE (Deutsche Kommission Elektrotechnik - Elektronik - Informationstechnik im DIN und VDE, als Deutsches Mitglied in IEC und CE) mit einem Parallelwerk zu IEC 62309 unter der Bezeichnung *Dependability of Software Products containing reusable components – Requirements for functionality and tests* (IEC NP56-1332).

Der vorliegende Beitrag gibt einen Überblick über die Norm IEC 62309 sowie den Entwurf IEC NP56-1332 und erläutert ihre Struktur und Verwendung.

IEC 62309 umfasst nicht nur technische und wirtschaftliche Aspekte, sondern wirft auch gesellschaftliche und rechtliche Fragestellungen auf. Diese Aspekte und Fragestellungen werden im nächsten Abschnitt erörtert. Der Entwurf IEC NP56-1332 konzentriert sich eher auf technische Inhalte.

Abschnitt 3 umfasst die deutschen Normungsaktivitäten im Bereich Software-Wiederverwendung, während Abschnitt 4 Aspekte der Verlässlichkeit diskutiert. Abschnitt 5 fasst diesen Beitrag zusammen und schließt ihn mit einigen perspektivischen Bemerkungen ab.

2 Wiederverwendung von elektr(on)ischen Komponenten

Schonung unserer Ressourcen kommt an der Wiederverwendung von Bauteilen in neuen Produkten nicht vorbei. Nach bisherigen Erfahrungen eignen sich etwa 25 % der elektr(on)ischen Bauteile für die Wiederverwendung, was eine Kostensenkung in Milliardenhöhe bedeuten kann.

Die Wiederverwendung wurde inzwischen auch gesetzlich als die wünschenswerteste Form der Abfallbehandlung eingestuft. Die Förderung der Wiederverwendung erfolgt auf verschiedenen Ebenen und es ist damit zu rechnen, dass sie ihre Bedeutung weiter

zunehmen wird, wie auch aus der neuen Abfallrahmenrichtlinie zu ersehen ist. Daher wird jeder Entscheidungsträger für technische Produkte zukünftig mit der Frage des Einsatzes gebrauchter Teile in neuen Produkten und allen Nebenaspekten konfrontiert werden.

Die Akzeptanz der Wiederverwendung hängt zum einen vom Vertrauen des Kunden in die Qualität der wieder verwendeten Komponenten ab. Zum anderen muss für den Hersteller der Aufwand überschaubar bleiben, um dieses Vertrauen zu gewährleisten. Schließlich ist es notwendig, jegliches Risiko der Wiederverwendung für die Umwelt auszuschließen. Das verursacht Kosten.

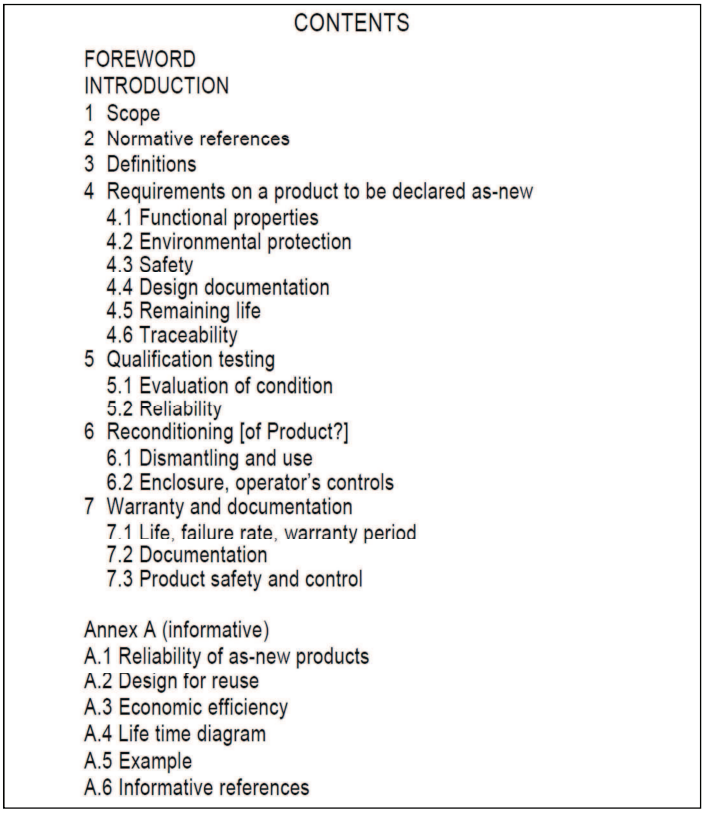

CONTENTS

Abbildung 1: Struktur der Norm IEC 62309

Zum Aufzeigen gangbarer Lösungswege für die Probleme der Wiederverwendung elektr(on)ischer Produkte und deren Komponenten wurde die internationale Norm *IEC 62309* von einem Team weltweit anerkannter Experten ausgearbeitet. In dieser Norm werden die Begriffe „neues Produkt“„ neuwertig“, „Gesamt-Lebensdauer“ u.v.a. aus dem Blickwinkel der Wiederverwendung überdacht und neu definiert. Auch die Schritte vom Ausbau der Komponente aus einem gebrauchten Produkt bis hin zum Vertrieb des neuen Produktes werden erläutert. Weitere Hilfestellung für die Praxis geben die Werke

[BBQ10] und [BQ12], welche den technischen, wirtschaftlichen und rechtlichen Fragestellungen eigene Kapitel widmen.

Abbildung 1 stellt die Struktur der Norm IEC 62309 dar; Abschnitt 4.1 geht auf die Verlässlichkeitsaspekte näher ein.

3 Wiederverwendung von Software-Komponenten

Die bisherigen Ausführungen legen bereits dar, dass der Qualitätsaspekt, insbes. Testen bei Wiederverwendung, einen Kostenfaktor darstellt. Dieser Aspekt wird augenblicklich kontrovers diskutiert.

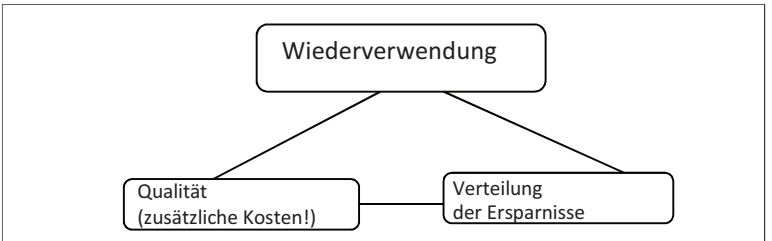


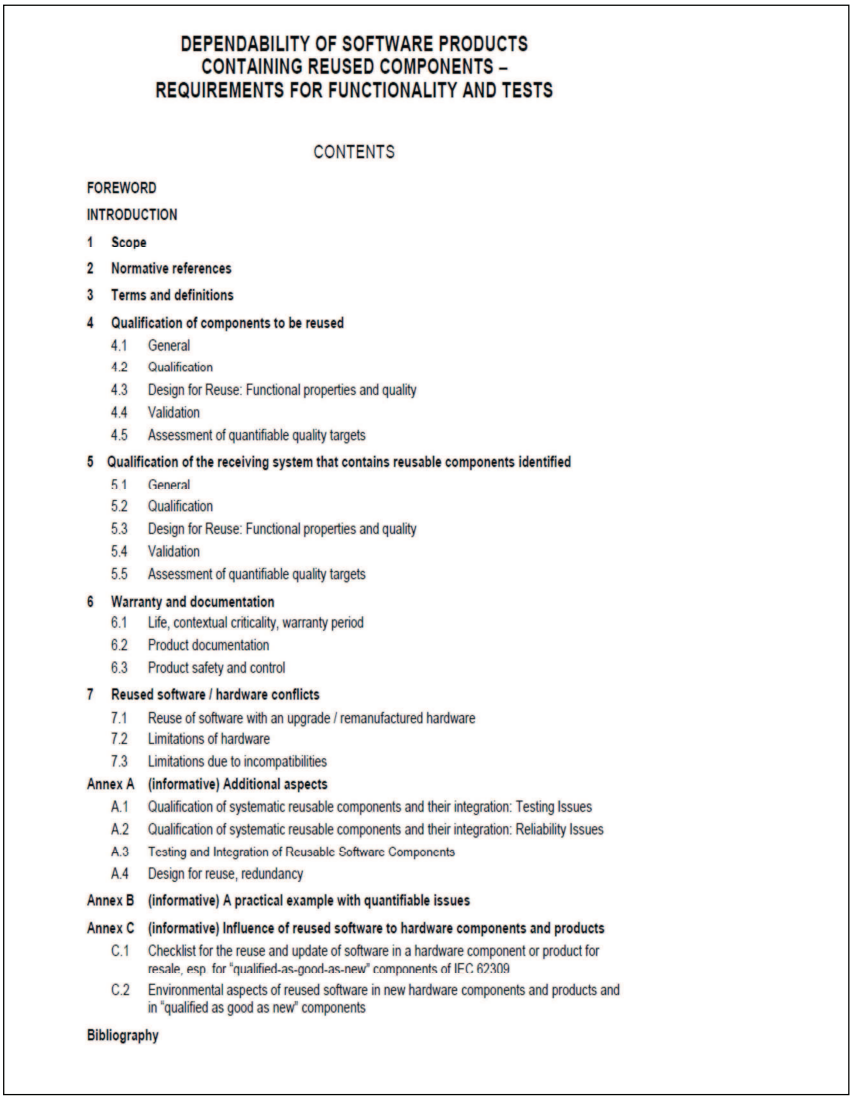
Abbildung 2:Spannungsfelder im Bereich Software-Wiederverwendung

Kontrovers diskutiert wird vor allem die Weitergabe der Ersparnisse durch den Hersteller: Der aufgeklärte Kunde, nicht zuletzt institutioneller Kostenträger eines Software-Systems, z.B. ein Autohersteller, ist heutzutage durchaus in der Lage zu erkennen, dass das erstellte System Komponenten erhält, die Bestandteile eines bereits existierenden Systems sind, wenn auch in leicht veränderter, angepasster Form, z.B. Software zur Realisierung von Wegfahrsperren bei Autos. Daher wird es ihm (als Hersteller von PKWs) nicht gefallen, die Kosten solcher Komponenten für jedes Modell seiner Produktpalette mehrfach und in gleicher Höhe in Rechnung gestellt zu bekommen.

Bei herkömmlichen technischen Systemen ist der Vorteil der Wiederverwendung für Endverbraucher eine Selbstverständlichkeit: Keinem Kunden werden beispielsweise die Konstruktionskosten einer M6-Schraube in Rechnung gestellt, wenn er ein Gerät kauft, in dem eine solche Schraube verwendet wird. Er bezahlt lediglich den Preis dieser Schraube als Ergebnis einer Massenfertigung, die entsprechend einer Norm genau spezifizierte Bauteile produziert. Bei Software-Erstellung handelt es sich um Erstellung eines Unikates, das in beliebiger Anzahl zu vernachlässigbaren Kosten kopiert werden kann.

Bereits die obige, kurze Diskussion erklärt die Brisanz der Wiederverwendung und das *Konfliktpotential* in dem Dreieck *Kostensenkung durch Wiederverwendung – Qualität - Verteilung der Ersparnisse* (Abbildung 2). Während der *Software-Ersteller* durch Wiederverwendung Ersparnisse erwirtschaftet, möchte der Kunde als *Kostenträger* an diesen Ersparnissen teilhaben. Sein „verlängerter Arm“, die *Qualitätssicherung*, wird dagegen durch spezifische, zusätzliche Prüfungen (als „Kostenverursacher“) diese Ersparnisse schmälern.

Lösungswege für diese Probleme möchte der Entwurf IEC NP56-1332 als Richtlinie geben, welche augenblicklich den Gegenstand der Arbeiten der bereits im ersten Abschnitt erwähnten Arbeitsgruppe DKE/K 134 bildet. Die vorläufige Struktur dieser Richtlinie wird im Abbildung 3 wiedergegeben. Dieser Entwurf, auf deren Verlässlichkeitsaspekte Abschnitt 4.2 eingeht, möge den Weg einer baldigen Norm ebnen.



DEPENDABILITY OF SOFTWARE PRODUCTS CONTAINING REUSED COMPONENTS – REQUIREMENTS FOR FUNCTIONALITY AND TESTS

CONTENTS

Abbildung 3: Struktur des Entwurfs IEC NP56-1332 bzgl. Verlässlichkeitsaspekte bei Wiederverwendung von Software-Komponenten

4 Wiederverwendung und Verlässlichkeit

4.1 Wiederverwendung von elektr(on)ischen Komponenten

Die Norm IEC 62309 regelt den qualitativen Rahmen bezüglich des Umgangs neuer Produkte mit gebrauchten Teilen. Sie stellt ein Konzept zur Überprüfung der Zuverlässigkeit und Funktionalität von wieder verwendeten Teilen und ihren Einsatz in neuen Produkten vor. Gleichzeitig werden Kriterien benannt, welchen Prüfungen die Produkte unterzogen werden müssen, die wieder verwendete Teile enthalten. Bezogen auf eine vorgesehene Nutzungsdauer des Produktes können sie nach Erfüllung der Kriterien als „qualifiziert als neuwertig" bezeichnet werden, was als begriffliches Herzstück dieser Norm angesehen werden kann: „Quagan": „Qualified as good as new". Zweck der Norm IEC 62309 ist es deshalb durch Prüfungen sicherzustellen, dass die Zuverlässigkeit und Funktionalität eines neuen Produktes mit wieder verwendeten Teilen vergleichbar sind mit einem Produkt, das nur neue Teile enthält. Dem Hersteller wird durch die Anwendung der Norm ermöglicht, dem Kunden für das Produkt mit „qualifiziert als neuwertigen" Teilen die volle Garantie wie für Neuware zu gewähren.

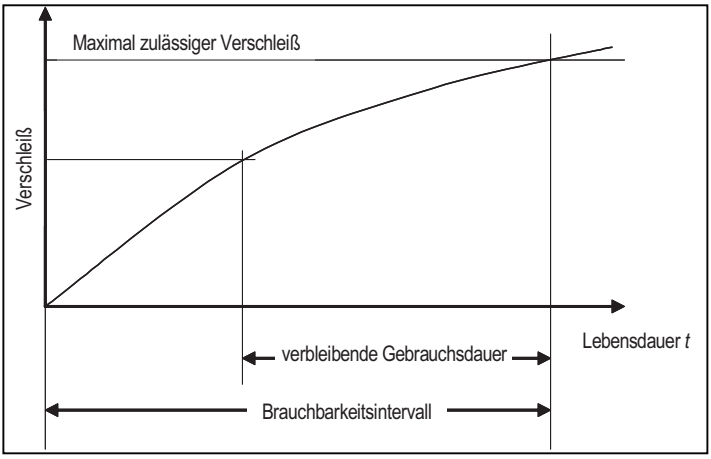


Abbildung 4: Zuverlässigkeit elektronischer Bauteile mit oder ohne Burn-in

Einen wesentlichen Punkt in der Norm stellt die folgende Aufforderung dar: „Wieder verwendete, „qualifiziert als neuwertige" Teile im Herstellprozess sollten die gleiche Funktionalität wie neue Teile haben und ihre erwartete Gebrauchsdauer muss mindestens der Auslegungslebensdauer eines neuen Produktes entsprechen". Damit wird klar, dass Teile mit einer Rest-Gebrauchsdauer, die sich nicht mehr für ein ganzes Produktleben eignen, nicht mehr im Blickfeld dieser Norm sind.

In Abbildung 4 stellt die Kurve die Ausfallrate gegen die Zeit dar, wie sie für elektronische Komponenten mit oder ohne Burn-in wiedergegeben wird. Sofern diese Kurve nachweislich gilt (dies wird immer wieder für manche Komponenten bestritten), sind auch elektronische Komponenten in dem Konzept zu berücksichtigen.

Abbildung 5 stellt die wiederholten Lebenszyklen von gebrauchten Quagan-Teilen nach IEC 62309 dar. Dabei darf die Restgebrauchsdauer wieder verwendeter Teile as-new Auslegungsdauer (ALLL) des Produktes nicht verkürzen (siehe Abbildung 5). Dies sollte verifiziert werden, entweder durch Analysen unter Nutzung von Informationen über die Ausfallverteilung neuer Teile und der verbleibenden Gebrauchsdauer der wieder verwendeten Teile, oder mit Stichproben aus dem Los der Teile, die als „qualifiziert als neuwertig" ausgewiesen sind. Solche Prüfungen führen zur Schätzung der Lebensdauerverteilung. Um Prüfungen/Analysen bei Einzelherstellung (keine Massenproduktion oder Herstellung mehrerer Teile) zu ermöglichen, sollten historische Daten von ähnlichen Teilen verwendet werden.

Kennlinien ermöglichen Angaben über die Restgebrauchsdauer der Teile, die dem Verschleiß unterliegen (siehe Abbildung 5). Dieses Modell kann für verschiedene Zuverlässigkeitskenngrößen angewendet werden. Entsprechende Kenngrößen sollten abhängig von den Anforderungen an das Produkt oder an die Teile ausgewählt werden.

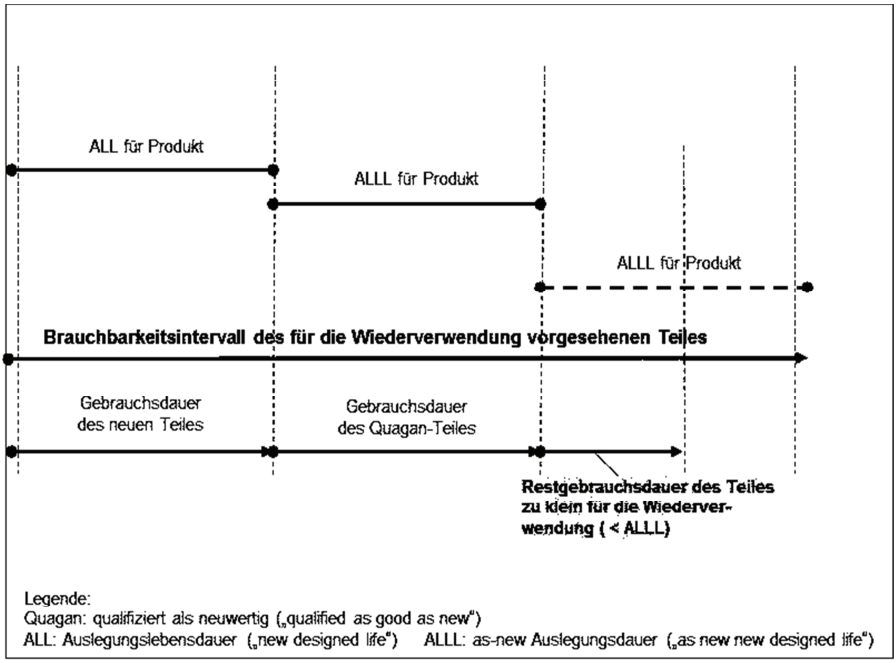


Abbildung 5: Wiederholte Lebenszyklen von gebrauchten, aber Quagan-Teilen nach IEC 62309

4.2 Wiederverwendung von Software-Komponenten

Software-Wiederverwendung bezeichnet den Prozess der Entwicklung von Software-Systemen unter Einsatz vorhandener Software, statt Neu-Entwicklung. Dieser Gedanke ist so alt wie der Begriff "Software" und wurde bereits 1968 im legendären NATO-

Workshop in Garmisch manifestiert, genau in dem Jahr also, als “Software Engineering“ als eine technisch-wissenschaftliche Disziplin konstituiert wurde.

Zahlreiche Berichte aus der IT-Branche bestätigen den stolzen wirtschaftlichen Erfolg der Software-Wiederverwendung: Fast alle größere Firmen loben diesen Gedanken, wie z.B. Nippon Electronic Company, GTE Corporation, Raytheon, DEC, HP, NASA, u.v.a.

Konträr zu den wirtschaftlichen Erfolgen warnen allerdings die Katastrophenberichte vor der Unterschätzung der Gefahren des sorglosen Umgangs mit Software-Wiederverwendung. Das Unglück des Röntgensystems Therac-25 wurde eindeutig durch die ungeprüfte Wiederverwendung der alten Software-Version im neuen System verursacht, wobei mehrere Menschen buchstäblich durch Verbrennung getötet wurden. Seit diesem Unglück ist bei den Herstellern von Medizingeräten die Software-Wiederverwendung ein Tabu-Thema, wie auch DKE K 134 bei den Verhandlungen mit den Vertretern eines deutschen Konzerns erfahren musste.

Ein weiteres Beispiel ist durch den Satelliten Ariane gegeben, dessen Absturz mehrere hundert Millionen Dollars kostete. Auch dieses Unglück wurde durch unsachgemäße Software-Wiederverwendung verursacht.

Für Software-Wiederverwendung ist aus den o.g. Gründen der Nachweis von fundamentaler Bedeutung, dass die Funktionsfähigkeit und die Verlässlichkeit eines Software-Systems mit vorbenutzten Komponenten voll und ganz denjenigen des Software-Systems entsprechen, das vom Grund aus neu entwickelt werden würde. Daher ist bei geplanter Wiederverwendung die Komponente durch deren Hersteller zu testen; dieser ist auch verpflichtet, die vorgesehenen Einsatzgebiete, so weit wie möglich, zu spezifizieren, und/oder komplementär die Gebiete zu kennzeichnen, in denen ein Einsatz ausgeschlossen wird.

Diese „Nachprüfung“ des Komponentenherstellers befreit keineswegs den Hersteller des Gesamtproduktes von der Pflicht der „Vorprüfung“ vor der Benutzung einer Komponente und der Prüfung des zusammengesetzten Systems. Die Prüfung entspricht einem Regressionstest, der sorgfältig ausgeführt werden muss.

Ein weiterer, wichtiger Aspekt der Wiederverwendung bildet die Sicherung der Energie-Effizienz und Öko-Freundlichkeit der wieder verwendeten Software, welche die Hardware steuert, in die sie eingebettet wird. Auch hier gilt, was für die Verlässlichkeit wahr ist: Eine Komponente minderer Qualität, die wieder verwendet wird, wird bei jedem Einsatz ihre „Un-Qualität“ wiederholen.

Insgesamt zielt der Entwurf IEC NP56-1332, die Norm IEC 62309 bzgl. Funktionalität, Test und Verlässlichkeit zu komplettieren, in der Software-Wiederverwendung ausdrücklich ausgeschlossen wurde.

5 Zusammenfassung und Ausblick

Dieser Beitrag fasste die Probleme und Chancen der Normung für die Sicherung der Verlässlichkeit bei Wiederverwendung vorbenutzter elektr(on)ischer und Software-Komponenten zusammen.

Obwohl im Bereich Software die Wiederverwendung schon sehr lange und gewinnbringend praktiziert wird, ist hier ein relativ großer Widerstand gegen eine Norm zu fühlen. Dagegen konnte im Bereich Hardware bereits 2004 eine internationale Norm erstellt werden, währen die Bemühungen im Bereich Software offensichtlich noch lange andauern werden.

Es bleibt zu hoffen, dass die Ausarbeitung des Entwurfs IEC NP56-1332 unter Einbeziehung weiterer Beteiligter und der Berücksichtigung ihrer Argumente die Normungsarbeiten im Bereich Software weiterbringen und somit IEC 62309 erfolgreich ergänzen wird.

Literaturverzeichnis

[BBQ10] Belli, F.; Bohnstedt, J.; Quella, F.: Einsatz gebrauchter Komponenten in neuen Produkte der Elektrotechnik, VDE-Schriftenreihe „Normen verständlich“, Band 136, CDE-Verlag, 2010.

[QB12] Quella, F.; Belli, F.: Reuse of components and products - 'qualified as good as new', in Handbook of Sustainable Engineering, Springer, to appear in 2012.

Tutorial on Reconfigurable Systems

organized by

Dirk Koch, Universitetet i Oslo, Norway
Jim Torresen, Universitetet i Oslo, Norway
Christian Beckhoff, ReCoBus
Daniel Ziener, Universität Nürnberg-Erlangen, Germany
Christopher Dennl, Universität Nürnberg-Erlangen, Germany
Volker Breuer, Universität Nürnberg-Erlangen, Germany
Jürgen Teich, Universität Nürnberg-Erlangen, Germany
Michael Feilen, Technische Universität München, Germany
Walter Stechele, Technische Universität München, Germany

Partial Reconfiguration on FPGAs in Practice
—
Tools and Applications

Dirk Koch[1], Jim Torresen[1], Christian Beckhoff[2], Daniel Ziener[3], Christopher Dennl[3], Volker Breuer[3], Jürgen Teich[3], Michael Feilen[4], and Walter Stechele[4]

[1] University of Oslo, Norway, Email: {dirk, jimtoer}@ifi.uio.no
[2] ReCoBus, Email: christian@recobus.de
[3] University of Erlangen, Germany,
Email: {daniel.ziener, teich}@informatik.uni-erlangen.de
[4] TU Munich, Germany, Email: {michael.feilen, walter.stechele}@tum.de

Abstract. Run-time reconfiguration of FPGAs has been around in academia for more than two decades but it is still applied very seldom in industrial applications. This has two main reasons: a lack of killer applications that substantially benefit from run-time reconfiguration and design tools that permit to quickly implement corresponding reconfigurable systems. This tutorial gives a survey on state-of-the-art trends on reconfigurable architectures and devices, application specific requirements, and design techniques and tools that are essential for implementing partial run-time reconfiguration on FPGAs. This is followed by a demonstration of the floorplanning and constraint generation tool GoAhead. Furthermore, the tutorial will reveal several applications that benefit from partial reconfiguration, including network data processing, digital signal processing, cognitive radio, and systems on a reconfigurable chip. For these applications, the individual challenges and implementation issues are presented together with the achieved results. This tutorial demonstrates that partial FPGA reconfiguration is beneficial and applicable in industrial systems.

1 Introduction

In the early days of field programmable gate arrays (FPGAs), the available logic capacity was very limited and using run-time reconfiguration had been suggested to raise resource utilization. For example, the dynamic instruction set computer (DISC) [27] is capable to change its instruction set at run-time according to a running program. Consequently, only the currently used instructions are loaded on the FPGA which takes fewer resources than having a configuration providing all instructions at the same time. However, for decades implementing such systems required deep knowledge about the used FPGA architecture and has to follow an error prune difficult design flow. Furthermore, long configuration time was another crucial issue that detained the use of run-time reconfiguration (RC) in industrial applications.

With the progress in silicon process technology, logic capacity raised steadily while getting cheaper (and often more power efficient) at the same time. This

removed the pressure on the FPGA vendors to add better support for partial reconfiguration (PR) in their tools and devices. However, by heading towards huge 1M LUT devices (million look-up table FPGAs), things are changing dramatically at the moment. Note that we focus on SRAM-based FPGAs in this tutorial.

Here, the configuration time required to write tens of megabytes of initial configuration data is too long for many applications. For example, the PCIe host interface standard requires a device connected to the bus to answer within 100 ms after reset. This requires either a costly multi FPGA solution, changes in the software, or bootstrapping. In the latter option, only the time critical parts of the system (e.g., a PCIe interface core) will be configured at system start while leaving the configuration of the rest of the system to a second partial configuration step. Together with the advantages of resource saving (monetary cost and power consumption) FPGA vendors are now forced to enhance the support for run-time reconfiguration. In other words, partial reconfiguration will be available in the majority of future FPGA devices and corresponding tools.

This tutorial is devoted to: 1) PR design tools and 2) applications that substantially benefit from run-time reconfiguration. The first topic is presented in Section 2 that will introduce the special requirements on the physical implementation and that will give an overview on PR tools and implementation techniques. This is assisted by the case study of a reconfigurable CPU in Section 3. After this, it will be shown how run-time reconfiguration can improve resource utilization as well as system flexibility for different applications. Section 4 reveals a self-adaptive video processing platform that is capable of sharing reconfigurable regions by various modules arbitrary at run-time. Next, Section 5 presents a reconfigurable SQL database accelerator. After this, Section 6 and Section 7 give examples of how to reduce implementation cost while enhancing adaptability in SDR applications with the help of PR.

The here presented applications give only a short insight into the active research on implementing run-time reconfigurable systems and there are many more examples that demonstrate that the use cases of partial configuration are virtually unbound. For example, [14,3] demonstrate partial FPGA reconfiguration for image processing, [25,4] propose self-adaptive control systems, and [5,20] investigate to exploit partial reconfiguration for high performance computing (HPC).

2 Design Tools and Techniques for Partially Reconfigurable Systems

Partial run-time reconfiguration has to be supported by the FPGA and by the corresponding design tools. While for example all recent FPGAs from the vendor Xilinx support partial reconfiguration, this feature is only available for the Virtex series devices within the vendor tools. Moreover, this feature is not available by default and has to activated as a separate feature.

2.1 Terms and Definitions

A reconfigurable system is partitioned into two parts. 1) The part of the system that is always present (e.g., a memory controller or a soft CPU) is commonly called *static region* while 2) the part containing run-time reconfigurable modules is called *partial region*. If the static part of the system includes the logic to re-configure the device, this is called *self-reconfiguration*. All recent FPGAs from the vendor Xilinx provide an internal configuration access port (ICAP) for self-reconfiguration and no external pins are required to access the FPGA configuration. The same is announced for the new Stratix-V Series from Altera.

As shown in Figure 1, the partial region can be arranged in different configuration styles. In the easiest case, the partial region follows the *island style* approach that is capable of hosting one reconfigurable module exclusively per island. A system might provide multiple islands and if a set of modules can only run in one specific island this is called *single island style*. If modules can be *relocated* to different islands this is called *multi island style*

While the island style is ideal for systems where only a few modules are swapped, it might suffer *internal fragmentation*. This is a waste of logic resources that arises if modules with different resource requirements share the same island exclusively. This means, if a large module cannot be replaced by multiple smaller ones (to be hosted at the same time) the utilization of the reconfigurable region is getting weak. Internal fragmentation is improved by tiling reconfigurable regions using the one-dimensional *slot-style* or the two-dimensional *grid-style* approach. In this technique, a module occupies a number of tiles according to its resource requirements and multiple modules can be hosted simultaneously in a reconfigurable region. This is illustrated in Figure 1b) and Figure 1c)

Tiling the reconfigurable region is considerable more complex as the system has to provide communication to and from the reconfigurable modules and to determine the placement for a module. The latter one has to consider that FPGA resources are in general heterogeneous (i.e. there are different primitives like logic, memory, or arithmetic primitives on the fabric). Moreover, depending on the present module layout, a tiled reconfigurable region might not provide all free tiles as one continuous area. If this results in unused tiles, this overhead is called *external fragmentation*. External fragmentation can be removed by defragmenting the module layout which is called *compaction*. Note that some modules cannot be interrupted for compaction because a module would loose its internal state or it would not meet its throughput.

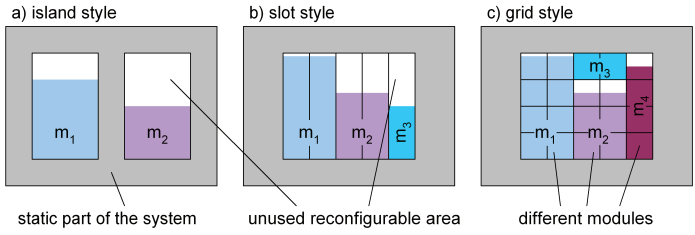


Fig. 1. Reconfiguration style.

2.2 FPGA Support for Partial Run-time Reconfiguration

Partial reconfiguration requires from the FPGA architecture that parts of the fabric can be overwritten without affecting other parts of system. Note that we imply that the static part of the system is active at any time (independent of a configuration process).

Different FPGA architectures show different behavior during the reconfiguration process. For example, older Xilinx FPGA families, including Virtex-II or Spartan-3 FPGAs, could only be reconfigured full column-wise, meaning that a full column of resources (logic, or routing) was affected when writing new configuration data to the device. This can cause side effects to the static system or other reconfigurable modules if they use resources in a column that can be reconfigured at run-time. On recent Xilinx FPGA architectures, the height of the columns had been reduced to the height of a clock region. This represents four block RAMs, four multiplier primitives, or 16-20 configurable logic blocks (CLBs), depending on the FPGA family. Note that the fabric of Xilinx FPGAs consists of a regular mesh of CLBs that each provide 8 look-up tables and an attached switch matrix to carry out the routing with surrounding CLBs. In some columns, the LUTs have been replaced with dedicated blocks such as memories while maintaining regularity of the mesh. There are several wires leaving each switch matrix that are named by their routing distance (e.g., single wires route one CLB further, double two CLBs, and so forth). By restricting reconfigurable tiles to span the height of full clock regions, side effects due to partial reconfiguration can be avoided on recent Xilinx FPGAs.

Altera announced to use mask mechanisms to control which part of the FPGA will be changed by reconfiguration. This technique should allow arbitrary reconfiguration without side effects to other active parts of a system [17].

2.3 Place & Route Constraints for Island Style Reconfiguration

For implementing a run-time reconfigurable system, we have to ensure that 1) a partial module uses only resources (logic & routing) that are not used by other parts of the system (another partial module or the static system) and 2) that identical wires of the FPGA fabric are used to connect partial modules over all physical implementation steps. This means that we have to constrain the routing to use a specific wire for crossing the module border for each signal bit of the module entity. The begin and end ports of these wires can be compared with plug-socket pairs on a printed circuit board. As additional constraints, we have to 3) activate all clock trees that are used by partial modules and 4) constrain the timing. This is mostly everything to consider and there is nothing mystic behind applying partial reconfiguration. The real difficulty stems from a lack in the design tools that, for example, provide no constraints to bind a signal to specific wires on the fabric. For instance, there is no constraint to bind a signal in a top level design that is responsible for the communication with a partial module to a specific wire that crosses the partial-to-static border.

To overcome this, macros called *bus macros* have been used in earlier partial design flows by the vendor Xilinx [15,28]. As shown in Figure 2.3 a), the signal to wire binding has been achieved by instantiating a macro consisting of a LUT

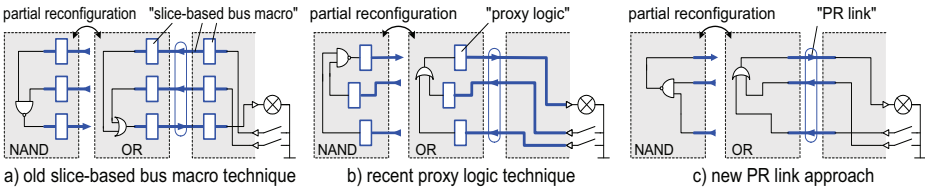


Fig. 2. Different approaches for constraining the signal to wire binding. In all three examples, a partial NAND module is exchanged by an OR module.

in the static part and a corresponding LUT in the reconfigurable region. By placing bus macros at a defined position on the partial module border, signals are bound to the internal macro wires.

The bus macro approach costs two LUTs per signal wire and additional latency. This was improved in the recent Xilinx vendor tools using an incremental partial design flow [29]. As shown in Figure 2.3 b), anchor LUTs in route-through mode (called *proxy logic*) are placed in the partial region for each signal crossing the partial module border. During the implementation of the static system, the partial interface signals are routed to the anchors. The partial modules are implemented incrementally from this static system without modifying any static routing. Consequently, the partial module interface wires are constrained by preserving the initial static routing.

The proxy logic approach costs one LUT per signal wire. Its main drawback is that the routing is different for each reconfigurable island. This prevents module relocation even if the islands provide an identical logic or memory layout. Moreover, changes in the static system will in general result in a different routing to the proxy logic. Consequently, all permutations of module type and placement position have to be rerouted on each modification in the static system. Hence, this approach is only suitable for systems of low complexity.

As a third approach, it is possible to prohibit selectively the use of wires by occupying them with the help of *blocker macros* [12]. Blocker macros instantiate within a defined region all logic primitives and use all available wires (or a selected set of wires) within this region. By selectively not blocking wires, *tunnels* can be drilled through a blocker such that only one possible routing path exists to route across the border to and from a reconfigurable module. By this, we force the router to bind a signal to the wires of the tunnel. By defining a tunnel in the partial region during the implementation of the static system and by defining a tunnel in the static region during the partial module implementation, each module interface signal is bound to a corresponding interface wire, called *PR link*. As shown in Figure 2.3 c), this allows module integration without logic overhead while permitting module relocation and an independent implementation of the static system or any partial module. The important difference between the proxy logic and the PR link approach is that the latter one constraints a signal to a specific wire rather than allowing the router to decide the signal to wire binding during the static system implementation.

For defining the placement of logic primitives and for specifying the timing requirements, the Xilinx vendor tools provide sophisticated constraints. Note that the definition of reconfigurable regions and module bounding boxes does not have to be rectangular. And because routing will not be interfered by recon-

figuration if the routing is overwritten with exact the same configuration data, it is possible to relax strict bounding box constraints for the routing. This can substantially improve performance and routability [13].

The clock signals are routed via dedicated clock networks. By connecting the clock to the blocker primitives located in the partial region during the static system implementation with the clock network, all clock network drivers are activated such that the reconfigurable modules can access one ore more clocks when loaded into the reconfigurable islands.

2.4 Communication Architectures for Slot and Grid Style Reconfiguration

When moving from an island style scenario to a slot or grid style reconfiguration scheme, extra precaution is needed to provide communication from and to the different partial modules loaded together into a reconfigurable region. In order to permit module relocation, we have to implement a *homogeneous communication architecture* that possesses an identical logic and routing footprint within each tile and at the tile borders. The base idea of such an architecture is shown in Figure 3 and related approaches have been presented as research work, e.g., [16,8,10]. The last work [10], is designed for high performance, flexible module placement and low implementation cost; and a system with 60 individual reconfigurable modules using grid style reconfiguration has been demonstrated in [11]. By arranging the logic and routing identical in each tile of a reconfigurable region and inside the modules, glitches on running bus transactions or data streams can be avoided during reconfiguration, regardless to the placement position. This is possible because the internal routing and logic of the communication architecture will never be changed at runtime.

2.5 PR Design Tools

The bus macro approach for Xilinx FPGAs has dominated the FPGA run-time reconfiguration community for more than a decade. However, Xilinx moved completely over to their new 4th generation PR flow based on proxy logic while

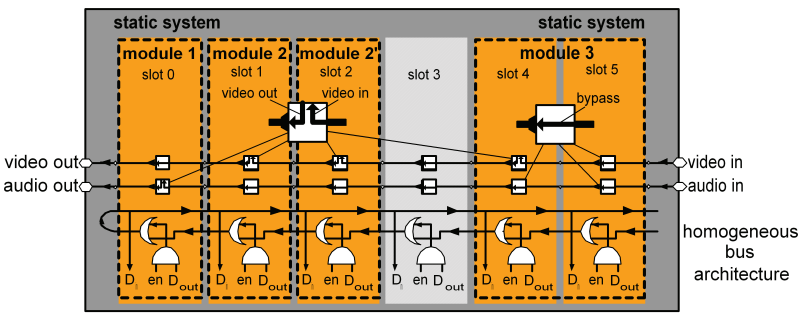


Fig. 3. Basic approach for a homogeneous communication architecture.

removing the support for bus macros. For this flow, Xilinx included some extra features in their flooplanning and constraint generation program PlanAhead. This includes resource budgeting, defining reconfigurable islands, and an automatic placement of proxy logic. While especially the last feature has substantially simplified the design flow, a lack of routing constraints prevents this flow to implement multi-island or slot and grid style reconfiguration as described in Section 2.3.

In addition to Xilinx, Altera (as another major FPGA vendor) has announced to support partial reconfiguration in their devices and design tools. According to [17], designing partial systems will be based on an incremental design flow very similar to the recent flow proposed by Xilinx. Consequently, these systems will include the same limitations (no module relocation, no static system to partial module decoupling during the implementation).

While a couple of research projects build tools on top of the Xilinx vendor PR tools, only little research was undertaken on developing independent alternatives. One alternative is the project ReCoBus-Builder [10]. This tool can generate constraints similar to PlanAhead, but it can also generate homogeneous communication architectures, as described in Section 2.4. In addition, the ReCoBus-Builder can constrain the routing by generating *blocker macros* as described in Section 2.3. The applications in Section 4, 5, and 6 have been implemented using the ReCoBus-Builder. As a further example, the tool OpenPR [24] reintroduces bus macros (see also Figure 2.3b)) into latest PlanAhead versions.

Next Generation PR Tools The preliminary purpose of PR design tools is to generate implementation constraints for the physical implementation of a system. Consequently, PR design tools have to generate constraints that are compatible with other tools following the constraints (e.g., place & route tools). Unfortunately, devices and design tools of the vendor Xilinx have changed that much that the concepts of the ReCoBus-Builder could not be easily shifted to recent devices and the support of that tool is limited to Spartan-3 and Virtex-II/II Pro FPGAs. Similarly, OpenPR is bound to Virtex-4 and Virtex-5 FPGAs.

For implementing run-time reconfigurable systems on recent FPGAs using latest vendor tools, a completely redesign of the ReCoBus-Builder is currently under development under the name GoAhead. This tutorial will announce its features and the tool will be available on the COSRECOS project website [2]. GoAhead provides a GUI (Fig. 4) and command script interface for floorplanning and macro placement that is similar to PlanAhead from Xilinx. The tool is shipped with a macro library containing various macros for different Xilinx FPGA families (Virtex-5/6/7 Spartan-6), including different *bus macros* and *connection macros*. A connection macro is basically a connection primitive (e.g., a look-up table) used to force the router to generate a routing path to or from this macro. GoAhead supports any reconfiguration style (Fig. 1) and the integration of modules using bus macros, PR-links (Fig. 2.3), or homogeneous communication architectures (Fig. 3). It can generate VHDL templates, UCF constraints (user constraints to be used with the Xilinx vendor tools), and routing constraints by generating blocker macros. The next section gives an example of how a reconfigurable system can be implemented using GoAhead.

Fig. 4. GoAhead GUI showing a Spartan-6 LX16 FPGA.

3 Reconfigurable Instruction Set Extensions

Changing the instruction set architecture (ISA) of a softcore CPU at run-time can substantially enhance performance and area at the same time. This has been demonstrated several times before (e.g., [27]) and even small instructions can gain high speed-ups. For example, if we consider a dedicated custom instruction for permuting all bits in a 32 bit operand, this would easily take a hundred cycles on a conventional CPU but would be only wiring, if implemented as a dedicated instruction. Implementing such instructions reconfigurable allows more instructions on less area. This involves some reconfiguration overhead that might be hidden by *configuration prefetching* in some systems. However, the real difficulty in implementing reconfigurable custom instructions is that a relatively large number of signals have to be connected to small modules. For example, a 32-bit instruction with two operands and one result requires a connection of roughly 100 signal bits (=wires); and assuming two LUTs per result bit, the instruction can be implemented in just eight CLBs on a Xilinx FPGA.

Figure 5 a) shows a simplified architecture of a CPU extended by four slots to host reconfigurable instructions. We will now reveal how a corresponding system can be implemented with the tool GoAhead. As sketched in Figure 5 b) for Xilinx Spartan-6 or Virtex-6 FPGAs, different wire resources have been used to connect two operands (with single and double lines) and the individual results (quad lines) for four adjacent slots. The wires have been chosen such that the operands and results can be connected at exactly the same relative position in each slot in order to permit module relocation. Modules may take more than one slot by using only one of the result vectors. Note that the input operands get swapped after each slot. This might require design alternatives, if operations are not commutative.

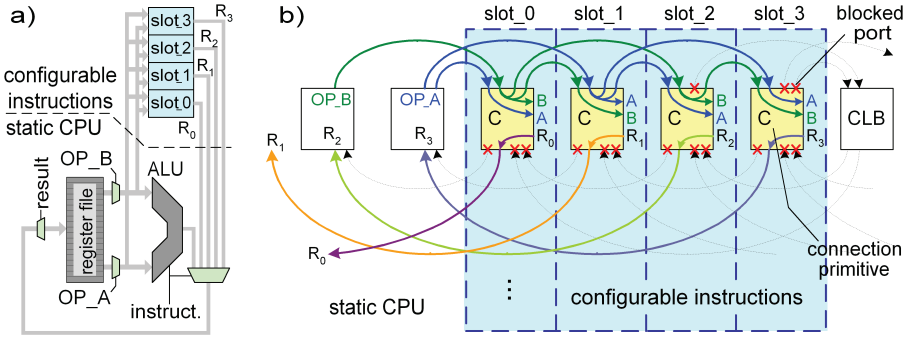


Fig. 5. Reconfigurable instruction set extension. a) RTL b) FPGA implementation. The implementation uses neatly single, double and quad lines that route one, two, and respectively four switch matrices (CLBs) further on Xilinx Virtex-6/Spartan-6 FPGAs.

On Xilinx Spartan-6 or Virtex-6 FPGAs we can connect four bits per two operands and one result per CLB. Figure 5 b) follows this mapping when assuming that each shown connection represents a bundle of four wires connected to a slice acting as the connection macro. Then a 32-bit instruction can be mapped in a slot as small as eight CLBs. The following GoAhead script places the connection macros, releases the ports used to route operands and results, generates a blocker, and writes instantiation code and user constraints for the Xilinx ISE tools. As can be seen, all constraints can be generated with only a few different commands. The X/Y coordinates are CLB tile coordinates, `WW4`, `EE2`, and `ER1` are names used by Xilinx for quad lines towards west, double lines towards east, and single lines towards east. The connection primitive `Connect4_S6_CI` is provided by GoAhead.

```
 1: AddSingleMacro MacroName=Connect4_S6_CI InstanceName=Slot0Inst0 Slice=SLICE_X19Y12;
 2: AddSingleMacro ...  # add 8 macros in each of the four slots
33: AddToSelectionXY X1=31 Y1=50 X2=34 Y2=57;  # select Slot0 and Slot1
34: DoNotBlockPort PortNameRegexp=WW4B; # release west quadline begin ports (result vectors)
35: DoNotBlockPort PortNameRegexp=EE2; # release east double line begin/end ports (operands)
36: DoNotBlockPort PortNameRegexp=ER1; # release east single line begin/end ports (operands)
37: AddToSelectionXY X1=35 Y1=50 X2=36 Y2=57;  # select Slot2
38: DoNotBlockPort ... # release used ports for slot2 and also slot3 according to Figure 5b)
50: PrintVHDLMacroInstantiation PortMapping=Sin:OPA,Din:OPB,Res:Res0,CLK:clk Filter=Slot0;
51: PrintVHDLMacroInstantiation ...  # generate VHDL instantiation code for all four slots
54: AddToSelectionXY X1=31 Y1=49 X2=38 Y2=57;  # select all four slots
55: BlockSelection OutFile=StaticBlocker.xdl  # block all remaining wire resources
56: PrintLocationConstraintsForPlacedMacros FileName=static.ucf;  # for ISE project
57: PrintPlacementProhibitConstraints FileName=static.ucf;  # prevent the Xilinx tools to
      # use any further primitive inside the current selection (i.e. the reconfig. area)
```

By placing the macros in a bottom-up order, we cause a connection of the operand and result vector signals also in a bottom-up order. This reduces congestion in modules using carry chain logic that propagates in the same bottom-up direction. By connecting four signals per CLB row, we incorporate that a carry chain processes also four bits per CLB row on Xilinx FPGAs. In order to generate the configuration bitstream for the static system, we run synthesis, technology mapping and placement in the Xilinx Vendor tools. After this, a batch script in-

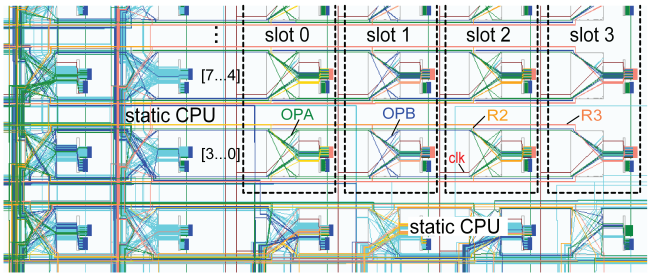


Fig. 6. FPGA Editor screen shot of a MIPS CPU providing four slots for reconfigurable custom instructions on a Xilinx Spartan-6 FPGA.

cludes the blocker into the design and runs the vendor router. Finally, we delete the blocker and generate the configuration bitstream. An FPGA Editor screen shot of the final system is shown in Figure 6.

The partial modules (i.e. the reconfigurable instructions) are implemented very similar to the static system, except that the connection macros are now placed left and right beside the used slots as a placeholder for the static system. We use blockers with released ports that result in an interface compatible to the static system. For the timing verification, GoAhead can compose netlists of the static system and any possible combination of placed modules. Finally, GoAhead generate partial bitstreams for a user defined region that can be directly written to any configuration port of the device.

As sketched in this section, GoAhead can be used for implementing very sophisticated reconfigurable systems. However, this requires still a considerable knowledge about the used FPGA architecture. The final version of the tool will provide wizards that hide most of the low level details of the fabric.

4 A Self-adaptive Reconfigurable Video Processing System

In this section, we present a system architecture for building partially reconfigurable *System-on-Chips* (SoCs), described in details in [21]. This architecture is exemplary applied for a smart camera system. FPGA-based embedded systems are of increasing importance especially in the signal and image processing domain. For instance, intelligent embedded systems for image processing, such as smart cameras, rely on FPGA-based architectures [23]. With the advantage of reconfigurability, we can envisage new designs with new and improved possibilities and properties, like an adaptive design which can adapt itself to a new operation environment. The static part of the system provides a CPU, the SoC infrastructure and the interfaces for the video input of the camera system. Most of the image processing algorithms, e.g., filtering, color transformation and detection, or visualization modules (called marker modules) are implemented as partially reconfigurable modules which can be dynamically loaded and unloaded at run-time.

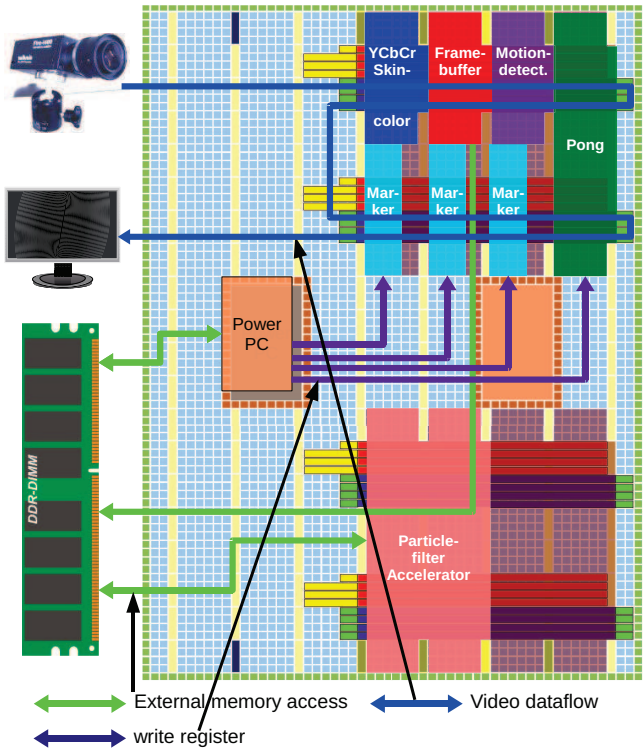


Fig. 7. System overview of the heterogeneous FPGA-based smart camera SoC platform consisting of CPU sub-system and reconfigurable area. Reconfigurable modules can vary in size and be freely placed, allowing a very good exploitation of the FPGA space.

4.1 Architecture

The system is implemented on the Xilinx Virtex-II Pro XUP board and consists of an embedded CPU sub-system including the external DDR-memory and the reconfigurable part (see Figure 7). In the following, these components and the communication interfaces between them are presented.

Embedded CPU Sub-system The main purpose of the software part on the embedded CPU is to control and manage the overall system. It contains high-performance peripherals, interfaces, and other IP cores. These are, e.g., a memory controller to provide access to an external RAM, a serial port interface for user commands, and a module for accessing the integrated reconfiguration interface of the FPGA. All components of the embedded CPU sub-system are connected by the main on-chip system bus, the *processor local bus* (PLB).

Reconfigurable Area The FPGA area is divided into a static and a dynamic part (see Fig. 7). The two marked areas on the right top and bottom compose the dynamic part of the system. Reconfiguration is only possible in the dynamic part

which contains a reconfigurable on-chip bus (*ReCoBus*) and *I/O bar* communication primitives to provide a communication infrastructure for dynamically loaded hardware modules. Both communication primitives part of the ReCoBus-Builder framework [10]. In the smart camera platform, the I/O bar is used to stream video data between the various reconfigurable processing modules. The modules can read and modify the video stream or generate additional output signals. To allow communication between the embedded CPU sub-system and the reconfigurable part, a PLB/RCB bridge translates the ReCoBus (RCB) protocol to the PLB protocol and vice versa. Using the ReCoBus and the bridge, the modules can be accessed from the CPU, e.g., to configure the module with memory-mapped registers. Furthermore, the modules have also direct access to the external memory (DMA). To allow high-speed data transfers between hardware masters and the memory controller, the bridge uses the *native port interface* (NPI) of the memory controller (provided by Xilinx).

4.2 Reconfigurable Modules

We implement several reconfigurable modules to tackle a wide spectrum of applications for our smart camera platform. In this section, we present some of these modules.

The *skin color detection* is implemented as a hardware slave module that reads the color values from the I/O bar and marks them as *skin* or *non-skin* by comparing them with a color template. We have implemented modules for RGB and YCbCr color spaces. The classification is written as an additional signal (skin color bit) onto the I/O bar together with the unmodified video stream.

The *filter module* is a sliding-window image processing filter. The current implementation supports a 3x3 filter matrix. To access different image lines, the module stores two lines in a BRAM-FIFO. The coefficients are stored in CPU accessible registers. Therefore, a module can be configured for different filter functions, for example, with the coefficients of a Sobel filter which can be used for edge detection.

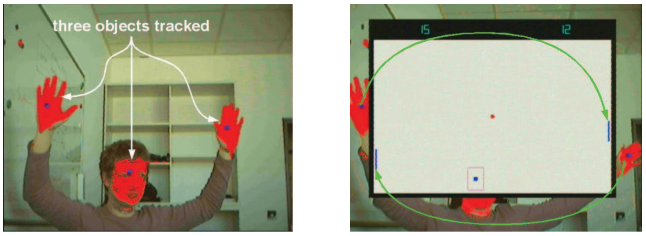
The *framebuffer* hardware master module is implemented to store the current input image. This is done by double buffering the images in the on-chip memory via the ReCoBus using the NPI interface. We use 32 Bit for storing one pixel, with 24 Bit for the input RGB values and the remaining 8 Bit free for classification results, e.g., the skin color bit.

The *particle filtering framework* is partitioned into a software and hardware part. The software part performs the sampling and applies the motion model. The hardware part is used as a co-processor to perform the evaluation steps.

The *motion detection module* compares the pixel values of two subsequent images to detection motion. Like the skin color detection module, the result (motion/no motion) is written as an additional signal onto the I/O bar.

The *pixel marker module* colors classified pixel or regions with a specified color. The classification of the pixel is signaled to the marker module with additional I/O bar signals. The color can be configured by a register interface.

An embedded design for tracking human motion is implemented as an example application to show the flexibility of the proposed platform. The idea is



(a) The particle filter tracking three objects.

(b) The object tracker used to play a pong game.

Fig. 8. The smart camera tracks three image regions (a person's head and hands). The tracked hand positions are directly used to control the paddles of the video game.

to detect and track skin-colored image regions, which is done by applying particle filtering. The current implementation makes it possible to track up to three image regions. One marker module is used per region tracker. A simple *tennis game* is implemented on top of this application, which can be directly controlled by the hands of a person, using the results of the tracker (see Fig. 8).

5 Partial Reconfiguration for SQL Query Processing

This section describes how dynamic partial reconfiguration can be used for SQL query processing for large databases. There exist already static FPGA approaches, e.g., *Glacier*, a query-to-hardware compiler [18], and Netezza's *FAST*-engines [7]. Both approaches have a lack of flexibility: *Glacier* has to run a full synthesis for every new incoming query and Netezza's *FAST*-engines have a fixed pipeline with no possibility to reorder operations.

We cover restrictions (WHERE-clauses) and allow different data types for table attributes as well as different operations on these data types. Restrictions are Boolean expressions which are used to filter out tuples from a table. Tuples remain in the result table if they are evaluated to *true* regarding the restriction, otherwise they are omitted. Figure 9 shows an overview of the proposed SQL accelerator technique.

We offer a module library with different operators, which can be used to assemble a pipeline at run-time by using partial reconfiguration, to form different restrictions. We support integer types up to 32 bit and fixed-length strings. Furthermore, we offer arithmetic-logical operators ($+$, $-$, $*$, AND, OR, NOT, XOR, $NAND$, NOR) and comparisons ($<$, $\leq$, $=$, $\neq$, $\geq$, $>$). The latter ones can be used on integer attributes as well as string attributes. The pipeline processes tuples one after another and computes the restriction result for each tuple. The supported operators only rely on tuple data itself, thus we can evaluate the restriction for each tuple itself independent from other tuples.

We use I/O bars to pass data from module to module. Furthermore, we use them to configure modules which are plugged onto the bars. Usually, tuples are wider than the data bus width of the I/O bars, which is 32 bit in our system, therefore tuples are divided into several chunks and tuples are processed

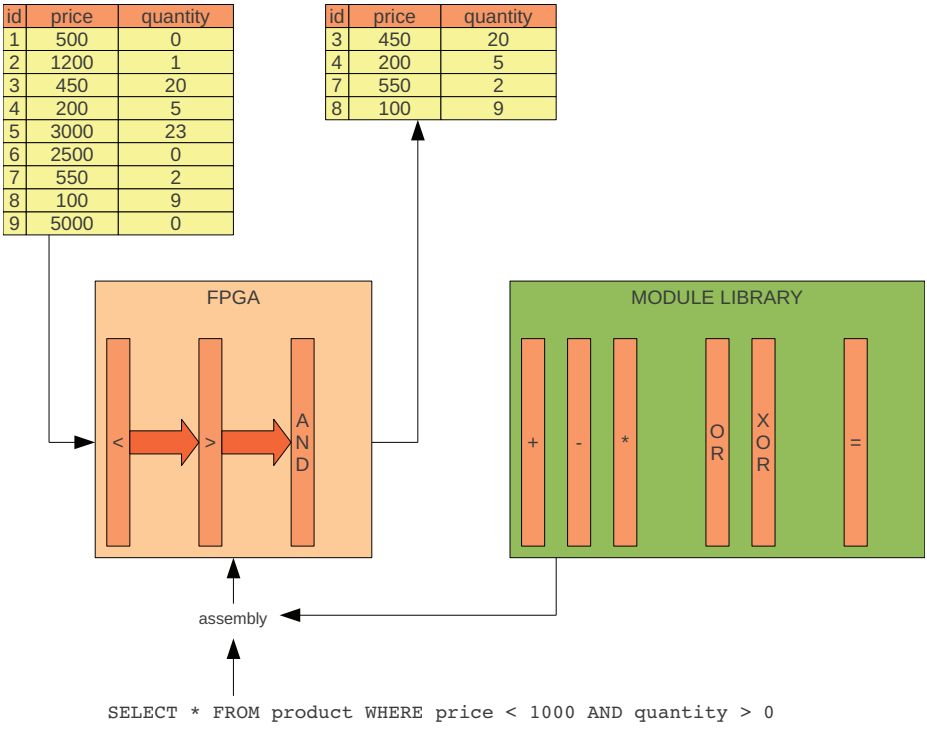


Fig. 9. FPGA configured with partial modules to perform a restriction (WHERE-clause). The partial modules are assembled to a pipeline for high-throughput.

chunk-by-chunk in a fully pipelined manner. Modules are configured with the information about tuple sizes and the chunk indices of their operands. Furthermore, they are told where to put their results in the tuple stream. The modules are capable of doing attribute-attribute operations as well as attribute-constant operations, i.e., either both operands are part of the tuple or one operand is a constant value which is configured during configuration of the module. Figure 10 shows an example of an arithmetic-logical module which performs an addition.

We append spare chunks to the tuples, thus the modules can place their results inside the tuple stream. By looking up the result of the last module in the pipeline, which is either *true* or *false*, we can decide whether we keep a tuple in the result table or not.

With the use of dynamic partial reconfiguration and a presynthesized module library, we are able to switch the functionality during run-time, thus we can execute queries with different restrictions one after another without any further synthesis, which was the drawback of *Glacier*. Furthermore, it would be possible to implement further modules to support more operations, e.g., projections, or more data types like floating point. Thus, we could switch the operation order of such SQL operations, which is not possible with Netezza's *FAST*-engines because their pipeline is fixed. We used the XUP Virtex-II Pro Evaluation Board for

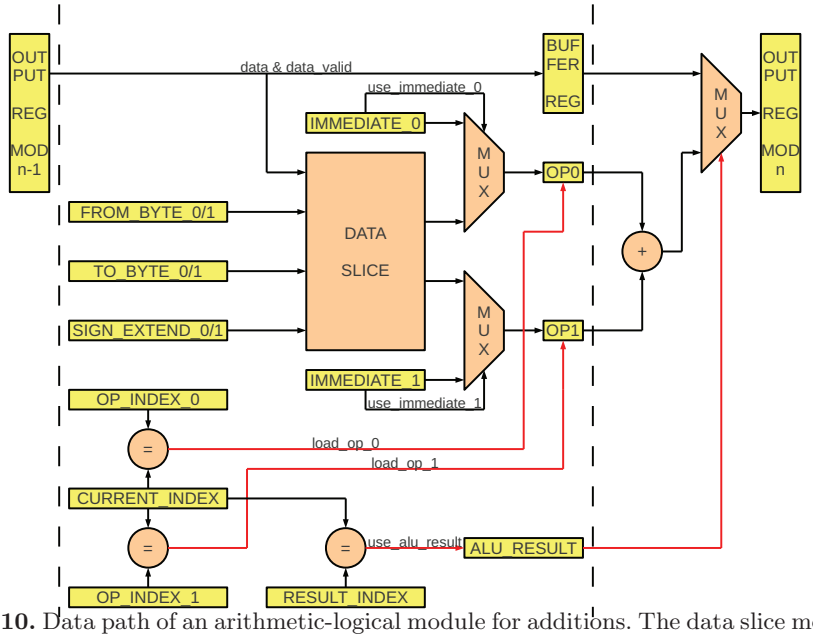


Fig. 10. Data path of an arithmetic-logical module for additions. The data slice module is responsible for truncating or padding attributes to the size of a 32-bit chunk.

prototyping. Our system runs with 100 MHz, thus we reach a total throughput of $400\frac{MB}{s}$ due to our pipelined design.

6 Parameter Controlled Software Defined Radio Platform

Software Defined Radio (SDR) is promoted by the the Wireless Innovation Forum which describes it as *"[. . .] a collection of hardware and software technologies where some or all of the radio's operating functions (also referred to as physical layer processing) are implemented through modifiable software or firmware operating on programmable processing technologies."* [26]. The idea behind SDR is to extend the range of flexibility in radio development and to provide the ability to adapt to future wireless standards, to add new features and capabilities to existing infrastructure or to fix corrupt designs which causes system misbehaviors. Such a system has been developed to enhance the flexibility of the hardware design of a wireless tracking technology by utilizing the benefits of the ReCoBus technology and partial reconfiguration of FPGAs. The system offers generic digital signal processing modules for mixing, filtering and correlation tasks. The modules can be placed as pleased and the necessary parameters like filter coefficients are configured by a SoC infrastructure.

6.1 Architecture

The system is implemented on custom hardware platform which features a Xilinx XC2VP70 FPGA. Furthermore, the hardware provides an optical datalink to a

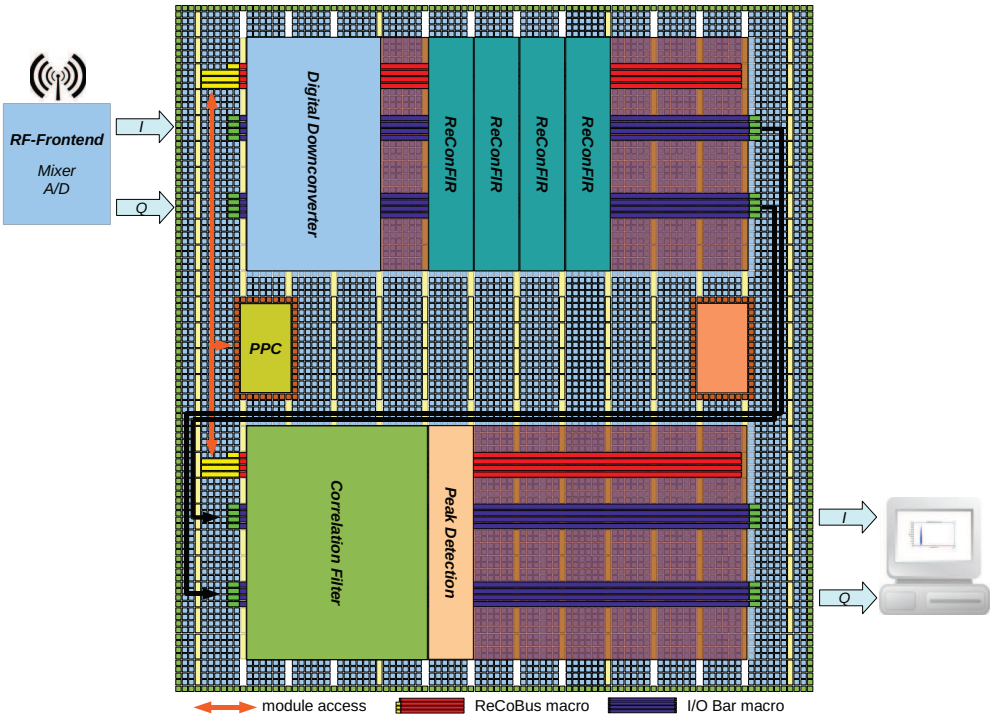


Fig. 11. SDR platform consisting of a controll CPU sub-system and two reconfigureable areas. The example shows the configuration used to track individual radio transmitters.

RF frontend and a 64 Bit PCI-X interface which is used to retrieve the processing results for further usage on the host system (see Figure 11).

One of the embedded PowerPCs of the Virtex-II Pro FPGA is used to create a configuration master which duty it is to communicate with configured modules via the ReCoBus macro to set module specific parameters.

The design features two large reconfigureable areas which can be used to freely place the designed modules. The areas are crossed by two I/O Bars with each providing a 32 Bit interface to offer a reasonable amount of signal bits. Above the I/O Bars, an on-chip bus structure has been place to configure and steer the modules (the ReCoBus macro). Furthermore, the reconfigureable areas feature two clock signals driven by low skew clock nets which can be utilized by modules independently of their placement.

6.2 Reconfigureable Modules

We implemented several reconfigurable modules for SDR applications. One property that applies to all modules is that the signal bit width is limited due to the finite size of the I/O Bars. As a consequence, all modules include an adjustable bit slicer to meet the specifications.

The *digital down converter module* divides the incoming baseband signal into subchannels and down samples the signal to the absolute necessary rate. As a new approach and as an extension to the existing ReCoBus design flow, this module is designed by structural description in *Xilinx System Generator* (XSG), which is a blockset add-on for the Matlab Simulink environment. This procedure enables the user to rapidly design complex modules.

The *mixer module* comprises a *Direct Digital Synthesizer* (DDS) to generate the sine and cosine waveforms and a complex multiplier to shift the signal from one frequency to another. The *ReConFIR* named module is designed to offer the user a generic 8-tap FIR filter. Necessary parameters, including the filter coefficients and the expected sampling rate of the FIR filter, are configured by the SoC. As a special feature, this module can be cascaded with other ReConFIR module instances to create an ReConfigureable n-tap FIR filter of variable length *n*. The *correlation filter module* is designed to meet the use case specific requirements regarding the type and amount of coefficients. Nevertheless, these can be configured via the PowerPC platform to change the tracked signals. The *peak detection module* searches for peaks in the correlation results and indicates its results by setting a signal.

6.3 Summary

The system benefits significantly from the discussed technologies by adding the ability of reusing design elements, changing key characteristics such as the used frequency band or the tracked signal sequences without the need to design and configure a completely new design. A big variety of use cases are imaginable besides the one presented. The static platform can not only be used for receiver but also for transmitter designs by making use of the already build generic modules and additional system specific modules. Especially pulse-based transmission systems (like the mentioned tracking technology) offer a great margin for run-time reconfiguration because of the idle time between two bursts.

7 An SNR-Adaptive Cognitive Software-Defined Radio using Partial Reconfiguration

In mobile reception scenarios, the signal-to-noise ratio (SNR) of a receive signal can strongly vary over time. In situations where the signal is weak it might be necessary to spend more computational effort in decoding the signal to get a better performance. On the other hand, when the receive signal is strong with respect to the noise, it might be beneficial to reduce the decoding complexity as the reception quality is sufficient for the target application. Bearing this in mind, the use of PR of FPGA resources enables to change the complexity of one component inside the SDR chain according to the perceived SNR while keeping the other components active. Thus, a PR-based design may give a benefit over a static design, due to the hardware reusability in the reconfigurable area.

In the following sections we will introduce a PR use case for broadcast receivers, where the algorithms of an FM radio receiver chain will be adopted according to the estimated signal-to-noise ratio. The reconfigurable FM receiver

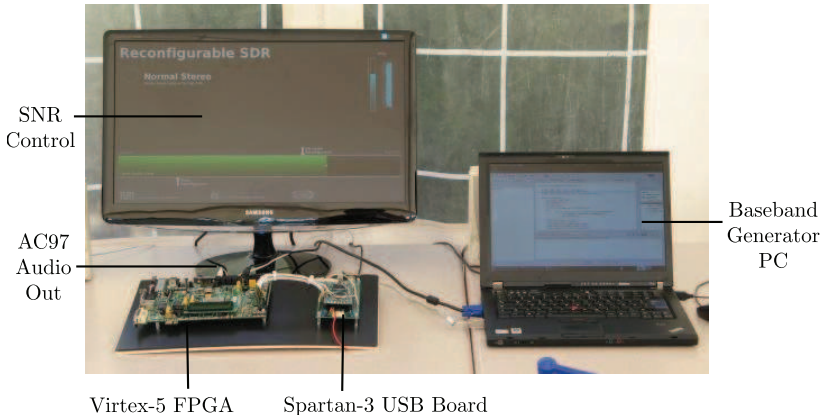


Fig. 12. The reconfigurable broadcast FM receiver demo system.

prototype was developed at the Technical University Munich as a simple case study to demonstrate the concept for SNR-adaptive cognitive radios and to serve as a template for further investigations of more complex applications.

7.1 Demonstration Platform

Our SNR-adaptive demo system implements a reconfigurable FM-RDS stereo broadcast receiver together with an SNR estimation stage, where the FM multiplex decoder can be reconfigured according to the estimated SNR [19]. The SNR is defined as carrier-to-noise ratio, reflecting the FM carrier signal power divided by the noise power.

The demonstration platform is shown in Figure 12 and consists of a PC, a Xilinx Spartan-3 FPGA and a reconfigurable Xilinx Virtex-5 FPGA (XC5VSX50T). The PC generates two complex baseband signals, each at a sample frequency of 500 kHz and transmits the data to the Spartan-3 FPGA via USB. The reconfigurable Virtex-5 device reads the data from a 16 bit parallel GPIO interface and processes it internally. The PC generates modulated FM stereo broadcast signals including Radio Data System (RDS) services and the SNR of these signals can be varied by adding white Gaussian noise to the respective stream.

On the reconfigurable Virtex-5 FPGA two FM baseband signals are received, decoded and the SNR is estimated. According to the estimated SNR at the receiver, the FM multiplex (MPX) decoding routines are adopted. For example, in case the SNR is very low, the receiver can either increase the computational effort and return a stereo audio signal which is more acceptable in quality or decrease the computational effort by switching to monaural decoding. In case the SNR is very high and the signal is very strong, the receiver can use low complexity demodulation algorithms while still getting a sufficient audio quality.

In the following sections, the receiver flow graph and the partitioning of the receiver chain on the FPGA are discussed.

7.2 FM Receiver Signal Flow and MPX Decoding

The flow graph of the FM receiver chain is shown in Figure 13. The receiver chain can be subpartitioned into three main parts, i.e. the mono decoding audio

part, the additional logic for stereo decoding and the RDS decoder. The mono signal combines the sum of the left and right audio signals. The stereo signal consists of the difference of the left and right audio signals. It is located at a frequency of 38 kHz using amplitude modulation with suppressed carrier. In order to coherently demodulate the stereo signal, the carrier has to be reconstructed. This is done by using the 19 kHz pilot tone which is extracted by a PLL in the stereo decoder part.

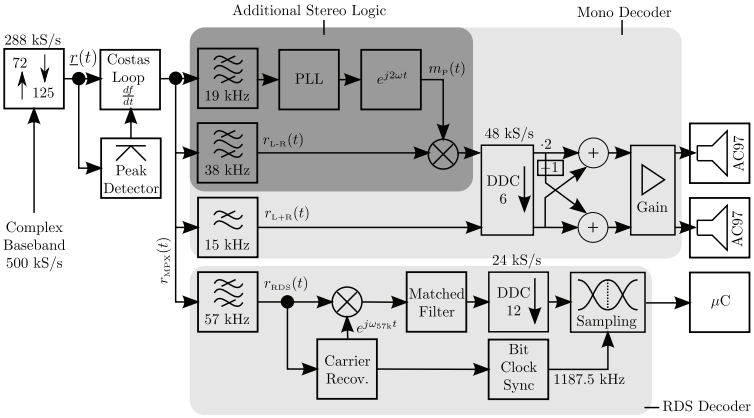


Fig. 13. FM receiver signal flow and partitioning. The grey background highlights the three different PR modules for MPX decoding.

In the next section, the partitioning of the decoder modules of the FM receiver chain for the implementation on the FPGA is presented. The module-based PR flow was used for the design of the reconfigurable partitions (c.f. [29]). The bitstreams for the different configurations are stored on a fast external on-board DDR2-Memory (max. 6.4 GB/s) and loaded on demand by the configuration control block.

7.3 Virtex-5 FPGA Partitioning

The FPGA configuration comprises one static and two reconfigurable partitions (reconfigurable islands). The static partition includes a Microblaze microcontroller, a multiplexed Costas loop for FM demodulation and two SNR estimation stages. The reconfigurable partitions are used for the demodulation of the respective MPX signal. Each partition can hold one of the following demodulator types: *Stereo* audio demodulator, *Mono* audio demodulator and an *RDS* demodulator. The number of logic elements of the partition was chosen with respect to the most complex design, i.e. the stereo demodulator. All configuration permutations are possible, e.g. the receiver can have two stereo demodulators, or one mono and one RDS demodulator, two RDS demodulators etc. The partitioning and signal flow of the demo platform is depicted in Figure 14.

In the figure, the reconfigurable partitions are denoted as *Partition A* and *Partition B*. Each partition can be reconfigured individually without interrupting

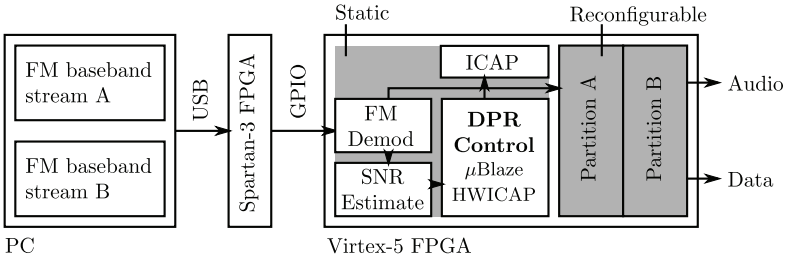


Fig. 14. Signal flow graph of FM demonstrator system. The Virtex-5 FPGA is used for FM demodulation and RDS decoding and comprises two reconfigurable partitions.

the other. Modules cannot be relocated and have been implemented separately for each partition (single island reconfiguration style). The microcontroller evaluates the estimated SNR values and is able to trigger a reconfiguration of partition A or B if the SNR reaches a certain threshold. The reconfiguration is done by reading the partial bitstreams from the external memory and writing them to the HWICAP module over the PLB. While the FM-MPX decoding chain is reconfigured via PR, the SNR estimation and the FM signal demodulation are constantly active in the static part of the device.

Either one of the three presented MPX decoder modules or an empty bitstream can be written to the reconfigurable FPGA partitions A and B. All configurations strongly differ in the number of slices, BRAMs and DSP multipliers as depicted in Table 1.

MPX Decoder	Slices	DSP48	BRAMs
Stereo audio decoder	804	9	3
Mono audio decoder	458	2	2
RDS data stream decoder	503	6	5
Empty (no decoding)	0	0	0

Table 1. MPX decoder resource overview.

The stereo decoder is the most complex decoding branch, followed by the RDS decoder and the monaural audio decoder. In case one of the received signals is too noisy to demodulate, the respective partition can be replaced. If the noise power increases above a level where decoding is not feasible anymore, the MPX decoder in question is replaced by an empty bitstream.

The trigger for the reconfiguration is given by the Microblaze CPU. The reconfiguration conditions are presented in the following paragraphs.

7.4 Reconfiguration Conditions

For mono broadcasts our experiments have revealed that the audio distortion at SNRs below 4 dB is so strong that it is unbearable for the listener. In this case, the mono decoder will be removed from the active partition after the received signal has fallen below this threshold.

For stereo broadcasts the SNR must be approximately 21 dB above the mono threshold [22]. This is due to the fact that in FM the power spectral density of the demodulated MPX signal increases quadratically as the frequency increases [9]. Since the stereo difference signal is located at an intermediate frequency of 38 kHz it is more prone to noise than the monaural sum signal at DC. Thus, in case of stereo broadcasts it is feasible for the receiver to switch from stereo to mono if the SNR drops below 25 dB. Similarly, with our decoder implementation the SNR threshold for decoding RDS with a bit error rate below 10^{-3} is reached at an SNR of approximately 25 dB. Below that threshold, the RDS decoder is replaced with an empty bitstream in order to reduce the dynamic power consumption.

Hence, if the SNR estimator signals that the SNR has fallen below a certain threshold, the Microblaze CPU initiates a PR of the FM multiplex decoder to become more or less complex. The SNR-thresholds for the different reconfigurable partitions are summarized in Table 2.

MPX Decoder	SNR Region
Stereo audio decoder	$\gamma \geq 25$ dB
Mono audio decoder	4 dB $\leq \gamma < 25$ dB
RDS data stream decoder	$\gamma \geq 25$ dB
Empty audio (no decoding)	$\gamma < 4$ dB
Empty RDS (no decoding)	$\gamma < 25$ dB

Table 2. SNR regions for PR partitions. γ denotes the carrier-to-noise ratio in dB.

With the presented configuration conditions, the resources inside the reconfigurable region can be traded with respect to the actual requirements. An important fact is that by using PR and by regulating the amount of dynamic logic on the device, the dynamic power consumption of the receiver can also be regulated according to the user constraints.

7.5 Summary

The SNR in mobile reception scenarios is a function of time and vicinity. With PR of FPGAs the logic occupation of a mobile SDR receiver can be adopted to the actual requirements. The MPX decoders of the twin-tuner FM receiver presented in the analysis can be modified independently during the runtime. Thus, PR enables more degrees of freedom for cognitive SDRs on FPGAs.

Nowadays the reconfigurable region must be chosen with respect to the largest configuration, which might cause fragmentation of reconfigurable areas. In the future, the fragmentation could be reduced by sub-partitioning the reconfigurable partitions as proposed in [6]. However, although supported by the FPGA fabric, at the moment PR sub-partitions and module relocation are not supported by the Xilinx software suite, for this reason, we will investigate alternatives, such as GoAhead.

8 Conclusion

In this paper, we demonstrated promising use cases for partial reconfiguration on FPGAs as well as corresponding design tools. This started from tiny reconfigurable modifications of a CPU, over complex video processing and database acceleration up to two software defined radio applications. Common for all these applications is that they benefit from being able to relocate modules to different positions and to pack multiple modules together into a shared reconfigurable region. While this is not well supported by the tools from the major FPGA vendors, we showed upcoming alternatives developed in academia.

Acknowledgment

This tutorial is joint work that is supported by different institutions. This includes the Norwegian Research Council founding the project Context Switching Reconfigurable Hardware for Communication Systems (COSRECOS) ([2]), under grant 191156V30 and the Bundesministerium für Wirtschaft und Technologie for supporting the project Programmable Telematic On-Board Radio (PROTON) ([1]) under Grant 10 P 8012B

References

1. Programmable Telematic On-Board Radio (PROTON), `http://www.proton-plata.fr`
2. Project website: *Context Switching Reconfigurable Hardware for Communication Systems*, `http://www.mn.uio.no/ifi/english/research/projects/cosrecos/`
3. Claus, C., Stechele, W., Kovatsch, M., Angermeier, J., Teich, J.: A Comparison of Embedded Reconfigurable Video-processing Architectures. In: International Conference on Field Programmable Logic and Applications (FPL). pp. 587–590 (2008)
4. Commuri, S., Tadigotla, V., Sliger, L.: Task-based Hardware Reconfiguration in Mobile Robots Using FPGAs. J. Intell. Robotics Syst. 49, 111–134 (June 2007)
5. El-Araby, E., Gonzalez, I., El-Ghazawi, T.: Exploiting Partial Runtime Reconfiguration for High-Performance Reconfigurable Computing. ACM Trans. Reconfigurable Technol. Syst. 1, 21:1–21:23 (January 2009)
6. Feilen, M.: Concept and Design of an SNR-adaptive DRM+/FM Receiver using Dynamic Partial Reconguration (DPR) of FPGAs. 11th Workshop Digital Broadcasting (September 2010)
7. Francisco, P.: The Netezza Data Appliance Architecture: A Platform for High Performance Data Warehousing and Analytics. Tech. rep., IBM (2011), `http://www.netezza.com/documents/whitepapers/Netezza_Appliance_Architecture_WP.pdf`
8. Kalte, H., Porrmann, M., Rückert, U.: System-on-Programmable-Chip Approach Enabling Online Fine-Grained 1D-Placement. In: Proceedings of the 11th Reconfigurable Architectures Workshop (RAW). pp. 141–146. New Mexico, USA (2004)
9. Kammeyer, K.D.: Nachrichtenübertragung. B.G. Teubner, Reihe Informationstechnik, Stuttgart, Deutschland, 4 edn. (Mar 2008)
10. Koch, D., Beckhoff, C., Teich, J.: ReCoBus-Builder– a Novel Tool and Technique to Build Statically and Dynamically Reconfigurable Systems for FPGAs. In: Proc. of Int. Conf. on Field-Progr. Logic and Applications (FPL). pp. 119–124 (Sep 2008)

11. Koch, D., Beckhoff, C., Teich, J.: Minimizing Internal Fragmentation by Fine-grained Two-dimensional Module Placement for Runtime Reconfigurable Systems. In: 17th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM). IEEE Computer Society, Napa, CA, USA (Apr 2009)
12. Koch, D., Beckhoff, C., Torresen, J.: Zero Logic Overhead Integration of Partially Reconfigurable Modules. In: Proceedings of the 23rd symposium on Integrated circuits and system design - SBCCI '10. p. 103. ACM Press (2010)
13. Koch, D., Torresen, J.: Routing Optimizations for Component-based System Design and Partial Run-time Reconfiguration on FPGAs. In: Proc. of Int. Conference on Field-Programmable Technology (ICFPT). pp. 460–464. IEEE (2010)
14. Krill, B., Ahmad, A., Amira, A., Rabah, H.: An Efficient FPGA-based Dynamic Partial Reconfiguration Design Flow and Environment for Image and Signal Processing IP Cores. Image Commun. 25, 377–387 (June 2010)
15. Lysaght, P., Blodget, B., Mason, J., Young, J., Bridgford, B.: Invited Paper: Enhanced Architectures, Design Methodologies and CAD Tools for Dynamic Reconfiguration of Xilinx FPGAs. In: 2006 International Conference on Field Programmable Logic and Applications (FPL). pp. 1–6. IEEE (2006)
16. Mak, T.S.T., Sedcole, N.P., Cheung, P.Y.K., Luk, W.: On-FPGA Communication Architectures and Design Factors. In: Proceedings of the 16th International Conference on Field Programmable Logic and Applications (FPL). pp. 1–8 (2006)
17. Mark Bourgeault: Alteras Partial Reconfiguration Flow (2011), available online: http://www.eecg.utoronto.ca/~jayar/FPGAseminar/FPGA_Bourgeault_June23_2011.pdf
18. Müller, R.: Data Stream Processing on Embedded Devices. Ph.D. thesis, ETH Züich (2010)
19. Münch, D.: Receive signal dependent adaption of an FPGA-based Software Defined Radio receiver system. Master's thesis, TUM (2011)
20. na, M.S., Patel, A., Madill, C., Nunes, D., Wang, D., Chow, P., Wittig, R.: MPI as a Programming Model for High-Performance Reconfigurable Computers. ACM Trans. Reconfigurable Technol. Syst. 3, 22:1–22:29 (November 2010)
21. Oetken, A., Wildermann, S., Teich, J., Koch, D.: A Bus-based SoC Architecture for Flexible Module Placement on Reconfigurable FPGAs. In: Proceedings of the International Conference on Field Programmable Logic and Applications (FPL). pp. 234–239. Milan, Italy (Aug 2010)
22. Schad, F., Steil, A.: Laboruntersuchung über Versorgungskriterien für eine UKW-FM Monoabstrahlung (2008)
23. Schlessman, J., Chen, C.Y., Wolf, W., Ozer, B., Fujino, K., Itoh, K.: Hardware/software co-design of an FPGA-based embedded tracking system. In: Proceedings of CVPRW. p. 123 (2006)
24. Sohanghpurwala, A.A., Athanas, P., Frangieh, T., Wood, A.: OpenPR: An Open-Source Partial-Reconfiguration Toolkit for Xilinx FPGAs. In: Proc. of the IEEE Int. Symp. on Parallel and Distr. Processing Works. (IPDPSW). pp. 228–235 (2011)
25. Toscher, S., Reinemann, T., Kasper, R.: An Adaptive FPGA-Based Mechatronic Control System Supporting Partial Reconfiguration of Controller Functionalities. In: Pro. of the 1st Conf. on Adaptive Hardw. and Syst. (AHS). pp. 225–228 (2006)
26. Wireless Innovation Forum: What is Software Defined Radio, available online: http://www.wirelessinnovation.org/assets/documents/SoftwareDefinedRadio.pdf
27. Wirthlin, M., Hutchings, B.: A Dynamic Instruction Set Computer. In: Proceedings IEEE Symposium on FPGAs for Custom Computing Machines. pp. 99–107
28. Xilinx Inc.: Two Flows for Partial Reconfiguration: Module Based or Difference Based (May 2002), available online: www.xilinx.com/support/documentation/application_notes/xapp290.pdf
29. Xilinx Inc.: Partial Reconfiguration User Guide (2011), rel 13.2

Parma 2012

3rd Workshop on Parallel Programming and Run-Time Management Techniques for Many-core Architectures

organized by

Alexandros Bartzas, National Technical University of Athens, Greece
Giovanni Agosta, Politecnico di Milano, Italy

The PARMA Workshop is jointly supported by the 2PARMA project
funded by the FP7-ICT Programme under the Objective Computing Systems

http://www.2parma.eu/

Message from Workshops Chairs

On behalf of the Organizing and Program Committee, it is our great pleasure to welcome you to PARMA 2012, the 3nd Workshop on Parallel Programming and Run-time Management Techniques for Many-core Architectures, which takes place in the Garching Campus of the TU Muenchen, Germany at February 29, 2012. PARMA 2012 is co-located with ARCS 2012 – Architecture of Computing Systems.

The current trend towards many-core architecture requires a global rethinking of software and hardware design approaches. The 2PARMA workshop focuses on parallel programming models, design space exploration and run-time resource management techniques to exploit the features of many-core processor architectures.

The Technical Program Committee had the challenging task of selecting the best contributions from many paper submissions. The program includes one keynote talk from PD Dr.-Ing. Michael Huebner, Karlsruhe Institute of Technology, two invited talks i) from Prof. Juergen Teich, University of Erlangen; ii) for Dr. Dr. Patricia Sagmeister, IBM, and nine regular oral papers. We also gratefully acknowledge and thank all authors who prepared and submitted papers and the participants who attended the conference.

Many dedicated volunteers have helped in making possible this very rich technical program, including the members of the Technical Program Committee and supporting staff and students from National Technical University of Athens. We cordially express our thanks for their assistance. In particular, we would like to thank Mr. Ioannis Koutras, who built and supported the PARMA 2012 website the last months.

The organization of the PARMA 2012 workshop would not have been possible without the help of the ARCS 2012 Workshop Chair Gero Muehl, University of Rostock, Germany.

Furthermore, the PARMA Workshop is jointly supported by the 2PARMA project funded by the FP7-ICT Programme under the Objective Computing Systems http://www.2parma.eu/

It is our most sincere wish that you find PARMA 2012 a high-quality workshop and we wish you enjoy your stay in Munich.

Alexandros Bartzas
Giovanni Agosta
PARMA 2012 Chairs

Organizing Committee

Workshop General Co-Chairs

Alexandros Bartzas, National Technical University of Athens, Greece

Giovanni Agosta, Politecnico di Milano, Italy

Program Co-Chairs

Vittorio Zaccaria, Politecnico di Milano, Italy

Torsten Kempf, ISS-RWTH Aachen University, Germany

Publicity Chair

Diana Göhringer, Fraunhofer Institute of Optronics, System Technologies and Image Exploitation, Germany

Publication Chair

Carlo Galuzzi, Delft University of Technology

Web Chair

Ioannis Koutras, National Technical University of Athens, Greece

Program Committee

Lars Bauer, Univ. of Karlsruhe, Germany

Mladen Berekovic, TU Braunschweig, Germany

Holger Blume, Univ. of Hannover, Germany

Rainer Buchty, Informatik Universität Karlsruhe, Germany

Ramon Canal, Universitat Politecnica de Catalunya, Spain

João Cardoso, U Porto, Portugal

Pedro Diniz, INESC-ID, Portugal

Kees Goossens, TU/e, The Netherlands

Michael Hübner, Karlsruhe Institute of Technology, Germany

Cédric Bastoul, University Paris-Sud, France

Stefano Crespi Reghizzi, POLIMI, Italy

William Fornaciari, POLIMI, Italy

Ahmed Jerraya, CEA, France

Tulika Mitra, National University of Singapore, Singapore

Alex Orailoglu, University of California, San Diego, USA

Gianluca Palermo, POLIMI, Italy

Yannis Papaefstathiou, TUC, Greece

Massimo Poncino, POLITO, Italy

Praveen Raghavan, IMEC, Belgium

Roel Wuyts, IMEC, Belgium

Cristina Silvano, POLIMI, Italy

Georgios Sirakoulis, Democritus Univ. Thrace, Greece

Dimitrios Soudris, National Technical Univ. Athens, Greece

Leonel Sousa, TU Lisbon, POrtugal

Sander Stuijk, TU Eindhoven, The Netherlands

Kari Tiensyrja, VTT, Finland

Stavros Tripakis, Verimag, France

Eugenio Villar, Universidad de Cantabria, Spain

Sotirios Xydis, National Technical Univ. Athens, Greece

Chantal Ykman-Couvreur, IMEC, Belgium

Efficient Memory Allocations on a Many-Core Accelerator*

Ioannis Koutras, Alexandros Bartzas, Dimitrios Soudris

School of Electrical and Computer Engineering
National Technical Unicersity of Athens
9, Iroon Polytechniou str.
157 80, Athens, Greece
{joko, alexis, dsoudris}@microlab.ntua.gr

Abstract: Memory management is one of the key challenges in the design of embedded systems where memory is a scarce resource. The problem scales disproportionally as new embedded systems incorporate many-core architectures where the cores have to struggle accessing an even more limited amount of resources. In this paper we present a way of creating custom memory allocators for many-core accelerators. We evaluated our approach in the P2012 platform, a many-core accelerator from ST. It is shown that a custom memory allocator created by our framework could save on average 62% of the total cycles spent on memory resource management when compared with the platform's current memory allocator without increasing the allocator's overhead.

1 Introduction

The current design trend in System-on-Chips (SoCs) utilizes heavily multiple processors and is therefore shifted towards the Multi-Processor SoC (MPSoC) design paradigm. As technology allows the integration of an aggressively increasing number of transistors, the concept of many-core computing steadily emerges, suggesting tens of processor cores integrated in one chip as a computing fabric [Bor07]. Without any question, memory management on such architectures could contribute notably in improving performance and mitigating the scalability bottleneck, as the processors struggle constantly to access data from shared memory locations. This bottleneck could have an even greater impact on many-core architectures designed for the embedded system context due to the small size of memories used in such hardware.

Applications developed for such systems are slowly adapting to this model while trying to exploit every possible resource by using data- and task-level parallelism. This leads to applications with highly dynamic behaviour and parallel execution of their tasks. This increased dynamism leads to unexpected memory footprint and fragmentation variations, which are difficult to be identified adequately at the design time. Developing dynamic multi-threaded applications using worst-case estimates for managing memory in a static

*This work is partially supported by the E.C funded FP7-ICT-2009-4-248716 2PARMA Project. Official Website: http://www.2parma.eu.

manner would impose severe overheads in memory footprint and power consumption. In order to avoid such type of costly over-estimations, developers are motivated to efficiently utilize dynamic memory [WJNB95].

The dynamic memory manager (DMM) handles memory requests that happen during the application's execution. Dynamic memory managers are responsible for organizing the dynamically allocated data in memory and also servicing the applications memory requests (allocation/de-allocation) at run-time [WJNB95, BMBW00]. In case of a memory request for allocation of a new object, the dynamic memory manager returns a pointer to the application, which points to the memory position of the allocated object. In C / C++ programming language, dynamic memory allocation is performed through `malloc()` / `new()` function calls. In case of a memory request for de-allocation of an already dynamically allocated object, the DMM returns to the application either a true or a false value in respect to success of the de-allocation process.

In this paper we study memory allocations on a many-core accelerator, namely Platform 2012, and propose a custom based memory management system to improve the performance of these operations.

The rest of the paper is organized as follows. In Section 2 we present previous work for memory management. Section 3 explains the P2012 hardware architecture, as well as the available programming models and execution patterns. Section 4.1 is dedicated to designing memory allocators for this platform and gives a brief overview of the current memory allocator. Experimental results are shown in Section 5. Finally, the conclusions and suggestions for future work are made in Section 6.

2 Related Work

Extensive research has been conducted for general-purpose dynamic memory management targeting both the single processor and the multi-processor domain. In [Kri99] a brief overview of the heap management is given and [WJNB95] discusses what policies and search algorithms are optimal for general-purpose use in single-processor systems. In [Iye93] authors are experimenting with maintaining multiple lists, a method which is more suited on multi-processor systems. In [VH99] a simple parallel allocator is implemented, which promises good scalability due to its simplicity, while [Mic04] proposes a way to reduce the synchronization overhead of dynamic memory allocation by using lock-free data structures. Authors in [LK99] benchmark various general-purposed memory allocators on typical server application workloads. Finally, the Hoard multi-threading memory allocator is presented in [BMBW00] aiming for low average fragmentation and little scaling overhead. Essentially, the inherent generality of existing DMMs eliminates the potential for hardware- and application-specific optimizations.

On the other hand, there is plenty of research work regarding memory allocation for specific hardware platforms, which can not be easily applied in other systems. A memory allocator which favours cache locality on specific SMP systems is proposed in [SAN06]. Authors in [BS10] study heap management in the Cell processor, a relevant hardware ar-

chitecture, but they do not handle shared memory; instead of this, the processing units have to handle their own, dedicated memory and they communicate with the system through explicit DMA calls, a limitation posed by the individual hardware platform.

Customized (i.e. application-specific) DMMs have also been proposed as an effective way either to improve performance [BZM01] or to reduce memory footprint [Ati06]. However, DMM customization has been studied only for single-threaded applications running on single-processor platforms. Recently, a systematic exploration strategy was proposed [XBA+10], which is performed at design-time in order to customize DMM services to the specific needs of a multi-threaded application. However, the evaluation was performed on a general-purpose machine using a general-purpose POSIX-compliant operating system, which provided the necessary synchronization primitives. In contrast, in this work we are targeting memory management on many-core accelerators with low-level synchronization primitives.

Overall, memory management specialized for minimal memory footprint on a massive number of processing elements is not fully explored in the current bibliography.

3 Architecture Case: ST Platform 2012

Platform 2012 (P2012) is a many-core accelerator developed by ST Microelectronics and CEA. The ultimate goal of P2012 is to fill the area and power efficiency gap between general-purpose embedded CPUs and fully hardwired application accelerators [SC11]. This goal makes this platform fully compliant with the scope of many-core accelerators in which we want to study the memory resource management.

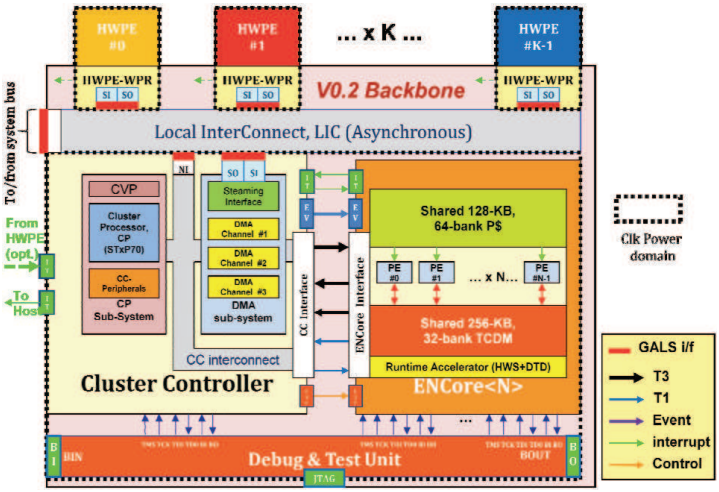


Figure 1: P2012 Cluster as depicted in [SC11]

Processing elements in P2012 are organized into clusters. The organization of a cluster is shown in Figure 1: Clusters are composed by several general-purpose processing elements of the STxP70-v4 architecture, called ENCore processors. There is an option to include specialized hardware IP cores, but this hardware feature, as well as others shown in the Figure 1, are out of the context of memory resource management. Each cluster is managed by a processor unit similar to the ENCore processors in terms of architecture, called cluster controller, and the whole accelerator (also called computing fabric) is managed by another similar processing element, called fabric controller. A first implementation of the platform contains 4 clusters with 16 processing elements, but it is expected to be even more scalable.

Regarding the memory hierarchy, P2012 adopted a multi-level one: There are three levels of memory, namely Level-1, 2 and 3. All the ENCore processors use the same memory map and can access every level. L1 is shared by all the processing elements of each cluster. It is a very fast, yet small (256 kB) memory. It is designed to be accessed in a uniform way, meaning that it is guaranteed to be accessed by ENCore processors in a fixed, low number of cycles regardless of the traffic. L2 is the memory which is shared among clusters in a NUMA way. It is unclear how many cycles it will take for a processing element to access it, as this depends on the traffic of the Network-on-Chip (NoC) which connects the clusters. The size of L2 is currently set to 1 MB, but there are configurations where L2 is completely omitted. Lastly, a portion of the host's memory is accessible as L3 memory with typical size of 256 MB. The penalty in execution is certainly big whenever a processing element tries to access this memory, so memory accesses from the fabric side to L3 should always be kept minimal.

3.1 P2012 Software Stack

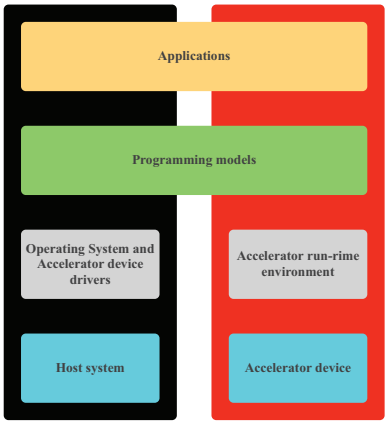


Figure 2: P2012 Software Stack

P2012 acts as an accelerator, so the existence of a host system is necessary. Figure 2 shows

the software stack required for deploying an application to an accelerator from a host. The application description and the programming models are the same between the host and accelerator. However, each one of them has a different run-time environment. From the host side is the Operating System (in P2012 Linux or Android are supported) and the accelerator device drivers, and from the accelerator side is the run-time environment of the accelerator. P2012 Software Development Kit (SDK) contains all the required tools of developing applications for P2012, as well as the source code of platform services up to some extent. A user community actively supports the SDK in [Min11].

The application execution is organized as follows: A Linux driver manages the communication between the host and the fabric, i.e. loading the application code from the host to the fabric and managing the P2012 resources from the host side. Runtime services are available by every cluster controller. They are responsible for the application deployment in each ENCore processor and they provide additionally the scheduling and the whole resource management of the cluster in terms of processor and memory resource allocation and de-allocation, as well as of power management. Runtime code is resident to P2012 cluster controllers and its API is used in order to develop programming models.

Applications may be developed currently in two programming models which are available and supported in P2012: Open Compute Language (OpenCL) and Native Programming Model (NPM). The first one is quickly adopted as industry's standard, as most of the giant hardware companies are active members to Khronos Compute Working Group which is responsible for the development of the official OpenCL specification. NPM is a more device-specific approach to program the platform. It handles concurrency as an Actor model. Other programming models can be implemented on top of NPM; P2012's OpenCL runtime on the P2012 platform is also built on top of it [SC11].

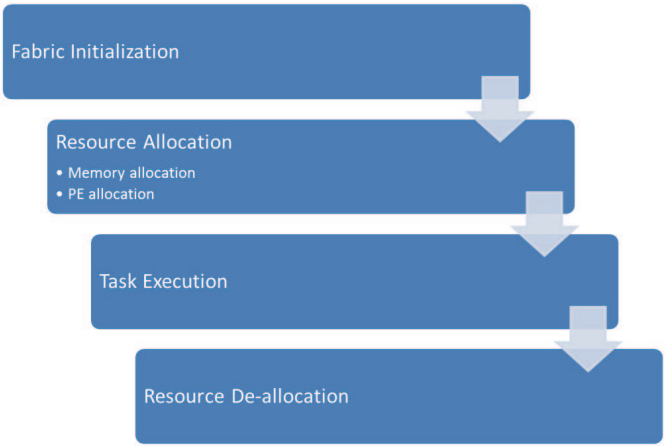


Figure 3: Executing an application on P2012

Both programming models utilize P2012 runtime in order to access resources and the low-level task scheduler. In fact, they both use the execution model as shown in Figure 3.

First, the host initializes the fabric and sends the necessary code to be executed. Resource allocation, such as task mapping to specific ENCore processors and memory allocations for processor stacks and data buffers, is the next step in the execution scheme. In fact, resource allocation should be completely finished before the task execution phase begins. During task execution there is no new memory allocation: even the new task instances use the memory space which was previously allocated to the initial identical task. After the end of every task on the ENCore processors, resource de-allocation takes place and P2012 is set off.

4 Creating a custom memory allocator for P2012

4.1 dmmlib Framework

dmmlib is a highly portable dynamic memory management library written in C. It allows developers to generate custom memory allocators by choosing the desired features and policies.

The framework provides custom implementations for dynamic memory allocation, re-allocation and de-allocation which could replace classic system calls, i.e. `malloc()`, `realloc()` and `free()`. The generated functions could be completely standalone if the developer knows a priori the total memory to be managed, or they could use OS calls to access more memory space such as `sbrk()` or `mmap()`. Multi-thread dynamic memory management can be supported by using `POSIX` mutexes or platform-specific synchronization primitives. The memory can be organized in a multiple number of heaps, each of which may contain or not lists of fixed-sized free blocks. There are several implementations of block organizations (singly, doubly linked lists etc.) and a broad variety of search block algorithms is supported (exact-, first-, best-, next-, good-fit etc.) Additionally, there is support for coalescing and splitting of memory blocks to prevent excessive fragmentation. Respective thresholds are provided at design time or runtime for both of the previous mechanisms. Finally, the library supports a rich set of statistics, such as the number of memory accesses and requested allocation sizes, which are fully available at run-time at the expense of some overhead on the metadata structures.

Some of the aforementioned features can only change on design time, so the developer will have to select them before generating the library's binary code. In a similar manner, other features can be changed during runtime without any need to recompile the library. Most of them are highly parameterizable either statically in design time, or dynamically through knobs at runtime. The perspective of tuning those knobs at runtime in order to create more adaptive allocators is explored more thoroughly in [XSBS11]. In any way, it should be noted that there is a clear trade-off between code size and customizability as the developer chooses to implement features and policies statically or not.

In the context of this work the dmmlib framework was integrated in P2012. The P2012 runtime is targeted, as this is considered the lowest level for resource management inside a P2012 cluster. As a result of the deep integration inside the runtime, every custom

memory allocator generated by dmmlib, supports both NPM and OpenCL. Another benefit of this integration that the usage of the optimized allocator is transparent to the application developers, i.e. not requiring any modifications in the application code.

4.2 Original memory allocator

The memory allocator used on P2012 is a variation of Doug Lea's family of dynamic memory allocators (dlmalloc) [Dou00]. The Lea allocator is considered one of the best overall memory allocators competing even with custom allocators [BZM02].

For every allocation call a memory block is given to the application, containing the requested space bounded by the block's size information. This helps separating each block and improves operations on the data layout such as coalescing blocks.

Free memory blocks are maintained on circular doubly linked lists: each free block contains pointers to its previous and next in the list block inside the space destined for application data. There are 128 of free lists containing blocks according to their size, e.g. there are lists for blocks of 16, 32 bytes and so on. The maximum number of the blocks which can be included in these lists is getting lowered logarithmically as the size of blocks is being raised.

Blocks are sorted and searched through as in LRU or FIFO allocation with support to coalesce and split blocks.

4.3 Choosing options for a dmmlib memory allocator on P2012

dlmalloc is a very efficient general-purpose memory allocator, but it is not optimized for the current application execution scheme on P2012. In order to investigate further the integration of dmmlib in P2012, a custom memory allocator was generated with features and options we considered optimal for this platform.

The main reasoning for most of the choices was to achieve simplicity as the target platform is an embedded one. More specifically, we have chosen block structure identical to the dlmalloc's one in order to prevent metadata overhead on this low-memory environment. We have tried to achieve simplicity in the lists structure. Thus, we are using circular singly linked lists instead of using circular doubly linked lists, achieving a reduction in memory footprint compared to the original allocator. Furthermore, instead of using a great number of lists for free blocks, we choose to use only one such list. This simplifies a lot the procedure of finding an appropriate free memory block big enough to accommodate the allocation request. Finally, we have disabled coalescing and splitting, further simplifying the `malloc()` and `free()` calls, thus resulting in performance gains.

The resulted memory allocator is an extremely simple one, which fits in the current execution flow of P2012. It would cause fragmentation issues on applications which require executing various tasks on different times, but the current implementations of program-

ming models do not create such tasks or handle the resources this way.

5 Experimental Results

5.1 Experimental Setup

In order to compare the proposed custom memory allocator with the current one, it is necessary to run cycle-accurate simulations for multiple applications running on P2012. RTL simulation was chosen since NoC traffic and memory access timings are the most accurate ones in this type of simulation.

RTL simulation is one of the most accurate ones, but it is also one of the slowest. All of the tested applications use one cluster, so simulating just one cluster is adequate for the benchmarking needs. Even then, simulating one cluster in RTL requires a tremendous amount of time even when running simple applications. In order to improve the simulation time, the applications were simplified: Since all of the memory allocations and de-allocations take place in the cluster controller, we do not need to enable the ENCore processors. The memory allocation and de-allocation traces of applications were extracted and simple testbenches were created, where only these types of operations were used.

We will show results from the following applications, all of which are available through the official P2012 SDK [Min11]:

Integral Calculates the integral of an input matrix.

Gaussian Applies a Gaussian blur effect on an input picture.

FAST Features from Accelerated Segment Test, a corner detection algorithm implementation used on computer vision.

Matrix Performs matrix multiplication of an input matrix with a constant matrix.

5.2 Allocators' Evaluation

In terms of code size the simplicity of the generated allocator makes a big difference compared to Doug Lea's: 60 kB versus 88 kB, a 32% improvement on the code size.

Metadata (i.e. internal heap organization and usage information and statistics) is essentially the same as the block structure is kept intact: the block size information appears in the memory twice, before and after the space for application data. This is equivalent to 64 bits per block, as the size information in dmmlib takes up the size of a word and P2012's ENCore processors use 32-bit words.

The speed-up of using the custom memory allocator over Doug Lea's is also substantial. Figure 4 shows an average improvement of 60% percent in cycles for executing all the

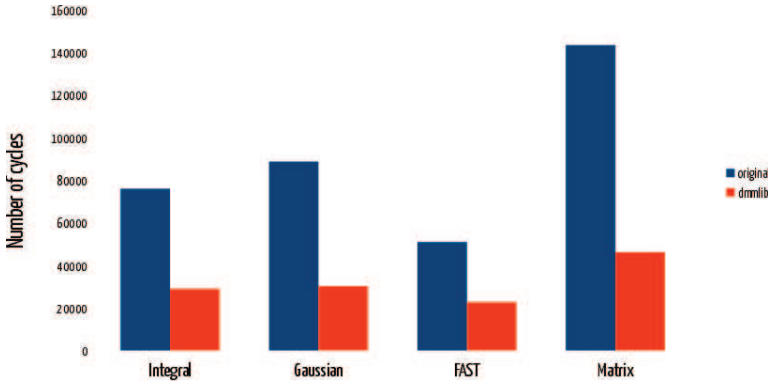


Figure 4: Comparison of the performance of the two memory allocators in various applications

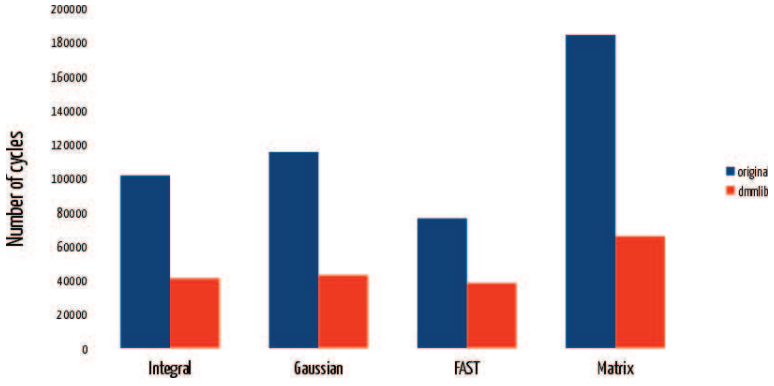


Figure 5: Comparison of the performance of the two memory allocators in various applications with de-allocation included

memory allocations required by each application. In Figure 5 the memory de-allocations have been included in the total time, but they have very little affect on the final outcome.

Tables 1 and 2 present a clear view on the average and the max time each allocator spends for one event and the respective differences.

The trend we could extract from these timings is that applications which use many small blocks (e.g. the Matrix application) are more benefited by the customized memory allocator of dmmlib.

Table 1: Average and Maximum values of Allocation Times

App.	Allocator	Avg. Cycles	Avg. Diff.	Max. Cycles	Max. Diff.
Integral	initial	1224	-62.54%	2278	-66.11%
	custom	329		772	
Gaussian	initial	886	-62.86%	2293	-66.33%
	custom	466		772	
FAST	initial	737	-55.5%	1205	-49.88%
	custom	328		604	
Matrix	initial	1462	-67.85%	2334	-67.65%
	custom	470		755	

Table 2: Average and Maximum values of De-allocation Times

App.	De-allocation	Avg. Cycles	Avg. Diff.	Max. Cycles	Max. Diff.
Integral	initial	412	-52.91%	603	-38.97%
	custom	194		368	
Gaussian	initial	409	-52.57%	603	-66.33%
	custom	194		368	
FAST	initial	367	-38.69%	550	-12.55%
	custom	225		481	
Matrix	initial	416	-51.92%	636	-41.04%
	custom	200		375	

5.3 Estimating memory allocations on more dynamic task execution cases

It is quite certain that it will be possible to have more complex task execution flows as P2012 matures. Resources would then have to be de-allocated and re-allocated in a more fine-grained way, i.e. between task executions and not just after the last task execution of the application. In respect with this, we need to test the memory allocators in use cases where data is allocated and de-allocated in a more dynamic way.

There are still not any available applications matching this behavior. What we propose in order to estimate the performance of memory allocators in such cases is to use the traces of the previous applications and to randomize the spots of data de-allocations. The allocation dependencies will still be respected (e.g. we can not de-allocate memory which is not allocated before) and the memory requests will be realistic enough.

For this part we have used the FAST and Matrix applications since the tasks of these applications is more near to real application kernels. In Figure 6 there is an overview of the total cycles spent on memory management operations. The custom memory allocator still outperforms the original one, but the gap is getting much more narrow. Table 3 and 4 show the average and worst cases of timings for allocating and de-allocating data. It should be noted that the number of allocations for the FAST application are less than the number of allocations for the Matrix, as the FAST application performs fine-grained memory allocations inside the kernel. As a result, the customized memory allocator seems

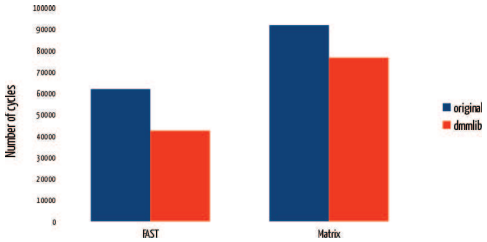


Figure 6: Comparison of the performance of the two memory allocators in dynamic cases

Table 3: Average and Maximum values of Allocation Times for more dynamic cases

App.	Allocation	Avg. Cycles	Avg. Diff.	Max. cycles	Max. Diff.
FAST	initial	549	-29.14%	1153	-32.35%
	custom	389		780	
Matrix	initial	583	-6%	1184	-9.62%
	custom	548		1310	

to perform better in the case of the FAST application over the Matrix one.

6 Conclusions and Future Work

Our approach generating customizable memory allocators allows us to speed up the exploration process of finding an efficient memory allocator for many-core architectures such as P2012. The custom memory allocator we proposed for P2012 achieves a speedup of 2.6x in cycles without compromising the metadata overhead introduced by the original memory allocator of P2012 runtime.

As the P2012 evolves and other many-core accelerators are introduced, we can expect multiple use-case scenarios and increased interaction between these systems and their surrounding environment. With regards to that, new features for memory management could be introduced, such as runtime, adaptive control of features like coalescing and splitting, support to allocate and de-allocate memory regions by the worker processors as well, and ability to perform fine-grained memory management on dedicated memory regions.

Table 4: Average and Maximum values of De-allocation Times for more dynamic cases

App.	De-allocation	Avg. Cycles	Avg. Diff.	Max. Cycles	Max. Diff.
FAST	initial	347	-38.69%	565	-12.55%
	custom	225		481	
Matrix	initial	353	-34.28%	617	-21.07%
	custom	232		487	

References

[Ati06] D. a.l. Atienza. Systematic dynamic memory management design methodology for reduced memory footprint. *ACM TODAES*, 11(2):465–489, 2006.

[BMBW00] E.D. Berger, K.S. McKinley, R.D. Blumofe, and P.R. Wilson. Hoard: A scalable memory allocator for multithreaded applications. *ACM SIGPLAN Notices*, 35(11):117–128, 2000.

[Bor07] Shekhar Borkar. Thousand core chips: a technology perspective. In *Proceedings of DAC*, pages 746–749, New York, NY, USA, 2007. ACM.

[BS10] K. Bai and A. Shrivastava. Heap data management for limited local memory (llm) multi-core processors. In *Proc. of CODES+ISSS*, pages 317–325. IEEE, 2010.

[BZM01] Emery D. Berger, Benjamin G. Zorn, and Kathryn S. McKinley. Composing high-performance memory allocators. *ACM SIGPLAN Notices*, 36(5):114–124, May 2001.

[BZM02] Emery D. Berger, Benjamin G. Zorn, and Kathryn S. McKinley. Reconsidering custom memory allocation. In *Proc. of OOPSLA*, page 1. ACM Press, 2002.

[Dou00] L. Doug. A Memory Allocator. http://gee.cs.oswego.edu/dl/html/malloc.html, April 2000.

[Iye93] A. Iyengar. Parallel dynamic storage allocation algorithms. In *Proc. of PDPS*, pages 82–91. IEEE, 1993.

[Kri99] M.R. Krishnan. Heap: Pleasures and pains. *Microsoft Developer Newsletter*, 1999.

[LK99] P.-. Larson and M. Krishnan. Memory allocation for long-running server applications. *ACM SIGPLAN Notices*, 34(3):176–185, March 1999.

[Mic04] Maged M. Michael. Scalable lock-free dynamic memory allocation. *ACM SIGPLAN Notices*, 39(6):35, June 2004.

[Min11] Minalogic. STMicroelectronics Platform 2012 users‘ Community [Restricted Access]. https://minalogic.net/projects/p2012comm/, November 2011.

[SAN06] S. Schneider, C.D. Antonopoulos, and D.S. Nikolopoulos. Scalable locality-conscious multithreaded memory allocation. In *Proc. of the 5th International Symposium on Memory management*, pages 84–94. ACM, 2006.

[SC11] STM and CEA. Platform 2012: A Many-core programmable accelerator for Ultra-Efficient Embedded Computing in Nanometer Technology. White paper, 2011.

[VH99] V.Y. Vee and W.J. Hsu. A scalable and efficient storage allocator on shared-memory multiprocessors. In *Proc. of I-SPAN*, pages 230–235. IEEE, 1999.

[WJNB95] P.R. Wilson, M.S. Johnstone, M. Neely, and D. Boles. Dynamic storage allocation: A survey and critical review. *Lectures in Computer Science*, pages 1–78, 1995.

[XBA+10] S. Xydis, A. Bartzas, I. Anagnostopoulos, D. Soudris, and K. Pekmestzi. Custom multi-threaded Dynamic Memory Management for Multiprocessor System-on-Chip platforms. In *Proc. of SAMOS*, pages 102–109. IEEE, 2010.

[XSBS11] S. Xydis, I. Stamelakos, A. Bartzas, and D. Soudris. Runtime Tuning of Dynamic Memory Management For Mitigating Footprint-Fragmentation Variations. In *Proc. of ARCS*. VDE Verlag, 2011.

Improving Cache Locality for Ray Casting with CUDA*

Yuki Sugimoto, Fumihiko Ino, and Kenichi Hagihara

Graduate School of Information Science and Technology
Osaka University
1-5 Yamadaoka, Suita, Osaka 565-0871, Japan
{y-sugimt,ino}@ist.osaka-u.ac.jp

Abstract: In this paper, we present an acceleration method for texture-based ray casting on the compute unified device architecture (CUDA) compatible graphics processing unit (GPU). Since ray casting is a memory-intensive application, our method increases the hit rate of the texture cache during rendering. To achieve this, our method dynamically selects the width and height of thread blocks (TBs) such that each warp, which is a series of 32 threads simultaneously processed on the GPU, can achieve high data locality for specific viewpoints. The objective of this selection is to allow every warp rather than every thread to access data with a small stride, because the GPU executes multiple threads at the same time. In experiments using a GeForce GTX 480 card (i.e., the latest Fermi architecture), we find that the speedup of our method ranges from a factor of 1.0 to that of 4.0, depending on viewpoints. We think that optimizing the shape of TBs is important to achieve more cache hits in the highly-threaded CUDA hardware.

1 Introduction

Ray casting [Lev88] is a visualization technique for intuitive understanding of three-dimensional (3-D) objects. For example, this technique is useful to analyze not only computed tomography (CT) images in medical area [TIH03] but also simulation results in computational fluid dynamics [NIH08]. In ray casting, the voxel values of the volume are accumulated into pixel values on the screen. To do this, a ray is generated from the viewpoint to each pixel, and then values of penetrated voxels are sampled at regular intervals along the ray for accumulation. Thus, the accumulation is accomplished from 3-D space to 2-D space. During this accumulation procedure, voxel values can be reused only within neighboring region. Therefore, ray casting is a memory-intensive application rather than a compute-intensive application.

To deal with this large amount of memory access, many renderers [NIH08, KW03, RV06] were implemented using the graphics processing unit (GPU) [MM05], which is an ac-

*This work was partly supported by JSPS Grant-in-Aid for Scientific Research (B)(23300007) and Young Researchers (B)(23700057)

celerator for graphics applications. The memory bandwidth of the GPU is an order of magnitude higher than that of the CPU. Furthermore, this architecture is capable of running thousands of lightweight threads in parallel, which are useful to hide memory latency with data-independent computation. Using this accelerator, the accumulation procedure can be easily parallelized because there is no data dependence between different rays (i.e., different pixels). The volume data is typically loaded as a 3-D texture to interpolate voxel values using texture mapping hardware of the GPU. This hardware has a cache mechanism to reduce the latency of data access for acceleration. Consequently, the rendering performance can be increased by maximizing the locality of references.

In this paper, we present a view-dependent method for increasing the hit rate of the texture cache, aiming at accelerating texture-based ray casting [HS89]. To achieve this, our method maximizes the data locality by dynamically selecting the width and height of TBs according to the geometrical relationship between the viewpoint and the volume axes. The shape of TBs is selected such that a group of threads called warp [NVI10] can access data with a small stride. Since threads in the same warp are simultaneously processed on the GPU, such parallel threads have to maximize the locality of references. Our method currently works with the compute unified device architecture (CUDA) [NVI10], which is a programming framework for the NVIDIA GPU.

2 GPU-based Volume Rendering

2.1 Compute Unified Device Architecture (CUDA)

The CUDA-compatible hardware [NVI10] consists of hundreds of CUDA cores structured in a hierarchy. The hardware has tens of streaming multiprocessors (SMs), each containing 8 or 32 CUDA cores depending on the generation. Using these cores, thousands of threads are executed in a single-instruction, multiple-thread (SIMT) fashion [NVI10]. This highly-threaded architecture is designed to overlap memory latency with computation.

To achieve efficient overlap, threads are classified into data independent groups, namely *thread blocks* (TBs). Therefore, more TBs should be resided and processed together on the SM. Since there is no data dependence between TBs, such concurrent TBs contribute to have more flexibility for efficient scheduling of threads. Each resident TB is further broken into groups of 32 consecutive threads called *warps*. A warp is the minimum scheduling unit managed by the SM.

Threads can be identified using a 2-D index, forming a 2-D TB. The TB shape $w \times h$, where w and h be the width and the height of TBs, respectively, can be specified by an argument to the kernel function, which runs on the GPU. On the other hand, the warp shape $p \times q$ cannot be directly specified by the program, where p and q represent the width and the height of warps, respectively. Since threads in a warp have consecutive indexes, the warp shape is automatically determined by the TB shape. The execution order of warps is dynamically determined by the warp scheduler, which cannot be controlled by the program.

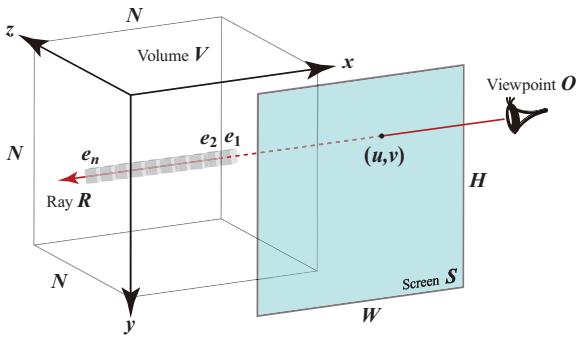


Figure 1: Geometry of ray casting. Pixel values are computed by accumulating color and opacity values of voxels penetrated by a ray from the viewpoint.

2.2 Ray Casting

Figure 1 illustrates the geometry used for ray casting [Lev88]. Let V be the volume to be rendered from the viewpoint O. We consider a cubic volume of $N \times N \times N$ voxels, where N represents the volume size. We assume that each voxel has a scalar data associated with color and opacity values. Let x, y, and z be elements of the voxel coordinates.

The ray casting technique casts a ray R from the viewpoint O to every pixel (u, v) on the screen S, where $1 \leq u \leq W$ and $1 \leq v \leq H$. W and H here represent the width and the height of the screen S. Ray R penetrates voxels in the volume, so that the value $S(u, v)$ of pixel (u, v) is computed by accumulating color and opacity values of penetrated voxels in front-to-back order. This accumulation is done at regular intervals along ray R as follows:

$$S(u,v) = \sum_{i=1}^{n} \left(\alpha(e_i) c(e_i) \prod_{j=0}^{i-1} (1 - \alpha(e_j)) \right), \tag{1}$$

where e_i represents the i-th voxel penetrated by ray R, n represents the number of penetrated voxels, $c(e_i)$ and $\alpha(e_i)$ represent the color and the opacity of voxel e_i, respectively, and $\alpha(e_0) = 0$.

2.3 Texture-based Rendering with CUDA

Eq. (1) indicates that different pixel values can be computed in parallel because there is no data dependence between them. Consequently, the computation of a pixel is assigned to a thread in typical renderers. A screen of $W \times H$ pixels can then be rendered by WH threads, which compose $\lceil W/w \rceil \times \lceil H/h \rceil$ TBs. Using this parallel scheme, voxels are accessed in front-to-back order.

Since rays do not always penetrate the center of voxels, voxel values must be interpo-

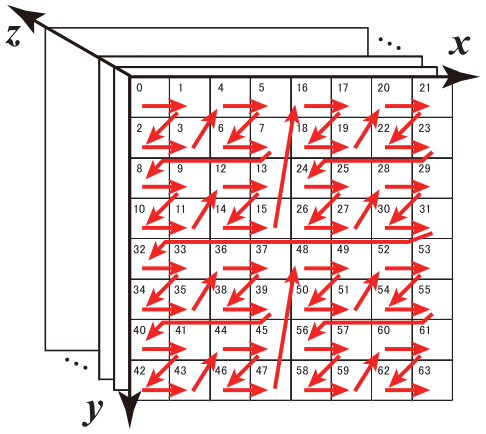


Figure 2: Organization of a 3-D texture in CUDA. A 3-D texture consists of a bunch of 2-D slices optimized for 2-D spatial locality via a z-order curve [Mor66]. A series of red arrows represents the sequence of physical memory address in a 2-D slice. The physical address is shown in each texel's upper left corner.

lated before accumulation. To accelerate this interpolation, many implementations employ texture-based rendering [HS89], which performs interpolation using texture mapping hardware of the GPU. Thus, the volume is accessed via a 3-D texture to take advantage of hardware accelerated interpolation.

3 Texture Memory Organization

Figure 2 shows how the GPU maps a logical address space onto a physical memory space in a 3-D texture [MM05, PF05]. As shown in this figure, a 3-D texture consists of a bunch of 2-D slices. Each slice is further optimized for 2-D spatial locality via a z-order curve [Mor66], as illustrated in a sequence of red arrows in Fig. 2. The z-order curve has a recursive hierarchy, so that a z-ordered block at the l-th level of hierarchy contains a 2-D slice of $2^l \times 2^l$ texels, where $1 \leq l \leq \lceil \log N \rceil$ (see Fig. 3). For simplicity, we assume N being a power of two (i.e., $N = 2^l$) in the following discussion.

Although the z-order curve is optimized for 2-D spatial locality, the physical stride between two adjacent voxels is not uniform in this data structure. To investigate this issue in more detail, let us consider accessing two adjacent voxels e_i and e_{i+1}. The stride between these voxels can then be classified into two groups depending on their coordinates:

1. The adjacent voxels have different z. In this case, voxels e_i and e_{i+1} exist on two adjacent slices. These voxels can be accessed with a stride of N^2, because they have the same x and y.

2. The adjacent voxels have different y or x. In these cases, voxels e_i and e_{i+1} exist

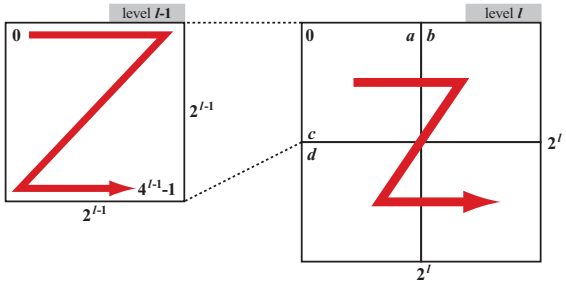


Figure 3: Hierarchical structure of a z-order curve. A block at the l-th level contains four internal blocks of the $(l-1)$-th level. The maximum stride appears between these internal blocks: between texels a and b along the horizontal axis, and between texels c and d along the vertical axis. The physical index of texels b and d are 4^{l-1} and $2 \cdot 4^{l-1}$, respectively. The physical index of texels a and c are $\sum_{k=0}^{l-2} 4^k$ and $2 \cdot \sum_{k=0}^{l-2} 4^k$, respectively.

on the same slice. The stride between them varies according to their location on the slice. For example, the strides along the x-axis range from 1 to 11 in Fig. 2. However, the maximum stride at the l-th level of hierarchy appears between adjacent blocks of the $(l-1)$-th level, as shown in Fig. 3. The maximum stride along the x-axis can be given by $(2 \cdot 4^{l-1} + 1)/6$ while that along the y-axis can be given by $(2 \cdot 4^{l-1} + 1)/3$. Since $N = 2^l$, voxels along the x-axis and the y-axis can be accessed with a stride of $(N^2 + 2)/6$ and that of $(N^2 + 2)/3$, respectively.

In summary, the x-axis, y-axis, and z-axis have a different stride between adjacent voxels, and their ratio can be approximated by $1 : 2 : 6$. Therefore, it is better to access voxels along the x-axis in order to achieve more cache hits.

4 Proposed Method

In this section, we describe our acceleration method that selects the TB shape $w \times h$ during rendering. We first explain our acceleration strategy, and then present how our method selects the TB shape according to the strategy.

4.1 Acceleration Strategy

As we mentioned in Section 1, our method maximizes the locality of references. To achieve this, we focus on four points as follows:

1. The SM processes threads in the same warp at the same time.
2. Each volume axis has a different stride between adjacent voxels.

3. The TB shape $w \times h$ determines the warp shape $p \times q$. For details, see [NVI10].

4. The number of resident TBs should be maximized to take advantage of the highly-threaded GPU architecture.

The first point motivates us to optimize the memory access pattern of a warp rather than that of a thread. This point is a unique feature owing to the highly-threaded GPU architecture. On earlier acceleration systems such as cluster systems [TIH03, MIH04], optimization is successfully done for a single process (i.e., a single ray). In contrast, we emphasize optimization of a warp (i.e., a ray frustum) as a key acceleration strategy for the GPU. Thus, we must investigate the memory access pattern that can be caused by a warp. Since voxels are sampled at regular intervals from the viewpoint, a warp accesses voxels on the surface of a sphere. For simplicity, we assume that this spherical surface can be approximated with a plane. Under this approximation, a warp accesses voxels on a plane that are parallel to the screen.

With respect to the second point, voxels should be always accessed along the x-axis, which has the smallest stride among the volume axes. However, this is not a practical solution because the volume can be rendered from an arbitrary viewpoint. For example, the x-axis can be rendered as a vertical line on the screen, but it can appear as a horizontal line with a different viewpoint. Thus, the volume axes have different appearance on the screen, depending on the location of the viewpoint (see Fig. 4). Therefore, we have determined to give priority to the volume axes: voxels should be accessed in the order of x, y, and z to have smaller strides. Clearly, this priority should be implemented as per-warp order instead of per-thread order. Therefore, we optimize the warp shape to realize the prioritization.

The third point plays the key role in realizing the prioritized access mentioned above. As we mentioned in Section 2.1, the warp shape $p \times q$ is determined by the TB shape $w \times h$. Table 1 shows their relationship when using the TB size wh of 128. This table indicates that horizontal warps (i.e., $p > q$) are generated if $w \geq 8$. Otherwise, vertical warps (i.e., $p < q$) are generated. The warp size $p \times q$ must be selected such that each warp can access voxels in the order of x, y, and z. For example, vertical warps are better than horizontal warps if the x-axis appears as a vertical line on the screen. In this case, vertical warps are allowed to access voxels with smaller strides than horizontal warps.

The last point determines the size wh of TBs. As we mentioned in Section 2.1, the number of resident TBs should be maximized to efficiently hide memory latency with computation. Due to the limitation of available resources, up to 8 TBs can be processed on the SM at a time [NVI10]. Similarly, up to 1536 threads can be resident on the SM. Therefore, the TB size wh must be $wh \leq 192$ to maximize resident TBs on the SM. In addition, the TB size wh must be a multiple of the warp size (i.e., 32) to avoid cores from being idle during SIMT execution. Since the number of resident TB depends on the amount of resource consumption, we compiled our rendering kernel with $wh = 192$, 160, and so on. We then found that $wh = 128$ is the maximum TB size that can run 8 TBs on the SM. Thus, our kernel runs with $wh = 128$.

Table 1: Relationship between TB shape $w \times h$ and warp shape $p \times q$. Values are presented for the TB size wh of 128. Horizontal warps (i.e., $p > q$) are generated if $w \geq 8$. Otherwise, vertical warps are generated.

TB shape $w \times h$	Warp shape $p \times q$	Aspect ratio of warp
1×128	1×32	$1 : 32$
2×64	2×16	$1 : 8$
4×32	4×8	$1 : 2$
8×16	8×4	$2 : 1$
16×8	16×2	$8 : 1$
32×4	32×1	$32 : 1$
64×2	32×1	$32 : 1$
128×1	32×1	$32 : 1$

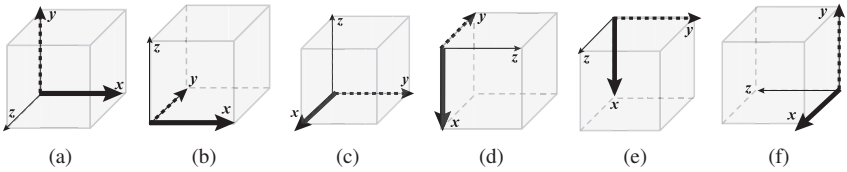


Figure 4: Geometrical relationship between the viewpoint and the volume axes. Each of subfigures (a)–(f) corresponds to one of six representative viewpoints. In these viewpoints, the x-axis can be parallel to one of the horizontal, the vertical, and the depth directions. The x-axis and the z-axis have the smallest stride and the largest stride among the volume axes, respectively.

4.2 Selection of Thread Block Shape

Our method selects the TB shape $w \times h$ according to the geometrical relationship between the viewpoint and the volume axes. For simplicity, we consider here six representative viewpoints, which make two of the volume axes parallel to the screen axes. Figure 4 shows six images rendered with the representative viewpoints. Given the viewpoint O, the TB shape is selected in the following three steps:

1. Plane detection. Our method detects the plane most parallel to the screen. For example, the xy-plane is such a parallel plane in Figs. 4(a) and 4(e). According to our approximation, voxels on a parallel plane are simultaneously accessed by a warp.

2. Primary axis detection. The primary axis with a smaller stride is selected from two axes that create the parallel plane. For example, the yz-plane is the parallel plane in Figs. 4(c) and 4(f), so that the y-axis is selected as the primary axis.

3. TB shape selection. The TB shape is selected according to the direction of the primary axis rendered on the screen. Vertical and horizontal warps are selected if

the primary axis is rendered in a vertical line and in a horizontal line on the screen, respectively. Our method currently uses the TB shape of $w \times h = 32 \times 4$ for horizontal warps and that of 1×128 for vertical warps. For example, the TB shape of 32×4 is selected for viewpoints in Figs. 4(a), 4(b), and 4(c), because the primary axis is rendered in a horizontal line from these viewpoints.

5 Experimental Results

To evaluate our method in terms of the rendering performance, we measured the frame rate and the hit rate of the texture cache. For experiments, we used a desktop PC equipped with a GeForce 480 GTX card. Our machine runs with Windows 7, CUDA 3.2 [NVI10], and graphics driver 260.61. The cache hit rate was measured by CUDA Visual Profiler.

As a rendering implementation, we used a sample code distributed with CUDA SDK. This code originally uses a texture to refer a color map table, which associates color and opacity values with each voxel. Since references to this table results in perturbation of cache behavior, we modified the code such that it stores the table in shared memory. Thus, the modified code uses textures only for the volume data. We used $N = W = H = 1024$ for performance evaluation. The volume consists of 8-bit data.

The volume data was rendered using six viewpoints (a)–(f) presented in Fig. 4. Note that these viewpoints are in symmetric positions. We used symmetric viewpoints, because threads are allowed to have the same workload despite the difference of viewpoints. That is, the same number of voxels is accessed for each viewpoint, but with a different order. This is necessary to avoid misunderstandings caused by asymmetric viewpoints, which assign different workloads to threads. Due to the same reason, optimization techniques such as early ray termination and empty space skipping [Lev90] are not used during experiments.

Figure 5 shows the frame rates of our dynamic method and a static method. The static method uses a fixed shape $w \times h = 16 \times 16$ (i.e., $p \times q = 16 \times 2$) for arbitrary viewpoints. For viewpoints (b), (d), (e), and (f), our method achieves higher frame rates than the static method. The speedup over the static method reaches 1.1, 4.0, 1.3, and 1.5 for viewpoints (b), (d), (e), and (f), respectively. In particular, the frame rate for viewpoint (d) increases from 11.6 to 46.6 fps, with increasing the cache hit rate from 51.2% to 73.9%. On the contrary, there is no significant difference for viewpoints (a) and (c). This is due to the warp shape used in the static method. For these viewpoints, the static method generates horizontal warps, as our method does. Therefore, our method cannot increase the cache hit rate for these viewpoints.

Figure 6 explains how the cache hit rate determines the frame rate. We measured both rates using eight different TB shapes, ranging from $w \times h = 1 \times 128$ to 128×1. Each frame rate is the average of ten trials. As shown in Fig. 6, higher frame rates are obtained when higher cache hit rates are achieved. For instance, the highest frame rate of 50.8 fps is observed when the cache hit rate reaches 73.8%. In contrast, the lowest frame rate of 5.2 fps results in a cache hit rate of 4.2%. Thus, the frame rate is mainly determined by

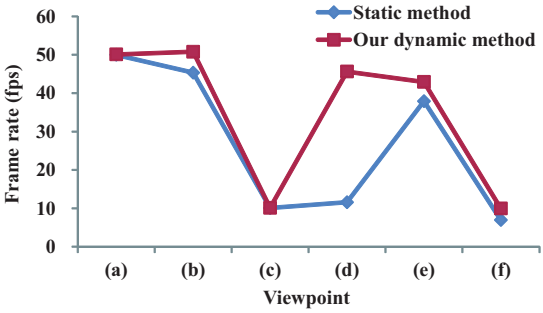


Figure 5: Comparison of frame rates between our dynamic method and the static method. The former uses the TB shape $w \times h = 32 \times 4$ for viewpoints (a)–(c) and $w \times h = 1 \times 128$ for viewpoints (d)–(f). The latter uses the default shape $w \times h = 16 \times 16$ for arbitrary viewpoints.

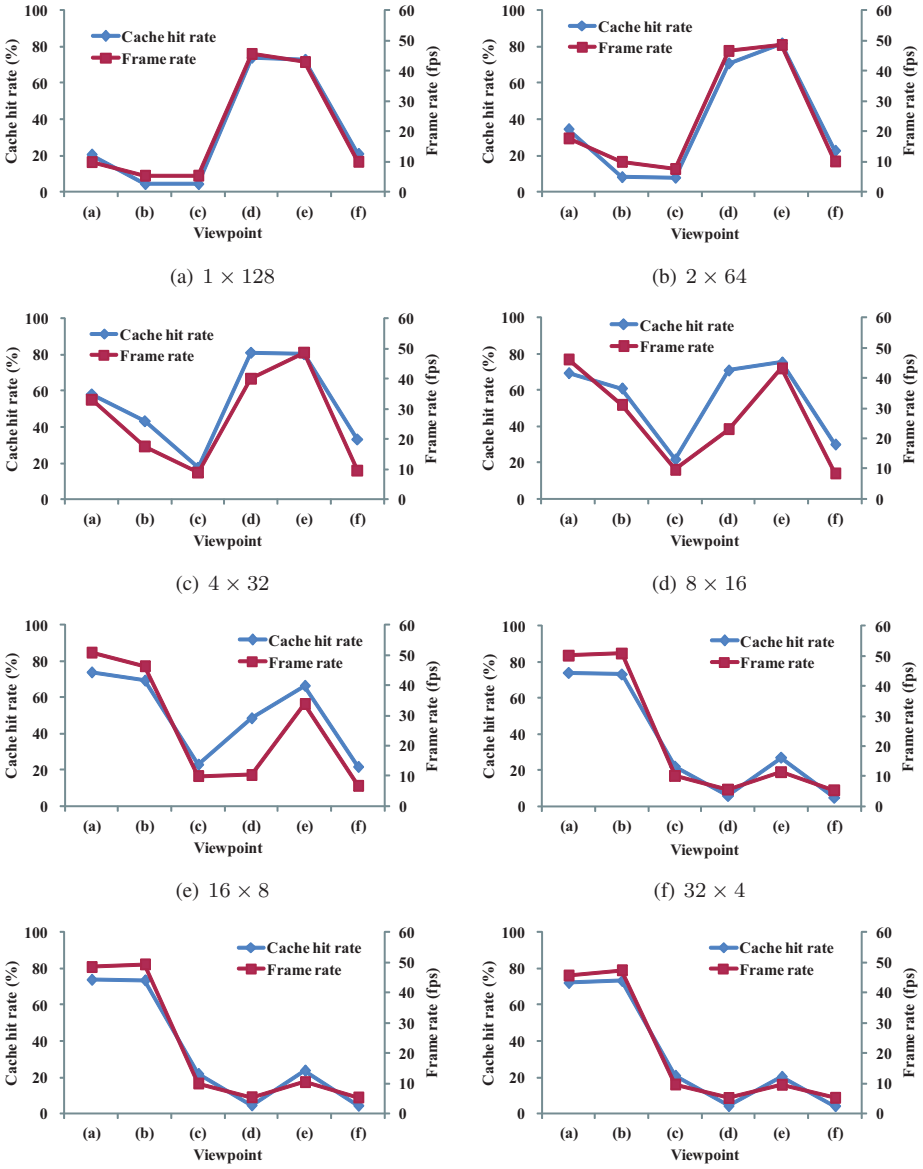
the cache hit rate.

Another important behavior in Fig. 6 is that all of the eight TB shapes have a wide range of cache hit rates ranging approximately from 5% to 80%. This behavior indicates that a single shape of TBs is not sufficient to obtain higher cache hit rates for all viewpoints. Therefore, it is better to change the TB shape according to the location of the viewpoint, as we do in our method.

We next investigate the relationship between the TB shape and the frame rate. Figure 7 shows frame rates with different TB shapes, ranging from $w \times h = 1 \times 128$ to 128×1. The results are classified into two groups: (1) viewpoints (a)–(c), which have smaller strides for horizontal warps; and (2) viewpoints (d)–(f), which have smaller strides for vertical warps. As we mentioned in Section 4.1, the warp size varies from $p \times q = 1 \times 32$ to 32×1, according to the TB shape $w \times h$ (see Table 1). Recall that horizontal warps are generated if $w \geq 8$.

Figure 7(a) indicates that horizontal warps rather than vertical warps yield high frame rates. In contrast, Fig. 7(b) shows that vertical warps achieve higher frame rates than horizontal warps. Actually, these figures are in a symmetric relation. A vertical symmetry axis exists between $w \times h = 4 \times 32$ and 8×16 (i.e., between $p \times q = 4 \times 8$ and 8×4). For example, viewpoints (b) and (d) have a cross point on the vertical symmetry axis, and both have the xz-plane as a parallel plane. Therefore, it is better to change the TB shape at the symmetry axis, which determines the warp shape to be vertical or horizontal. This dynamic optimization is exactly what our method implements.

Although our method optimizes the TB shape, the frame rates for viewpoints (c) and (f) result in lower values. When these viewpoints are used, the x-axis is parallel to the depth direction, as shown in Figs. 4(c) and 4(f). Therefore, voxels are always accessed with a large stride, which decreases the cache hit rate to at most 33.1%, as shown in Fig. 6. These results also imply that the capacity of the texture cache is not large enough to deal with $N = 1024$. Actually, the cache hit rate ranges from 33.2% to 69.2% if a smaller volume of $N = 512$ is rendered with viewpoints (c) and (f).



(a) 1×128 (b) 2×64

(c) 4×32 (d) 8×16

(e) 16×8 (f) 32×4

(g) 64×2 (h) 128×1

Figure 6: The frame rate and the cache hit rate with different TB shapes ranging from $w \times h = 1 \times 128$ to 128×1. Each subfigure contains results for six viewpoints shown in Fig. 4(a)–(f).

6 Related Work

Krüger *et al.* [KW03] presented the impact of optimization techniques such as early ray termination and empty space skipping [Lev90] on the GPU. Using these techniques, the

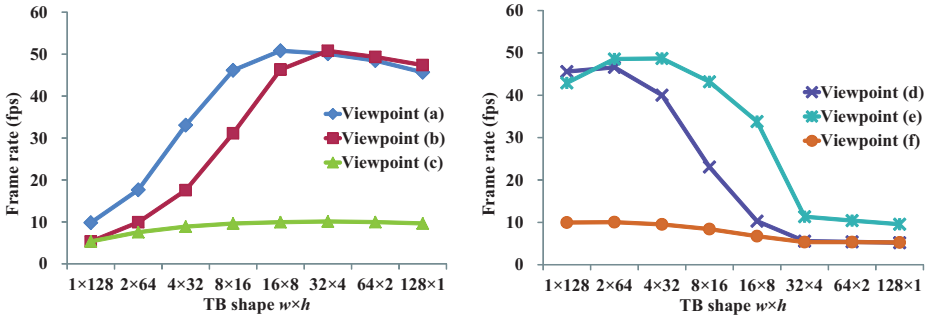


Figure 7: Frame rates with different shapes of TBs. (a) Results for three viewpoints (a)–(c), which are efficient with horizontal warps. (b) Results for the remaining viewpoints (d)–(f), which are efficient with vertical warps.

rendering performance is increased by a factor of 3. These techniques intend to reduce the amount of data access while our optimization strategy reduces the latency of data access. A similar technique is presented by Rijters *et al.* [RV06], who employ an octree data structure on the GPU.

An optimization strategy is presented by Ryoo *et al.* [RRS+08] for CUDA applications. Their strategy investigates the number of resident TBs to evaluate resource utilization. The TB size wh is optimized using this metric but the TB shape $w \times h$ is not investigated for further optimization.

Liu *et al.* [LZS09] presented an optimization framework capable of empirically searching for the best optimizations for GPU applications. Using their framework, we can easily find the best shape of TBs in terms of the performance. In contrast to this empirical approach, our approach gives insight into the relationship between the data locality and the memory access pattern. According to our insight, we can prune the search space in terms of the TB shape, which contributes to reduce the overhead of run-time optimization.

7 Conclusion

In this paper, we presented an acceleration method for texture-based ray casting on the CUDA-enabled GPU. Our method increases the hit rate of the texture cache by selecting the shape of TBs during rendering. This selection focuses on the geometrical relationship between the viewpoint and the volume axes. Our method determines the TB shape such that threads in the same warp can have a small stride of memory access. Such a small stride can be obtained if each warp accesses consecutive voxels along the x-axis. In experiments, we investigated the cache hit rate and the frame rate using six viewpoints. We found that our method increases the cache hit rate by approximately 20%. This higher locality achieves a frame rate of 46.6 fps, which is four times higher than that of a naive method

that uses TBs of a fixed shape. Future work includes further evaluation using other GPU architectures that are compatible with OpenCL.

References

[HS89] William Hibbard and David Santek. Interactivity is the Key. In *Proc. Chapel Hill Workshop Volume Visualization (VVS '89)*, pages 39–43, May 1989.

[KW03] J. Krüger and R. Westermann. Acceleration Techniques for GPU-based Volume Rendering. In *Proc. 14th IEEE Visualization Conf. (VIS'03)*, pages 287–292, October 2003.

[Lev88] Marc Levoy. Display of Surfaces from Volume Data. *IEEE Computer Graphics and Applications*, 8(3):29–37, May 1988.

[Lev90] Marc Levoy. Efficient Ray Tracing of Volume Data. *ACM Trans. Graphics*, 9(3):245–261, July 1990.

[LZS09] Yixun Liu, Eddy Z. Zhang, and Xipeng Shen. A Cross-Input Adaptive Framework for GPU Program Optimizations. In *Proc. 23th IEEE Int'l Parallel and Distributed Processing Symp. (IPDPS'09)*, May 2009. 10 pages (CD-ROM).

[MIH04] Manabu Matsui, Fumihiko Ino, and Kenichi Hagihara. Parallel Volume Rendering with Early Ray Termination for Visualizing Large-Scale Datasets. In *Proc. 2nd Int'l Symp. Parallel and Distributed Processing and Applications (ISPA'04)*, pages 245–256, December 2004.

[MM05] John Montrym and Henry Moreton. The GeForce 6800. *IEEE Micro*, 25(2):41–51, March 2005.

[Mor66] G. M. Morton. A Computer Oriented Geodetic Data Base and a New Technique in File Sequencing. Technical report, IBM Ltd, Ottawa, Ontario, August 1966.

[NIH08] Daisuke Nagayasu, Fumihiko Ino, and Kenichi Hagihara. A Decompression Pipeline for Accelerating Out-of-Core Volume Rendering of Time-Varying Data. *Computers and Graphics*, 32(3):350–362, June 2008.

[NVI10] NVIDIA Corporation. CUDA Programming Guide Version 3.2, September 2010.

[PF05] Matt Pharr and Randima Fernando, editors. *GPU Gems 2: Programming Techniques for High-Performance Graphics and General-Purpose Computation*. Addison-Wesley, Reading, MA, March 2005.

[RRS+08] Shane Ryoo, Christopher I. Rodrigues, Sam S. Stone, John A. Stratton, Sain-Zee Ueng, Sara S. Baghsorkhi, and Wen mei W. Hwu. Program optimization carving for GPU computing. *J. Parallel and Distributed Computing*, 68(10):1389–1401, October 2008.

[RV06] Daniel Rijters and Anna Vilanova. Optimizing GPU Volume Rendering. *J. WSCG*, 14(1/3):9–16, January 2006.

[TIH03] Akira Takeuchi, Fumihiko Ino, and Kenichi Hagihara. An Improved Binary-Swap Compositing for Sort-Last Parallel Rendering on Distributed Memory Multiprocessors. *Parallel Computing*, 29(11/12):1745–1762, November 2003.

Run-Time Dynamic Data Type Transformations[1]

Lazaros Papadopoulos, Alexandros Bartzas and Dimitrios Soudris

School of Electrical and Computer Engineering
National Technical University of Athens, Greece
{lazaros, alexis, dsoudris}@microlab.ntua.gr

Abstract: In the last years, complex applications from various domains are implemented in embedded devices. These applications make extended use of the dynamic memory to store dynamically allocated data structures. The implementation of these data structures affects the performance and the memory usage of the embedded system. A methodology for selecting the appropriate data structures at design time is the Dynamic Data Type Refinement (DDTR) methodology. In this paper we present an extension to this approach, by presenting a methodology for adapting the dynamic data structure implementations to the requirements of the embedded system at runtime. By implementing the proposed methodology to a set of various applications from different domains, we achieve a dynamic memory size reduction up to 32%.

1 Introduction and Motivation

In the emerging market of embedded systems, an increasing amount of applications (e.g., 3D games, video-players) comes from the general-purpose domain and this software needs to be mapped onto extremely compact and mobile devices, which struggles to execute them. These complex applications hold very different restrictions regarding memory usage features, and more concretely are not concerned with an efficient use of the dynamic (heap) memory. Also, they receive input from and serve directly the end user of the embedded system. This means that the actions of the user have significant impact on the control flow of the algorithms in the applications, thus making the execution dynamic and event-driven.

This has led to an increased reliance on specific data structures, which allow data to be dynamically allocated and deallocated at run-time (releasing the memory they occupied back to the Operating System, when it is no longer needed) and provide an easy way for the designer to connect, access and process data. They can cope, in the most efficient way, with the variations of run-time needs (e.g., network traffic, user interaction, controller input) and the massive amounts of data processed and stored. The most common examples of these dynamically allocated data structures are single and double linked lists.

[1] This work is partially supported by the E.C funded FP7-ICT-2009-4-248716 2PARMA Project. Official Website: http://www.2parma.eu.

The data structure implementations affect both the memory consumption and the performance of the application, because each data structure has different characteristics in terms of memory size which it occupies and memory accesses needed to access the data (which affects the performance of the application and the energy consumption of the whole system) [10]. However, the implementation choices made at design time do not take into consideration runtime information that can change during the execution of the application. This information can be derived from the system (e.g. available memory) or from the application (e.g. the current memory size of the data structure).

We argue that the data structure implementations can change at runtime, according the runtime information such as the available memory and the performance requirements of the application. Thus, it is possible to achieve more efficient system resource utilization at runtime. The approach in [10] is the Dynamic Data Type Refinement (DDTR) methodology that provides one optimal data structure implementation for each metric under consideration (i.e. performance and memory footprint). This is accomplished at design time, by inserting the DDTR library interface in the application and then executing the application using some input traces. However, the data structure selected as optimal may not be actually the optimal under certain circumstances.

For example, consider the memory size of two data structure implementations: single linked list (SLL) and dynamic array. The amount of memory that the data structure occupies is obviously affected by the number of objects stored in each data structure and the additional information that the data structure uses to store the objects (i.e. in the case of SLL a pointer to the next object). As a motivation example, Dijkstra application [9] contains a data structure that stores 258 objects of 12 bytes each one, which are accessed by the algorithm. In order to optimize the application in terms of memory footprint, one can use the DDTR approach. According to DDTR the optimal data structure implementation is the SLL. Indeed, Figure 1 displays the comparison between the memory size of SLL and dynamic vector up to 258 objects. However, when the data structure holds between 193 and 256 objects, dynamic vector implementation requires less amount of memory to store the same objects. In this case, we can achieve better memory utilization by changing the data structure implementation from SLL to dynamic array. Thus, in this paper, we examine whether by performing such data structure adaptations, is possible to achieve better memory utilization.

The remainder of the paper is organized as follows. In Section 2, we describe some related work. In Section 3, we analyze the design methodology. In Section 4 our benchmarks are introduced and the experimental results are presented. Finally, in Section 5 we draw our conclusions.

2 Related Work

The authors of [1] present a dynamic data type refinement methodology. Using this methodology the designer can make tradeoffs between performance and energy consumption by selecting different data structure combinations from a library of such implementations.

In [2] a set of metadata is presented that could be used to analyze the behavior of the dynamically allocated data structures. The metadata targeted most the access pattern characterization in sequential and random, as well as some other behaviors like the frequency of constructions and copy-constructions.

The Dynamic Data Type Refinement methodology provides a set of Pareto points, at design time, which the designer can use to make trade-offs between performance, memory footprint and energy. Each Pareto point represents a data structure (or a combination of data structures). These data structure implementations are set at design time and remain the same during the execution of the application. In this paper we present a new approach: We argue that by changing the dynamic data type implementation at runtime, we can achieve better resource utilization. The reason that our approach achieves better results that the DDTR, is the fact that it takes into account runtime information that determine which dynamic data type (or combination) is optimal at each point of the execution of the application.

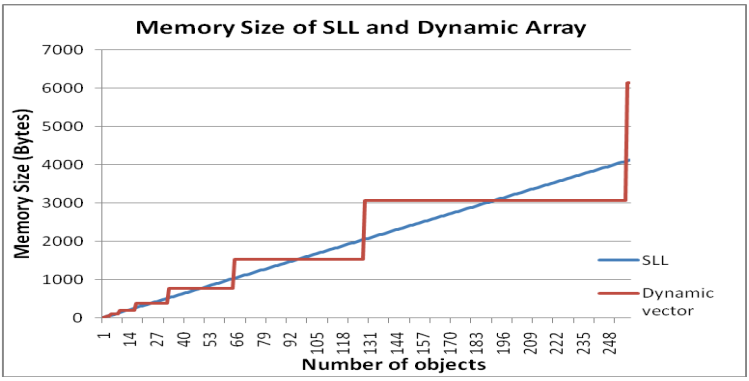


Figure 1: Memory size evolution of single linked list and dynamic array

Several approaches have been proposed for runtime adaptation of applications. For example [3] concentrates in the software adaptation using dynamic change in application components. The authors introduce a framework that monitors changes in the execution environment of applications and performs a dynamic recomposition of the application components, when significant changes in the environment take place.

A few works focus on the dynamic configuration of parallel applications. For instance, in [4] is described a runtime optimization approach that allows the automatic on-the-fly reconfiguration of the parallel simulation code for increasing the performance of the application. The dynamic adaptation is performed by collecting and combining runtime information from the application with static parallel performance models.

There are several tools that focus on runtime software adaptation. For example, Pin [5] is a dynamic binary instrumentation tool that performs in process-level and allows the modification of application instructions prior to the instruction execution. Similar tools, that operate in similar fashion, are Strata [6] and DELI [7].

Runtime adaptation has also been proposed for task migration. For example in [8] is described a mechanism for reducing task migration latency in multi-core architectures, by performing trade-offs between latency and bandwidth. Runtime adaptivity is achieved by using a latency/bandwidth trade-off parameter, which controls the trade-offs. All the aforementioned approaches that refer to the runtime adaptation can be used along with our approach.

3 Methodology Overview

The runtime information that our methodology takes into account is the number of objects stored in each data structure. As mentioned before, the amount of data in the data structure affects both the memory size and, in most cases, the number of accesses needed to access each object. We take advantage of this fact, in order to achieve better memory utilization at runtime.

3. 1 The modes

Since different implementation types for each data structure exists, we define each different combination of data structure implementations of the application as a different *mode* at which the application can run. For each mode the following information is calculated at design time:

- The size of the data structures for each number of objects
- The memory size and the performance overhead of the transition to each other data structure type implementation for each number of objects.

Table 1: Example of a mode

Data Structure	**Implementation**	**Number of objects**	**Memory size**	**Transition to**	**Performance overhead**	**Memory size overhead**
DDT 1	SLL	150	128	DLL	300	200
				Dynamic Array	320	400
DDT 2	DLL	100	140	Dynamic Array	320	450

Using the aforementioned information, it is possible to compare the modes and keep only the optimal ones, by discarding those for which better one already exist in terms of memory size for a specific number of objects. Modes that violate the designer constraints are also discarded. The modes are defined at design time and the runtime manager handles the transition between the available ones at runtime.

Table 1 presents an example of a mode for an application that uses 2 data structures. The first one is considered to be a single linked list that holds 150 objects and the second a double linked list where 100 objects are stored. For the first data structure a possible transition to double linked list and dynamic array is considered and the necessary overhead information is presented. For the second one, only the transition to single linked list is presented, since any other transformation is supposed to have intolerable overhead. Another mode, for example, would be DDT1 implemented as a vector and DDT2 as a SLL for a different number of objects. The conditions to consider a mode available at runtime are the following:

- The memory size of the mode plus the overhead of the transition is less that the memory size of the mode corresponding to the DDTR data structure combination.

- The performance and the memory footprint overhead of the transition to the specific mode from some other mode, do not violate the constraints set by the designer.

To evaluate each mode, we use as a reference mode, the one corresponding to the DDTR methodology. This is done, in order to prove in this paper that using this methodology we can achieve better results in terms of memory utilization than using only the DDTR methodology.

The condition to make a transition at runtime is the following: The memory footprint of the target mode to be less that the memory footprint of the current mode.

3.2 Methodology Description

The methodology is composed of the following three steps and is presented in Figure 2: 1) DDTR exploration; 2) Insertion of the necessary data structure information to the design time manager that automatically detects the available modes; and 3) Execution of the application along with the runtime manager.

The DDTR exploration is exhaustively described at [10]. By implementing the DDTR methodology, the designer obtains a set of optimal data structure implementations, in terms of memory footprint and performance. From this set, the one provided to our tool is the combination that is better in terms of memory footprint. The inputs of the design time manager are the following:

- *Input from the DDTR exploration*. The optimal dynamic data type implementation for each data structure of the application.

- *Object size of each data structure*. This information is necessary to calculate the size of each data structure and obtain the set of available modes.

- *Maximum number of objects for each data structure*. This information is obtained during the DDTR exploration methodology. Although the application is dynamic, and it is not possible to know the exact maximum number of objects in each data structure, the traces used in the DDTR exploration phase, can provide an estimation of the maximum number of objects each data structure holds.

Designer constraints. In the case of real time applications performance constraints may exist. Thus, since the transition from one mode to another causes a delay, these constraints determine whether a mode or a transition to a mode is available or not.

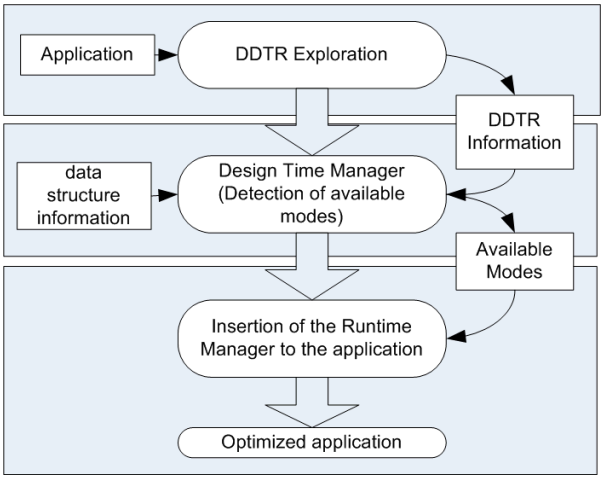


Figure 2: The proposed methodology

The designer inserts the aforementioned information to the design time manager tool. The tool, using the aforementioned inputs, produces a set of candidate modes in which the application can run. Each mode is a different set of dynamic data structure implementations of the application. For all the available modes, which are propagated to the runtime manager, the size of the data structure plus the overhead of the transition to this mode is less that the size of the dynamic data structure selected as optimal by the DDTR methodology. All the dynamic data structure combinations (i.e. modes), for which the aforementioned condition is not valid, are discarded. Thus, from the pool of all the candidate modes, only the optimal ones are provided to the runtime manager.

The input of the runtime manager is the set of the available modes selected by the design time step and the current number of objects in each data structure of the application. The application runs along with the runtime manager. The runtime manager handles the transition between the modes created at the design time phase. Each time an operation takes place in a data structure, the runtime manager checks if transition to another mode is possible.

3.3 The overhead of the methodology

In this subsection we examine the overhead of the presented methodology. The overheads are the following:

- *Design time exploration to obtain the available modes*. This overhead is rather trivial, since the whole process is based on mathematical calculations. The duration depends on the number of data structures and the maximum number of objects that each one holds. For example, for 5 data structures of 10,000 objects each one, the exploration takes less that 1 minute.

- *Increased code size (Memory size overhead)*. The code size of the application increases, since the runtime manager is also compiled along with the application. This overhead less than 1 KB of memory, so can be considered trivial, especially for large applications.

- *Performance overhead (runtime overhead)*: The fact that the runtime manager checks whether a better mode or not exists each time a data structure operation takes place, causes a delay in the execution of the application. However, as shown in the experimental results section, this delay is rather trivial, since every calculation needed is made at design time. Also, the routine that the runtime manager uses to check the available modes is very simple.

- *Overhead of mode changes (runtime overhead)*:

    - Performance overhead: This overhead is affected by the number of objects and the type of source and target data structures. (E.g. It is more time consuming to transfer or remove data from a dynamic vector, than from a single linked list, since the dynamic vector is resized). The performance overhead is known at design time. If the overhead is tolerable, then the corresponding mode is provided to the runtime manager. Otherwise, it is discarded.

    - Memory footprint overhead: During the transfer of data from one data structure to another, there are 2 data structures (the source and the destination) that coexist in the memory. Thus, there is a memory size overhead, which is affected by the type of source and destination data structures and the number of objects to be transferred. However, this overhead is calculated at design time and only if it is tolerable, the corresponding mode is forwarded as an input to the runtime manager.

Our tool precalculates the aforementioned overheads and ensures that a mode is available only when the destination data structure characteristics and the overhead of the transformation are better (in terms of memory footprint or performance) from the current data structure.

3.4 The tool

The tool which implements the described methodology is composed by two parts. The first one is the design time manager which provides the available modes to the runtime manager. The runtime manager handles the transitions between the different modes.

- *Design time manager*: To use the design time manager the designer provides the necessary input to the tool in text mode. The manager is composed by a set of routines which make the necessary calculations to generate all the possible modes. Then, the manager automatically produces the set of the available modes.

- *Runtime manager*: To use the runtime manager, the designer sets the provided interface to the data structures of the application. This is a straightforward process, especially, if the application uses the STL data structures. The runtime manager contains a set of dynamic data structures, along with a routine which decides when to change the current mode, taken as input the current number of objects of each data structure.

Table 2: Cumulative experimental results

App. name	Maximum memory size reduction	Average memory size reduction	Maximum Memory footprint overhead	Average memory footprint overhead	Perf. overhead	Code size increase
Dijkstra	25.1%	8.6%	50.9%	12.16%	22.4%	50%
2D Game	30.3%	16.4%	11.3%	7.1%	3.4%	27%
3D Engine	32%	5.23%	10.3%	6.58%	18.3%	14%
3D Game	25%	3.32%	65.4%	30.38%	22%	3%

4 Experimental Results

To validate our approach, we have chosen a wide range of applications from various application domains. By implementing each step of the methodology, we calculated the memory size gains, as well as the overhead added to each application by the tool. More specifically, we calculate the maximum and the average memory size reduction during the execution of the application, in comparison with the memory size of the data structure implementations suggested by the DDTR. All memory size information provided in the experimental results is the size of all data structures of each application.

As far as the overhead is concerned, we calculated the memory footprint overhead which exists when a mode change takes place at runtime. This overhead exists only during the mode change process and is eliminated after the end of the transformation. Performance overhead is compared with the performance of the original application. Finally, the code size shows the increase in the size of the executable of the application due to the tool. The cumulative results of our case studies are presented in Table 2.

4.1 Dijkstra application

The first test case is Dijkstra algorithm taken from the Mibench Suite [9], which stores network nodes in a data structure. As mentioned earlier, the optimal data structure implementation in terms of memory footprint according to the DDTR approach is the single linked list. Implementing the DDTR solution (i.e. the mode that corresponds to the DDTR solution), no mode changes take place. However, using the adaptive approach, a number of data structure implementation transformations are being made during the execution of the application, which results in a memory size reduction up to 25.1% in comparison with the memory size occupied by the application using single linked list during the whole execution, (which is the solution proposed by the DDTR approach). This is achieved by implementing 63 mode changes during the execution. The memory size overhead due to mode changes (shown in table 2) is not presented in figure 3. It is considered tollerable, thus only the data structure memory footprint evolution is shown. Figure 3 displays the memory size evolution of the application during the whole execution. It can be seen that the memory size using the adaptive approach can be higher that the memory size using the DDTR approach, at some points of the execution of the application. This is because the transformation to the optimal data structure implementation in terms of memory size has very low benefits or is intolerable, according to the constraints set by the designer. Figure 4 shows the memory size reduction achieved by using the adaptive approach. The maximum memory footprint overhead that takes place during the transformation is 50.9%, which can be considered relatively high. However, this overhead can be decreased by the designer, by setting the appropriate constraints, with a corresponding decrease to the memory size reduction.

4.2 Comboling application

A 2D game named Comboling contains a grid of tiles, which are stored in a singly linked list that is filled with tile elements (thus, the memory size is constantly increasing) and accessed in a random pattern [11]. The optimal data structure implementation according to the DDTR approach is the single linked list. Figure 5 displays the memory size comparison between the solution proposed by the DDTR against our approach. Using the adapting approach 8 transformations take place and 30.3% memory size reduction during the execution of the application is achieved. For instance, when the size of the data structure is between 786 and 1024 bytes the vector solution provides less memory size than the single linked list. The memory size gains are presented in Figure 6. The main overhead of our methodology in this application is the code size that seems relatively high. However, the code size of Comboling is less than 1 KB, so the overhead added by our tool can be considered trivial.

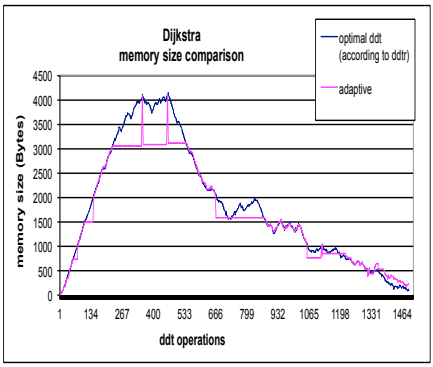


Figure 3: Dijkstra application – run-time evolution of memory footprint

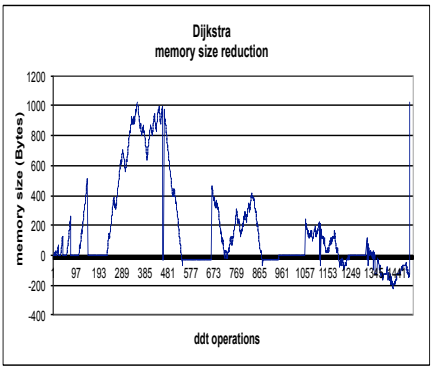


Figure 4: Dijkstra application –reduction of memory footprint

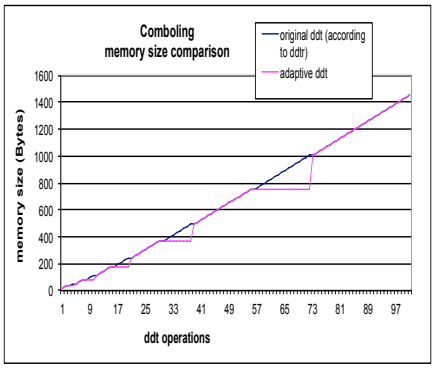


Figure 5: Comboling application – run-time evolution of memory footprint

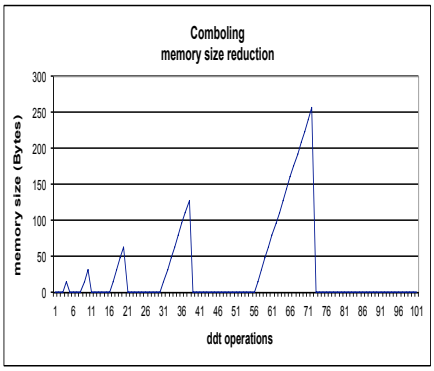


Figure 6: Comboling application – reduction of memory footprint

4.3 Simblob application

Simblob is a 3D environment creation engine that utilizes vectors to hold its dynamic data [12]. The optimal data structure in terms of memory footprint according to the DDTR approach is the single linked list. Figure 7 shows the comparison between the DDTR and our approach. Implementing the adaptive approach,4 transformations take place and the maximum memory size reduction is 32%. Figure 8 displays the memory size gains. It can be seen by Table 2 that the overheads of our methodology in this test case are relatively low.

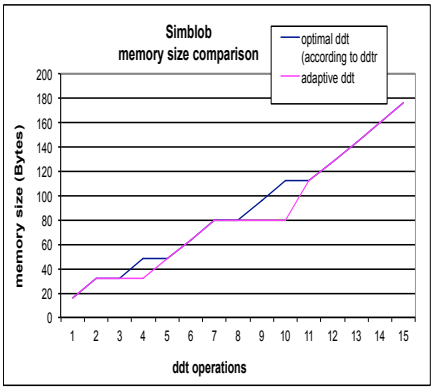


Figure 7: Simblob application – run-time evolution of memory footprint

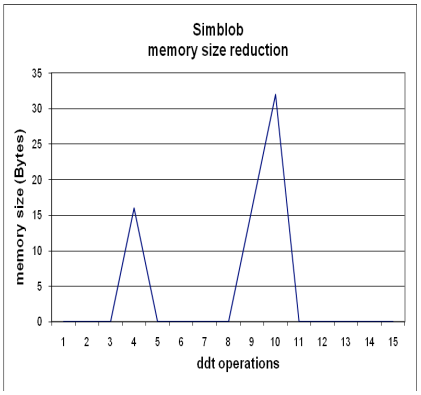


Figure 8: Simblob application – reduction of memory footprint

4.4 VDrift application

Vdrift is a 3D open source racing game with realistic physics [13]. The application uses vectors to store its dynamic data for graphics, physics and collisions. The DDTR approach suggests single linked lists as the optimal data structure implementations in terms of memory footprint. Figure 9 displays the memory size comparison between the DDTR approach and the adaptive one. During the adaptive approach 23 mode changes take place and the maximum memory size reduction achieved is 25%. The memory gains are presented in Figure 10.

The memory footprint overhead of the transformations is relatively high, as can be seen in Table 2. However, as mentioned with the Dijkstra test case as well, the overhead can be decreased by the designer, by setting the appropriate constraints. It should be taken into account that in this case, the amount of memory reduction will be also decreased, since some mode transformations will not take place.

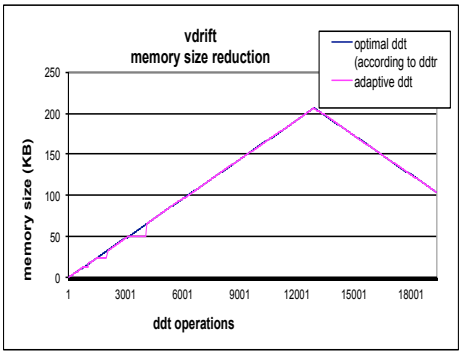


Figure 9: VDrift application – run-time evolution of memory footprint

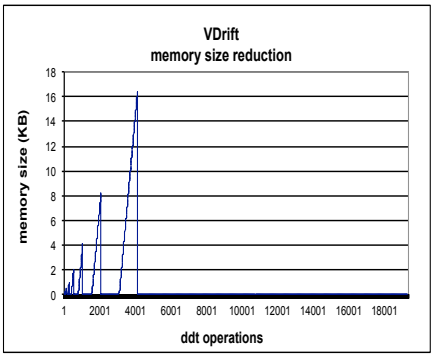


Figure 10: VDrift application – reduction of memory footprint

5 Conclusions

In this paper we presented a methodology for adapting the dynamic data structure implementations of an application to the runtime environment at which the embedded system executes the application. We proved that it is possible to achieve better dynamic memory utilization by using a runtime manager that adapts the data structure implementations by adding a tolerable overhead. Our future work addresses the extension of the data structure implementation library of the runtime manager, as well as the further reduction of the overheads of the proposed methodology.

References

[1] Alexandros Bartzas, Stylianos Mamagkakis, Georgios Pouiklis, David Atienza, Francky Catthoor, Dimitrios Soudris, and Antonios Thanailakis, "Dynamic data type refinement methodology for systematic performance-energy design exploration of network applications," In Proc. DATE, 740-745, EDAA, 2006

[2] Alexandros Bartzas, Miguel Peon-Quiros, Christophe Poucet, Christos Baloukas, Stylianos Mamagkakis, Francky Catthoor, Dimitrios Soudris, and Jose M. Mendias, "Software metadata: Systematic characterization of the memory behaviour of dynamic applications," in J. Syst. Softw. 83, 6, June 2010, Elsevier

[3] Arun Mukhija and Martin Glinz, "Runtime Adaptation of Applications Through Dynamic Recomposition of Components," In Proc. of ARCS, 124-138, Springer, 2005

[4] Xiaoji Ye and Peng Li, "On-the-fly runtime adaptation for efficient execution of parallel multi-algorithm circuit simulation," in Proc. of ICCAD, 298-304, IEEE, 2010

[5] Kim Hazelwood and Artur Klauser, "A dynamic binary instrumentation engine for the ARM architecture," in Proc. of CASES, ACM, 261-270, 2010

[6] K. Scott, N. Kumar, S. Velusamy, B. Childers, J. W. Davidson, and M. L. Soffa, "Retargetable and reconfigurable software dynamic translation," in Proc. of GCO, IEEE Computer Society, 36-47, 2003

[7] Giuseppe Desoli, Nikolay Mateev, Evelyn Duesterwald, Paolo Faraboschi, and Joseph A. Fisher, "DELI: a new run-time control point," in Proc. MICRO 35, IEEE Computer Society Press, 257-268, 2002

[8] Jahn, J.; Faruque, M.A.A.; Henkel, J., "CARAT: Context-aware runtime adaptive task migration for multi core architectures," in Proc. of DATE, 2011

[9] M. Guthaus, J. Ringenberg, D. Ernst, T. Austin, T. Mudge, and R. Brown, "MiBench: a free, commercially representative embedded benchmark suite", in Proc. of 4th IEEE Workshop Workload Characterization, 10-22, 2001

[10] L. Papadopoulos, C. Baloukas, N. Zompakis, D. Soudris, "Systematic Data Structure Exploration of Multimedia and Network Applications realized Embedded Systems", in Proc. of IC-SAMOS, 58-65, 2007

[11] http://www.Planet-Source-Code.com/vb/scripts/ShowCode.asp?txtCodeId=3222&lngWId=10Sim

[12] http://sourceforge.net/projects/simblob

[13] http://vdrift.net/

Evaluating Run-time Resource Management Policies for Multi-core Embedded Platforms with the EMME Evaluation Framework

Giovanni Mariani[1], Gianluca Palermo[2], Vittorio Zaccaria[2], Cristina Silvano[2]

[1] ALaRI – University of Lugano, Switzerland
[2] Politecnico di Milano, Italy
marianig@alari.ch, {gpalermo, zaccaria, silvano}@polimi.it

Abstract: Toady's embedded computing electronic products are based on multi-core platforms and they are capable to concurrently execute different applications. For these products it is of paramount importance that a Run-time Resource Management (RRM) system integrated in the Operating System (OS) arbiters about resource allocation to the active applications. The RRM should take decisions at run-time to maximize platform performance and minimizing non-functional costs such as power consumption or memory requirements. However, embedded system design covers a broad range of applications and the customer requirements are very different depending on the target device. In general there is not an unique RRM that best fits in all possible embedded scenarios.

This paper presents the **EMME Evaluation Framework**, an open source tool that provides a methodology and the accompanying infrastructure to quickly explore the effects of different RRM systems for a target use case scenario. The tool aims at the analysis of different figures of merit of the system being designed such as the applications' *response time*, system *throughput* and *power consumption*.

Different RRM modules are released with the framework. These modules implement different RRM policies that define how to allocate computing resources to the active applications while fitting in a power budget that is assumed assigned by other layers of the OS.

1 Introduction

The availability of many processors (or *computing elements*) on a single chip brought new challenges to system designers. In particular, efficient power management has become a primary factor for product success. This is evident for portable devices but has subtle yet important consequences on reliability and cooling costs for non-portable systems.

Traditional techniques for power management consider switching off or slowing down the frequency of computational elements which are underutilized [BBM00, IBC+06]. If the switching overhead is negligible and the performance is not saturated, it is possible to meet performance requirements with fewer active resources and lower power consumption.

When considering multiprogrammed multi-core scenarios where multiple applications are competing to access shared resources, traditional DVFS techniques [IBC+06, BHB+08] are not enough. The proposed *Efficient run-time resource Management for Multi-core Embedded systems* (EMME) framework shall also decide how to allocate available com-

puting resources to the active applications. In these scenarios, solving the RRM problem is a challenging task for the following reasons:

- **Dynamism of the system:** operating configurations providing high performance are typically power hungry hence they should be avoided unless strictly necessary. On the other hand, user behaviors are highly dynamic, unpredictable and unknown at design time. It is not possible to statically identify a setting of the operating parameters which maximizes system performance while fitting in a given power budget. To solve the RRM problem the operating parameters should be tuned at run-time once the system state and user requirements are known.
- **Complex system behaviors:** in a multiprogrammed multi-core scenario, the relationships between operating parameters and overall platform performance can be very complex when considering the resource distribution as a run-time tunable parameter. As shown in [SRS+95, MPSZ11], optimal processor assignment is very workload dependent (and is less than intuitive).
- **Complexity of the decision problem:** the problem of deciding which operating configuration to set for each active application (and thus for the system as a whole) belongs to the NP-hard class of problems [YCNCC06]. Hence, the problem is too complex to be solved exactly and only an approximate solution can be found. It is of crucial importance that heuristic algorithms implementing the run-time decision making should be fast in order to minimize run-time overhead.

The RRM system can significantly impact on the system performance. In order to fully exploit the potentialities of multi-core architectures, selecting the right RRM technique to apply for the target platform is of outstanding importance. Unfortunately, there is not an unique RRM system that best fits in all possible use case requirements in general.

In this paper we present the *EMME Evaluation Framework* (in short, **EMMEframework**) [Mar11]. The goal of this framework aims at providing to embedded system designers a tool for system level performance analysis of homogeneous multi-core systems. Given a run-time scenario and a RRM policy, the framework provides a fast empirical analysis of the system behavior. This enables the designer to explore the effects of different RRM policies and to validate the desired system properties at the earliest design phases. The approach is targeted to *soft real-time* applications where the RRM is responsible for *maximizing the system performance* while fitting in a *power budget constraint*.

The paper is organized as follows. Section 2 presents related work in the field of RRM for multi-core embedded systems. Section 3 describes in details the **EMMEframework**. Then, Section 4 reports some experimental results obtained from the analysis of different RRM techniques for an example multiprogrammed multi-core embedded scenario. Finally, in Section 5 some concluding remarks are outlined.

2 Background

The RRM scenarios considered in the present work are *multiprogrammed multi-core embedded systems* where different applications, each one composed of different parallel

threads, are competing to access the system resources. In these scenarios the relationships between run-time tunable parameters and overall platform behavior might be very complex and sometime counter-intuitive [SRS+95].

In general, analytical models of system performance are not available when RRM takes into account parameters such as resource distribution. For instance, performance values of an application when parallelized on a certain number of processors are strictly application-dependent. In fact, for some applications performance values might scale linearly with the number of computing resources allocated to their execution while other applications cannot exploit a huge number of processors.

Authors in [AW06] noted that for multiprocessor workloads, the *Instruction Per Cycle* (IPC) is a poor performance index and identified cases where IPC increases do not reflect any performance gain. For multiprogrammed multi-core scenario, *system-level performance metrics* as *throughput* measured in terms of *job/s* or other *user-oriented performance metrics* as the applications' *response time* are more suitable [EE08].

The RRM approach we consider assumes that the target application set is known at design time and application performance for different resource allocations can be measured by exploiting off-the-shelf simulation models [YCAM+11, YCNCC11, SGB+09]. Given the design-time analysis of application performance, the RRM should decide how to allocate the computing resources available on the embedded platform.

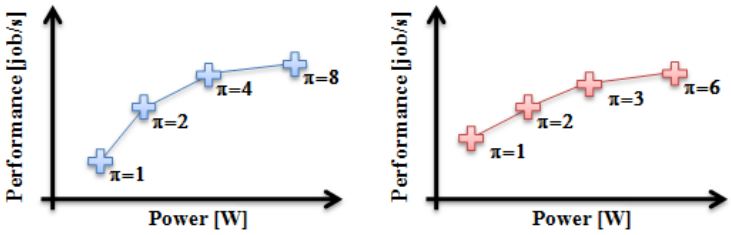
In order to maximize system performance and improving battery lifetime in portable devices, it is of outstanding importance to select the right RRM policy to deploy on the system being designed. For these reason we propose an open source framework for quickly analyzing the effects of different RRM systems targeting the optimization of an user-oriented performance index such as the applications' *response time*.

To avoid lengthy simulations, the performance evaluation considered in **EMMEframework** is carried out at a very high level. First, the applications are independently characterized in terms of their execution time. Then, to evaluate the performance of a specific RRM policy for a given run-time scenario, the applications' execution times are retrieved from the previous characterization. Thus, different RRM policies can be evaluated without the need to re-run a detailed simulation of each application. An example of power management system that considers functional and temporal independence between different applications can be found in [NMG11]. Few work has been done on tools for modeling and analyzing power management techniques for multi-core [BHB+08, TDM11]. However state of the art is manly dedicated to the modeling of DVFS and other orthogonal power management approaches rather than on the resource allocation problem.

3 The EMME Evaluation Framework

In the **EMMEframework**, the user selected RRM system takes decisions on the basis of the following information:

- The **application characterization** performed at design-time. The application characterization reports for each application the set of *operating configurations*, i.e. performance and power indices obtained when the application is executed using a cer-



(a) app0's operating configurations (b) app1's operating configurations

Figure 1: Design-time application characterization reporting the operating configurations of two applications in terms of: power consumption (x axis), performance (y axis) and resource requirement (the π value).

tain number of cores (Figure 1). In our approach we assume that this characterization can be obtained simulating each application independently.

- The **user activity**. We consider that the user activity (or the interaction with the external environment) issues the processing of some data by the active applications. From now on we will use the term *job* to refer to an unitary data chunk to be processed, e.g. a single frame in a video application. In general the user activity is unknown at design-time but we assume that it can be profiled during run-time. For example the execution of a video application might require the processing of a certain number of frames per second (e.g. 25 frames per second). This computing request can be profiled at run-time and the RRM can use the profiling data during its decision making.

- The **power budget**, which is assumed to be set by the OS. This can be set according to, among other things, the actual system state (e.g., the system is plugged into a power supply or not).

3.1 The run-time methodology

The **EMMEframework** targets homogeneous multi-core computing platform. We consider that at run-time different applications are executed and compete to access the available processing elements. The RRM introduces a run-time processor assignment policy to maximize the user-perceived performance (in terms of applications' response time) while fitting in the **power budget**. We define as application response time the average time an application job spend in the system from its arrival to its completion. The processor assignment depends on the **user activity** that dynamically issues the processing of different applications' jobs (e.g. the elaboration of different frames for a video application) and on the design-time **application characterization**.

We consider code versioning [YCNM+06] as the main enabling technology in order to change the task-level parallelization of an application. However, other mechanisms for

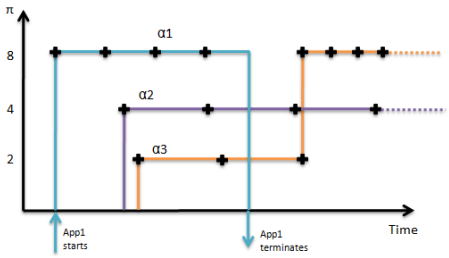


Figure 2: Run-time behavior. The parallelization π changes only between the execution of two jobs.

manipulating the program representation to exploit available processors can be considered with their additional overhead (e.g. stream program fusion [GTA06, GTK+02]).

We consider that application parallelization cannot be changed during the execution of a single job but only between the execution of two different jobs. We also consider that some jobs might be temporarily stored in memory while waiting to be processed. This might happen for bursty applications where many jobs are issued during a short time interval. Figure 2 shows an example behavior of the task-level parallelization chosen by a RRM for a scenario with 3 applications running concurrently (job starting times are indicated with '+'). When α_1 starts, 8 cores are allocated to it. Then applications α_2 and α_3 enter the system. The RRM allocates to these applications 4 and 2 cores respectively. When α_1 exits the system, the parallelization of α_3 is increased to 8 cores. The run-time decisions are taken following the RRM policy and influence the overall system performance.

3.2 Evaluation of system performance

Following an application specific design approach we consider that the target application set is known at design time. However, we consider that the mix of applications to be concurrently executed is not known a priori and, at any moment, the user is free to launch the execution of any application selected between the ones known at design time. We consider that at run-time the user activity issues the elaboration of a sequence of applications' jobs. We assume that at run-time the user activity can be profiled in terms of number of jobs issued for each application in a time unit (the applications' *arrival rates*).

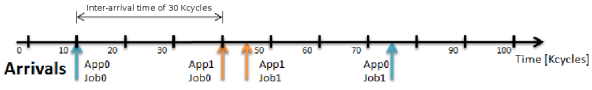
Using a traditional approach to evaluate the run-time performance of the target multiprogrammed multi-core system one would have to simulate the concurrent execution of the whole application mix with a detailed architectural model. To give a practical idea of the computational cost of simulating a realistic scenario, the simulation of a single MPEG2 application processing 2 frames using the SESC simulator [RFT+05] might take few minutes. Extending this simulation to a significant number of frames and considering the concurrent execution of other applications might lead to a simulation time of several hours or even few days.

To reduce this computational cost and to produce results for complex scenarios in few seconds, the **EMMEframework** takes some assumptions on the underlying computing

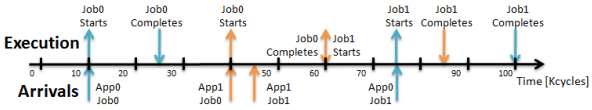
platform and on the predictability of the jobs' execution times. We assume that:

- At run-time, the set of computing resources is partitioned into disjoint subsets and each subset is allocated to a different application.
- The execution time of a specific job depends only on the input dataset and on the resources allocated to its elaboration. Thus, there are no interferences between different applications during the concurrent execution. It is worth to notice that this assumption might require specific communication infrastructure in order to keep a predictable communication time. In future works we envision to extend the framework to consider a less conservative assumption on the job execution time.
- For a given application, before starting the execution of a new job, all previous jobs should be completed. If a new job arrives while another job is under elaboration, the new job is temporary stored in the on-chip memory and it waits to be scheduled.
- The switching time required to change the operating configuration for a given application is negligible in reference to the execution time of a single job.

Under these assumptions, the evaluation of system performance can be obtained at a very high level. Given an **input trace**, defined in terms of jobs' arrival times, the **EMME-framework** simulates the computing system by scheduling the input jobs considering the resource distribution defined by the user selected RRM policy. The framework output consists of an **execution trace** that completes job arrivals information with data about job waiting and execution time (we consider the job response time as the overall time a job spend in the system, i.e. waiting plus execution times).



(a) Example of **input trace**.



(b) Example of output **execution trace**.

Figure 3: **Input trace** and output **execution trace** for an example two application scenario. Events related to different applications are highlighted with different colors.

An example is shown in Figure 3. As the first job of app0 arrives (Figure 3(a)), its execution starts and the elaboration is completed at $25Kcycles$ (Figure 3(b)). Then a job of app1 arrives and its execution starts (at $40Kcycles$). A second job of application 1 arrives at $45Kcycles$ but its execution will start only at $60Kcycles$. The second job of app1 is subject to a waiting time since we assumed that, before starting the execution of a new job, the application must terminate the elaboration of all previous jobs. A second job of app0 arrives while app1 is elaborating. In this example we consider that the RRM did not allocate all computing resources to the execution of app1 and thus the system can schedule the concurrent execution of the new job of app0.

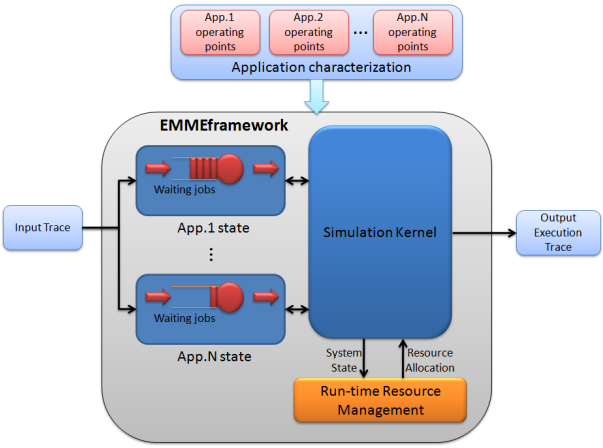


Figure 4: The **EMMEframework** structure.

To generate the output **execution trace** from the **input trace**, the **EMMEframework** does not simulate on the detailed architectural model the job execution. In fact, it uses the assumption on the job execution time predictability to compute, given the starting time of the job and the resources allocated to its execution, the completion time. Optionally, in the **input trace**, together with the job arrival times, one can specify information on the variability of each job execution time in reference to the average one. This variability is considered due to the specific input dataset.

In the **EMMEframework** (Figure 4), the application states are kept updated during the system simulation. As new jobs are arriving these are dispatched to the applications' waiting queue and the **simulation kernel** cares of iteratively identifying the event to be processed next (e.g. either a job arrival or completion or an RRM invocation). During the **EMMEframework** execution, the **simulation kernel** keeps track of the system state in terms of number of completed jobs, number of jobs currently in the system (either waiting or executing), and power consumption resulting from the current resource allocation.

When executing the **EMMEframework**, the user can select a **RRM policy** among the ones available (Section 3.3). During the simulation, the selected RRM will interact with the simulation kernel to access the current system state and to assign the resource allocation (Figure 4).

Together with the output jobs' starting and completion times, detailed information about resource distribution and power consumption are reported in the output **execution trace**.

3.3 The run-time resource management policies

A RRM policy defines how the resources should be distributed between the active applications. Three RRM policies are released with the **EMMEframework**, named *Pull High Push Low* (**`PHPL`**), *maximization of the current Throughput* (**`maxT`**) and *Application-*

specific Run-Time managEment (***ARTE***). The RRM policies can be linked to the **EMME-framework** at run-time. This linking mode enables to quickly evaluate different RRM strategies without the need of recompiling the framework for each strategy.

3.3.1 Pull High Push Low

The ***PHPL*** policy is derived from the approach presented in [IBC+06]. This policy periodically verifies the power consumption of the different applications and modifies the resource allocation to fit in the power budget (first) and to balance the power consumed by the different applications (second).

Every time the ***PHPL*** is invoked, the RRM verifies if the power budget has been exceeded. If this is the case, ***PHPL*** reduces the parallelism of the application consuming the most power. Otherwise, the power budget not yet in use is allocated to the application consuming the least power by increasing its parallelization.

3.3.2 Maximization of the current Throughput

The ***maxT*** policy is presented in [MPSZ09]. ***maxT*** is invoked every time an application switches between the idle and the ready states. ***maxT*** exhaustively explores the possible allocations of computing resources to the set of applications currently running. The resource allocation providing the maximum throughput sum (measured in Job/s) is selected.

3.3.3 Application-specific Run-Time managEment

The ***ARTE*** policy is presented in [MPSZ11]. ***ARTE*** takes some additional assumptions on the computing system to model it using queuing theory. In particular, it is assumed that job inter-arrival times are exponentially distributed with a certain mean (derived from the profiled data). Moreover, it is assumed that the job execution time for different datasets is approximately constant. Given these assumptions, each application is modeled as a M/D/1 (Markov arrival, deterministic execution, single server) queuing system and the job response time is given analytically [Tri02].

ARTE is invoked periodically. Within a RRM period resources are reserved to the applications. Resource reservation allows to manage constraints on the individual application throughput on a window-based, periodic basis. In ***ARTE***, the run-time exploration targets the minimization of the expected average job response time which metric is derived analytically at run-time on the basis of queuing theory.

4 A practical example

As example case study we analyze an embedded Chip Multi Processor (CMP) composed of an **host processor** and a **computing fabric** consisting of 16 computing elements (Figure 5). The OS, including the RRM system, runs on the host processor. The computing fabric

is used to process the input workload. The **host processor** takes decision about resource assignment and, in this sense, it acts as **fabric controller**.

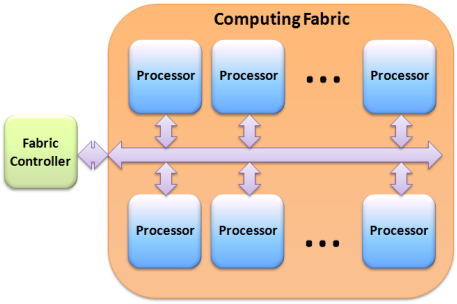
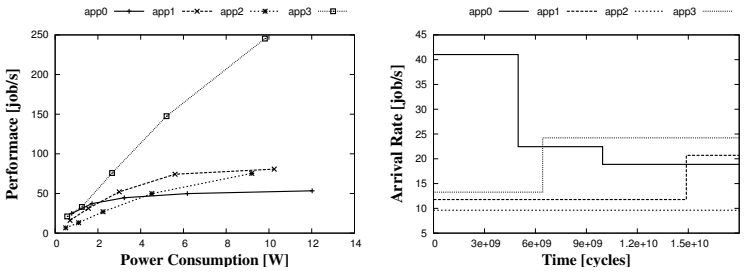


Figure 5: Overview of the target Chip Multi Processor.

We considered MIPS-like processors with a shared memory architecture. The inter-processor communication is based on a high-bandwidth split transaction bus supporting a write-invalidate snoop-based MESI coherence protocol acting directly between L2 caches.

In the considered case study the application mix populating the computing system is taken from the SPLASH-2 parallel benchmark suite [WOT+95]. We focus our attention on the following application kernels: Complex 1D FFT (*fft*) Integer Radix Sort (*radix*), Ocean Simulation (*ocean*) and Blocked LU Decomposition (*lu*). In the following text we will call these applications: {app0 ... app3}.

To generate the input **application characterization** (Figure 6(a)), we model the computing fabric using the SESC simulation tool [RFT+05], a fast MIPS instruction set simulator for CMPs. To generate the **input trace** we considered exponentially distributed jobs' inter-arrival times (Poisson process). We also considered that the unpredictable user behavior generates variations in the average job arrival rates (Figure 6(b)) and that some variability in the job execution time is introduced due to the input datasets. In particular, we considered a job execution time uniformly distributed in the range of 10% around the average.



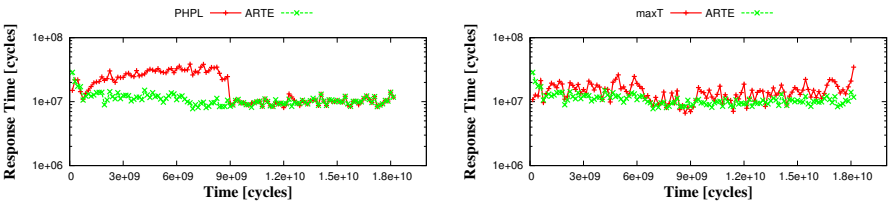
(a) Application characterization. (b) Average job arrival rate times.

Figure 6: Application characterization and average job arrival rates for the specific case study.

The operating frequency of the computing elements is $300MHz$ and the input trace is of 60 seconds length (i.e. $18Gcycles$). For our analysis we consider a power budget of $8.6W$,

that is 70% of the power consumed when all applications are concurrently executing on the 16 cores composing the computing fabric.

Evaluating the system performance. To evaluate the efficiency of the different RRM policies we simulate the system using the **EMMEframework**. A system simulation runs in less than 1 second on our host machine (Intel CoreTM Duo T6570, 2.1Ghtz). The output execution traces are then post-processed using the routines released with the same framework. These routines return (among other data), a graphical comparison of the geometric average of applications' response time (Figure 7).



(a) Comparison between ***ARTE*** and ***PHPL***

(b) Comparison between ***ARTE*** and ***maxT***

Figure 7: Response time comparisons between the considered RRM policies.

For the specific case study, ***ARTE*** provides better performance. To select a RRM policy for final deployment on the target system one might be interested in verifying that job response time for a certain application never exceeds a certain value or verifying that the storage requirements do not exceed the on-chip memory.

The **EMMEframework** produced detailed data on the system behavior that can be inspected. For example the post-processing routines return also the plot of the number of jobs resident in the system that is directly correlated with the on chip memory requirements (this happens because waiting jobs must be temporary stored in memory).

When using ***PHPL*** for our case study the number of jobs resident in the system reaches 160 instances (Figure 8). This might generate implementation problems in our design. The described phenomena happens since, during the initial $5Gcycles$, the arrival rate of app0 is very high. To adequately serve this computing request the most of computing resources should be allocated to app0. ***PHPL*** equally distributes the computing resources between the active applications, thus the resources allocated to app0 are not enough to serve such a high arrival rate. Consequently during the initial period some arriving jobs of app0 should be buffered increasing the storage requirements.

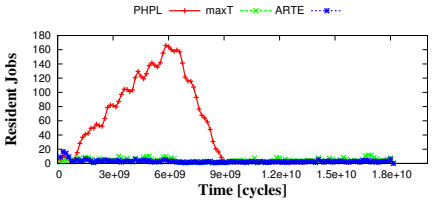


Figure 8: Number of jobs resident in the system when considering different RRM policies

5 Summary

In this paper we presented the **EMMEframework** that enables to quickly explore the effects of different RRM policies for a target multiprogrammed multi-core scenario.

The framework takes as input the **application characterization** obtained from the simulation of the target applications on a detailed architectural model. However, the concurrent simulation of the different applications on the detailed architectural model is avoided. This reduces significantly the computational costs related with the analysis of different RRM policies for the target use case scenario.

Together with the **application characterization** the framework takes an **input trace** defining jobs' arrival times. The **execution trace** output of the **EMMEframework** completes the **input trace** with data related to jobs' starting and completion time. Post-processing routines released with the **EMMEframework** enables to compare the different RRM policies at a very high level of abstraction.

References

[AW06] A.R. Alameldeen and D.A. Wood. IPC Considered Harmful for Multiprocessor Workloads. *Micro, IEEE*, 26(4):8 –17, jul. 2006.

[BBM00] L. Benini, R. Bogliolo, and G. De Micheli. A survey of design techniques for system-level dynamic power management. *IEEE Transactions on VLSI Systems*, 8:299–316, 2000.

[BHB+08] R. Bergamaschi, Guoling Han, A. Buyuktosunoglu, H. Patel, I. Nair, G. Dittmann, G. Janssen, N. Dhanwada, Zhigang Hu, P. Bose, and J. Darringer. Exploring power management in multi-core systems. In *Proc. Asia and South Pacific Design Automation Conference ASPDAC 2008*, pages 708–713, 2008.

[EE08] S. Eyerman and L. Eeckhout. System-Level Performance Metrics for Multiprogram Workloads. *Micro, IEEE*, 28(3):42 –53, may. 2008.

[GTA06] Michael I. Gordon, William Thies, and Saman Amarasinghe. Exploiting coarse-grained task, data, and pipeline parallelism in stream programs. In *Proceedings of the 12th international conference on Architectural support for programming languages and operating systems*, ASPLOS-XII, pages 151–162, New York, NY, USA, 2006. ACM.

[GTK+02] Michael I. Gordon, William Thies, Michal Karczmarek, Jasper Lin, Ali S. Meli, Andrew A. Lamb, Chris Leger, Jeremy Wong, Henry Hoffmann, David Maze, and Saman Amarasinghe. A stream compiler for communication-exposed architectures. In *Proceedings of the 10th international conference on Architectural support for programming languages and operating systems*, ASPLOS-X, pages 291–303, New York, NY, USA, 2002. ACM.

[IBC+06] C. Isci, A. Buyuktosunoglu, C. Cher, P. Bose, and M. Martonosi. An Analysis of Efficient Multi-Core Global Power Management Policies: Maximizing Performance for a Given Power Budget. In *Proceedings of the 39th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 347–358, 2006.

[Mar11] Giovanni Mariani. EMME: Efficient run-time resource Management for Multi-core Embedded platforms, 2011. http://www.alari.ch/emme.

[MPSZ09] G. Mariani, G. Palermo, C. Silvano, and V. Zaccaria. A design space exploration methodology supporting run-time resource management for multi-processor Systems-on-chip. In *Proc. IEEE 7th Symp. Application Specific Processors SASP '09*, pages 21–28, 2009.

[MPSZ11] G. Mariani, G. Palermo, C. Silvano, and V. Zaccaria. ARTE: An Application-specific Run-Time management framework for multi-core systems. In *Application Specific Processors (SASP), 2011 IEEE 9th Symposium on*, pages 86 –93, june 2011.

[NMG11] A. Nelson, A. Molnos, and K. Goossens. Composable power management with energy and power budgets per application. In *Embedded Computer Systems (SAMOS), 2011 International Conference on*, pages 396 –403, july 2011.

[RFT+05] Jose Renau, Basilio Fraguela, James Tuck, Wei Liu, Milos Prvulovic, Luis Ceze, Smruti Sarangi, Paul Sack, Karin Strauss, and Pablo Montesinos. SESC simulator, January 2005. http://sesc.sourceforge.net.

[SGB+09] H. Shojaei, A. Ghamarian, T. Basten, M. Geilen, S. Stuijk, and R. Hoes. A Parameterized Compositional Multi-dimensional Multiple-choice Knapsack Heuristic for CMP Run-time Management. In *DAC '09: Proceedings of the 46th conference on Design automation*, New York, NY, USA, 2009. ACM.

[SRS+95] E. Smirni, E. Rosti, G. Serazzi, L. W. Dowdy, and K. C. Sevcik. Performance Gains From Leaving Idle Processors in Multiprocessor Systems. In *in International Conference on Parallel Processing*, pages 203–210, 1995.

[TDM11] Ibrahim Takouna, Wesam Dawoud, and Christoph Meinel. Accurate Mutlicore Processor Power Models for Power-Aware Resource Management. In *Dependable, Autonomic and Secure Computing (DASC), 2011 IEEE Ninth International Conference on*, pages 419 –426, dec. 2011.

[Tri02] Kishor S. Trivedi. *Probability and statistics with reliability, queuing and computer science applications*. John Wiley and Sons Ltd., Chichester, UK, 2002.

[WOT+95] S. C. Woo, M. Ohara, E. Torrie, J. P. Singh, and A. Gupta. The SPLASH-2 programs: characterization and methodological considerations. In *Proc. 22nd Annual Int Computer Architecture Symp*, pages 24–36, 1995.

[YCAM+11] C. Ykman-Couvreur, P. Avasare, G. Mariani, G. Palermo, C. Silvano, and V. Zaccaria. Linking run-time resource management of embedded multi-core platforms with automated design-time exploration. *IET Computers & Digital Techniques*, 5(2):123–135, 2011.

[YCNCC06] Ch. Ykman-Couvreur, V. Nollet, Fr. Catthoor, and H. Corporaal. Fast Multi-Dimension Multi-Choice Knapsack Heuristic for MP-SoC Run-Time Management. In *Proc. International Symposium on System-on-Chip*, pages 1–4, 2006.

[YCNCC11] Ch. Ykman-Couvreur, V. Nollet, F. Catthoor, and H. Corporaal. Fast multidimension multichoice knapsack heuristic for MP-SoC runtime management. *ACM Trans. Embed. Comput. Syst.*, 10:35:1–35:16, May 2011.

[YCNM+06] Ch. Ykman-Couvreur, V. Nollet, Th. Marescaux, E. Brockmeyer, Fr. Catthoor, and H. Corporaal. Pareto-Based Application Specification for MP-SoC Customized Run-Time Management. In *Proc. International Conference on Embedded Computer Systems: Architectures, Modeling and Simulation IC-SAMOS 2006*, pages 78–84, 2006.

Processor Coupling Architecture for Aggressive Voltage Scaling on Multicores

Koji Kugata, Shinpei Soda, Yohei Nakata, Shunsuke Okumura, Shintaro Izumi, Masahiko Yoshimoto, and Hiroshi Kawaguchi

Graduate School of System Informatics
Kobe University
Kobe
Japan
657-8501
kugata@cs28.cs.kobe-u.ac.jp

Abstract: We propose novel multicore architecture in which dual processors (positive-true processor element and negativetrue processor element: pPE and nPE) can be combined and serve as a low-voltage/high-performance single PE. The coupling processor occupies a double area, but it operates at a lower voltage or faster than the original processor. The proposed scheme is suitable to voltage scaling on multicores that have abundant PEs. To evaluate and demonstrate the proposed architecture, we designed octa-core coupling DSP in a 65-nm CMOS technology, on which we confirmed a 1-MHz operation at a single supply voltage of 0.5 V.

1 INTRODUCTION

In multicore architecture, there are abundant processor elements (PEs). Several methods using the multicores have been proposed. For example, IBM's Cell processor increases its yield from 20% to 40% by disabling one of eight synergistic processor elements [1-2]. In another instance, a speed boosting technique, which called "Turbo Boost", is exploited in an Intel's Core i7 processor [3] and others. To dynamically vary the operating frequency, the Turbo Boost technology controls the number of active cores because the core frequency is determined by core temperature.

In this paper, we propose a power reduction technique using the abundant PEs. For a low-load task, not all PEs are needed; low power is desired rather than low processor usage. Under that condition, we can exploit an extra hardware resource to save the power. Aggressively reducing a minimum operating voltage (Vmin) with two PEs is allowed, which is effective for low power.

Fig. 1 shows the proposed multicores; there are a set of dual processors in which a positive-true PE (pPE) and negative-true PE (nPE) are adjacent. A normal task is carried out either pPE or nPE at a nominal voltage. For a low-load task, it is assigned to a

coupling processor (a pair of pPE and nPE) that can operate at a low voltage (voltage scaling). We call this "processor coupling architecture", which can also allocate a high-load task to a coupling processor running at a high frequency (speed boosting).

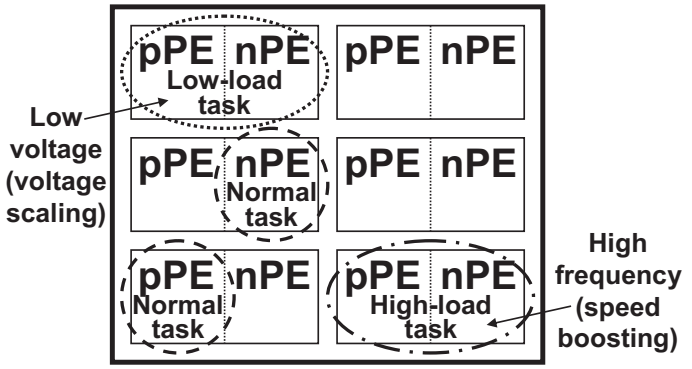


Figure 1: Proposed multicores: pairs of pPE and nPE.

2 PROCESSOR COUPING ARCHITECTURE

Fig. 2 portrays the two types (voltage scaling and speed boosting) of formations in the processor coupling architecture. Either of positive-true or negative-true path in logic circuits is selected depending on a process variation. Both SRAMs in the PEs are, however, used because their capacities are halved for the low-voltage (we will further explain this in detail in Subsections 2.1) or high frequency operation. In voltage scaling, coupling flip-flops ensure the low-voltage operation.

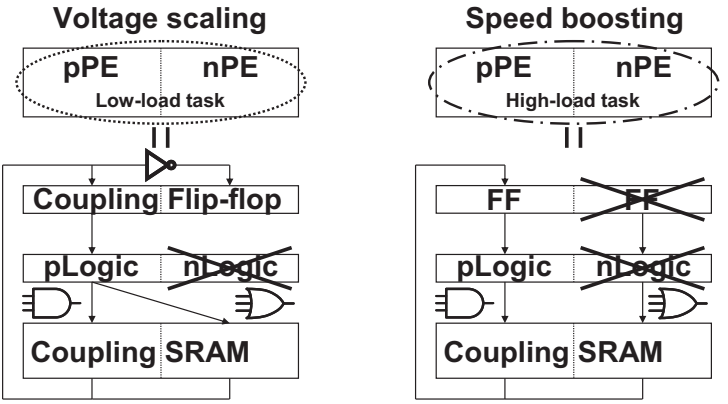


Figure 2: Processor coupling architecture: cases that a positive-true path is selected.

2.1 Coupling SRAM

Fig. 3 depicts coupling SRAM bitcells with a single-port (7T/14T) and dual-port (9T/18T) [4-5]. Two pMOSes are added to internal nodes in a pair of the conventional 6T and 8T bitcells. The coupling SRAM have two modes:

- Normal mode (7T and 9T): The additional transistors are turned off (/CL = "H"). The 7T bitcell acts as the conventional 6T cell.
- Coupling mode (14T and 18T): the additional transistors are turned on (/CL = "L"). Then, the internal nodes are shared by the memory cell pair. By doing so, a larger static noise margin can be obtained when either of the wordlines is merely activated, because a β ratio (a size ratio of a drive transistor to an access transistor) is doubled. When the both wordlines (WL0 and WL1) are activated, a high-speed operation is possible because a cell current is doubled (Fig. 4).

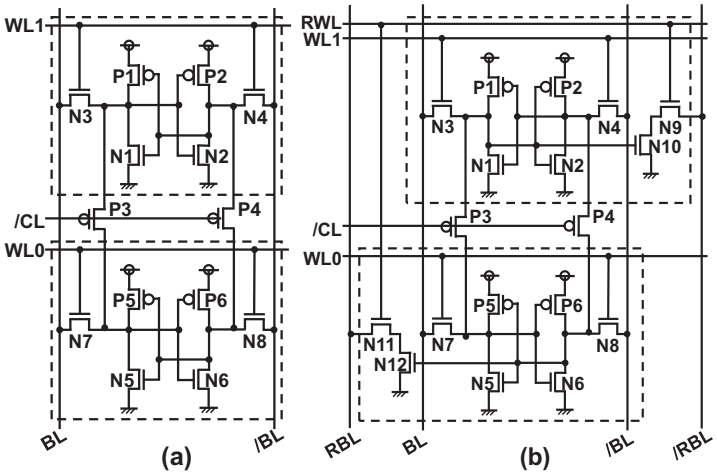


Figure 3: (a) 7T/14T and (b) 9T/18T coupling SRAM.

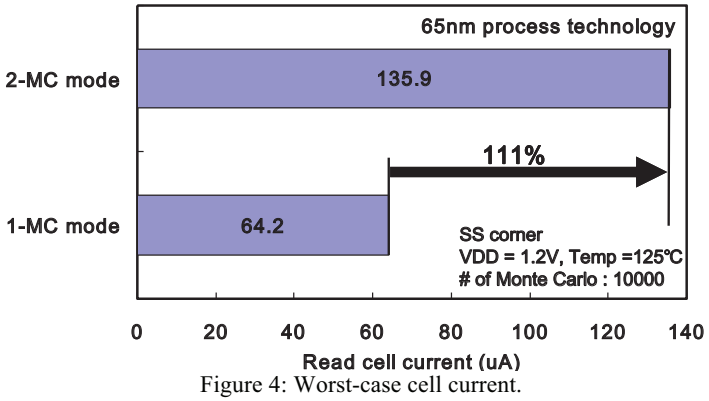


Figure 4: Worst-case cell current.

2.2 Coupling flip-flop

Fig. 5 shows a schematic of a coupling flip-flop pair. An unbalanced FS[1] (nMOS = fast and pMOS = slow) corner is the critical process corner in a low-voltage operation because of latch's data retention. This is similar to the worst retention condition in the 6T SRAM. As shown in Fig. 6, this coupling flip-flop can retain a datum at a voltage of 460 mV in the coupling mode. In the proposed flip-flops, the internal nodes are connected using four nMOS transfer gates, at which the two flip-flops complement the datum each other. The appended four gates are adaptively switched by a control signal (CTRL) according to the operating mode; the two flip-flops can independently operate by turning off the connecting gates in the normal mode.

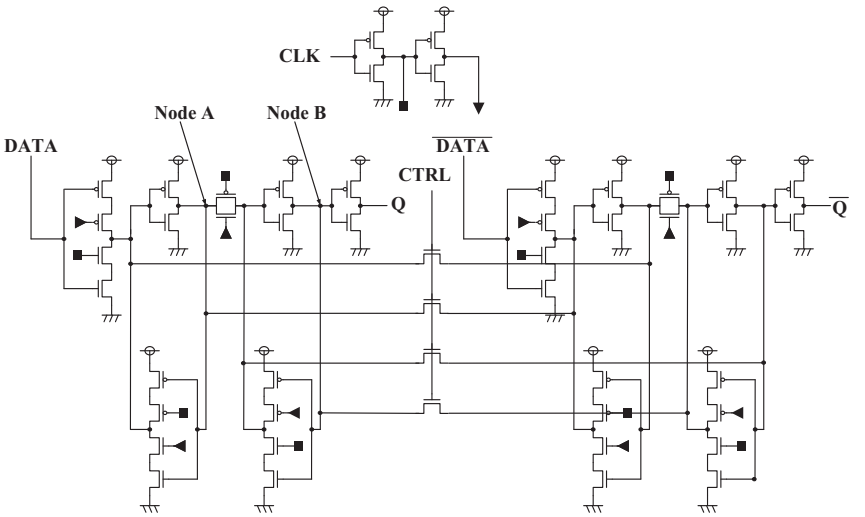


Figure 5: Coupling flip-flops.

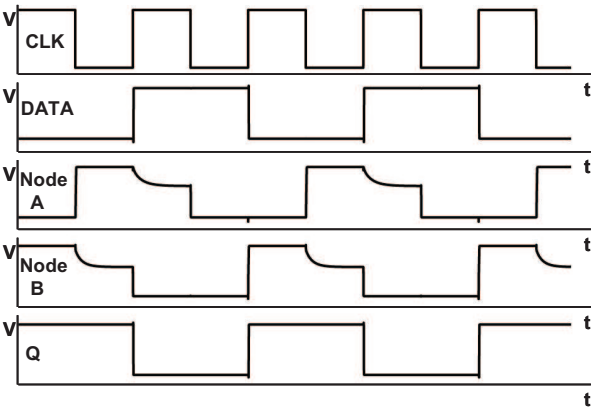


Figure 6: Simulated waveforms in the coupling flip-flops (1 MHz, 460 mV, FS corner).

[1] NMOS and pMOS out of the total variation in chip manufacturing VLSI, called process variation.

We designed and fabricated a coupling flip-flop on a test chip in a 65-nm process technology for measurement and verification. To evaluate behaviors at various process corners, triple-well structure is used for body biasing. In other words, applying body biases to pMOS and nMOS transistors give global Vt variations, which means that we can estimate circuit reliability under the global Vt variations. To guarantee the Vt control accuracy, we implemented a pMOS and nMOS test transistors on the chip for characteristic measurements.

Table 1 presents the body bias settings, at which four process corners (FF, FS, SF, and SS) are emulated, and measured Vmin's in the coupling flip-flop. In the table, ΔVtn (Δ|Vtp|) represents an nMOS (pMOS) transistor's threshold voltage difference from the fabricated CC transistor. Fig. 7 also portrays the measured Vmin when body biasing is applied. From these results, the Vmin of the flip-flop are reduced in the coupling formation at each process corner. This is because the coupling flip-flop improves the data retention characteristic. The voltage scaling mode can be applied to extremely low-power applications; for instance, biomedical sensing, sensor networking, and wearable computing.

Table 1. Body bias settings and measured Vmin in the coupling flip-flops.

Corner	ΔVPW [mV]	ΔVNW [mV]	ΔVtn	Δ|Vtp|	Vmin [mV]	
					Normal	Coupling
CC	±0	±0	±0	±0	384	166
FF	+500	-300	-146	-92	430	260
FS	+450	+400	-97	+67	540	424
SF	-600	-250	+74	-61	434	336
SS	-750	+600	+108	+99	430	210

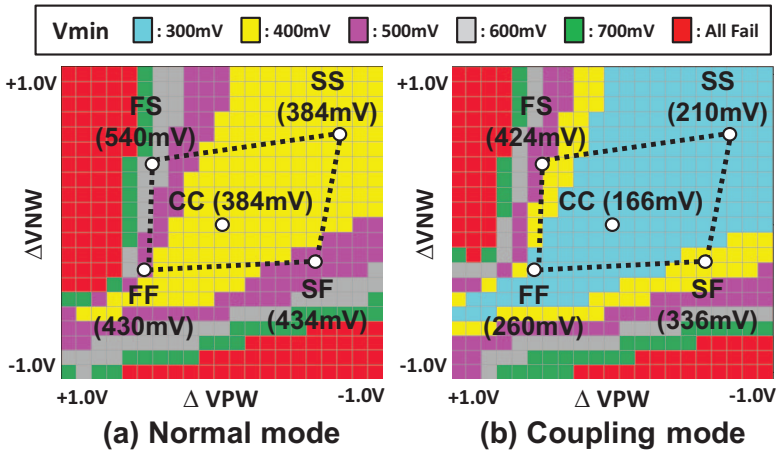


Figure 7: Measured Vmin in the coupling flip-flops at process corners.

2.3 Dual logic circuits

Usually, we consider the SS, TT, and FF (slow, typical, and fast) corners for logic synthesis. As described in the previous subsection, the worst process corner is, however, the unbalanced (FS or FS) corner at the very low supply voltage because the storage circuits like the flip-flop and SRAM are very sensitive to the unbalanced corner.

Fig. 8 illustrates the relationship between the process corner and eligible logic gate. For example, a decoder circuit in Fig. 9 (a) has a long falling time around the SF corner (Fig. 10 (a)) because a 3-input NAND has three stacked nMOS. On the other hand, an NOR logic has a long rising time around the opposite FS corner (Fig. 10 (b)). In our proposed scheme, because all data paths are dual, a better one can be selected according to process variation.

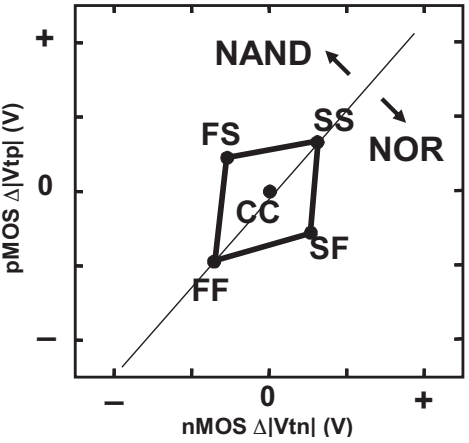


Figure 8: Process corner and eligible logic gate.

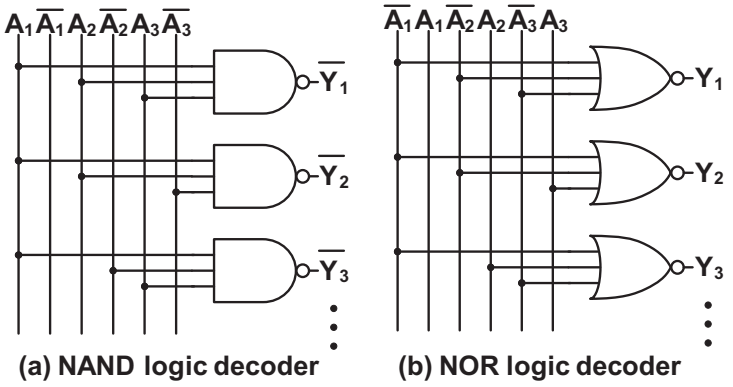


Figure 9: Decoder circuits (3-input NAND and NOR).

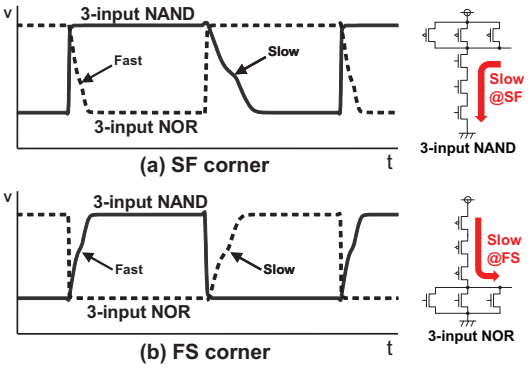


Figure 10: Rising and falling time comparison.

3 APPLICATION EXAMPLES

As an application example of the proposed coupling architecture, we designed octa-core digital signal processor (DSP) for a sound processing unit of microphone array networks.

3.1 Microphone array networks

Recent improvements in information processing technology have produced real-time sound-processing systems using microphone arrays [6]. The microphone array processes signal recordings and also performs noise reduction, sound source separation, speech recognition, speaker identification, and other tasks. To implement a microphone array as a realistic ubiquitous sound acquisition system with scalability, division of the huge array into sub-arrays with a multi-hop network is effective; an intelligent microphone array network was proposed in our previous work [7].

3.2 Coupling DSP multicores

Fig. 11 presents a block diagram of coupling DSP multicores for multi-channel signal processing. To perform SIMD operation for 16-channel sound processing, the proposed DSP consists of three units: instruction issuing unit (called master), sound processing unit, and network processing unit used to control packet and sound data communication.

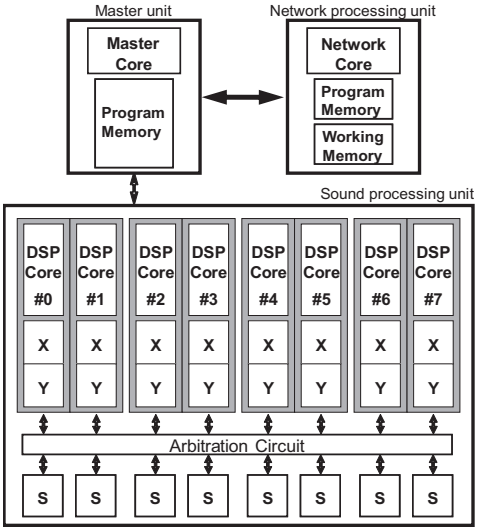


Figure 11: Proposed octa-core DSP multicores.

The master unit has a program memory (24 bits × 2048 word) comprising of the 7T/14T SRAM, decoder circuits, a program counter, and stack registers; it provides a same instruction to eight DSP cores in the sound processing unit. The four coupling DSP cores can be formed by the eight normal DSP cores. Each instruction is executed in six cycles (Fig. 12). The interrupt management and state transition are handled by the master unit.

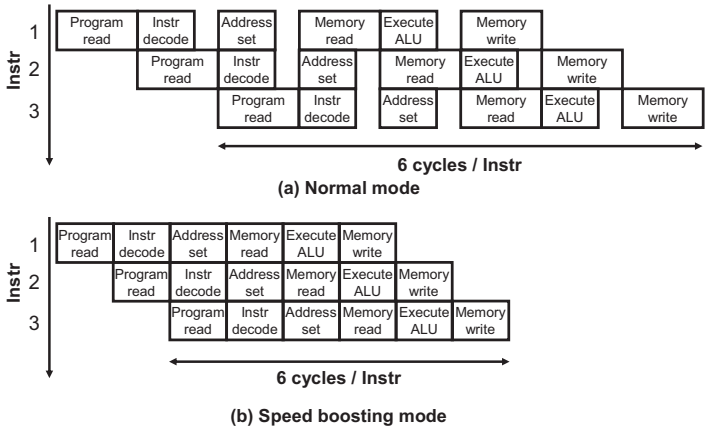


Figure 12: Six-stage pipelined execution.

As shown in Fig. 13, a DSP core has two working memories (X and Y in Fig. 11: each memory has 16 bit × 512 words) to process two-channel sound inputs. The shared-memories (S in Fig. 11) can be accessed from all the DSP cores through arbitration circuits for data exchanging. The X, Y, and shared memories can be accessed at the same time; the MAC and other operations can be executed in one cycle. The X, Y, and shared

memories consist of the 9T/18T dual-port coupling SRAMs to incorporate the dual-port feature for simultaneous read and write.

The working registers (40 bits) are implemented as an accumulator, which is comprised of the coupling flip-flops. As well, the other flip-flops in the DSP multicores consist of the coupling flip-flop.

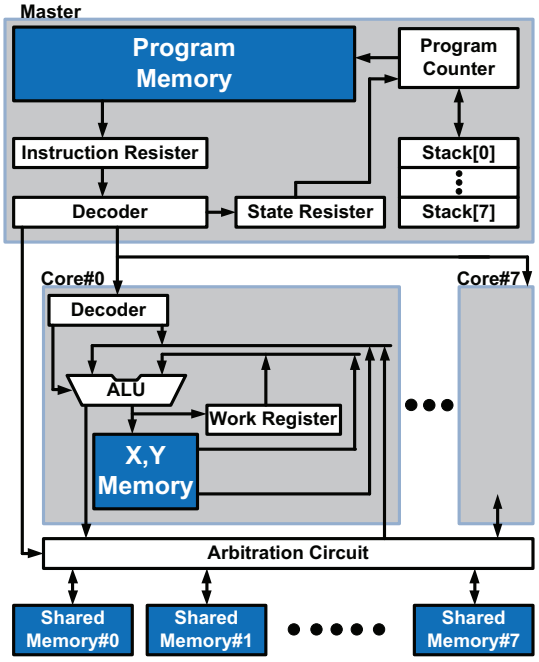
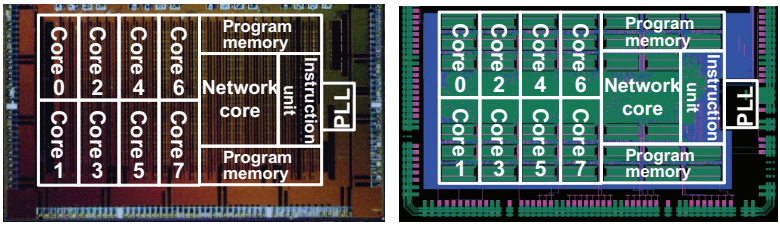


Figure 13: DSP core architecture.

3.3 VLSI implementation

We designed the octa-core coupling DSPs in the 65-nm CMOS process. Fig. 14 depicts the chip micrograph and layout plot image. The core size is 1722 × 2841 um^2. The power in the coupling mode (0.5 V, 1 MHz) is 0.73 mW.



(a) Chip micrograph (b) Layout

Figure 14: The coupling DSP multicores.

Summary

We proposed the processor coupling architecture for voltage scaling and speed boosting. The coupling SRAM achieves the low-voltage/high-speed operations by connecting two bitcells. The coupling flip-flop can run below 0.5 V at any process corner, which enlarges an operating voltage region in random logic circuits. The coupling logic circuits adapt to process variation by selecting an eligible data path. To evaluate the proposed coupling architecture, we designed octa-core coupling DSPs for sound signal processing in the microphone array network application. We fabricated a test chip in a 65-nm triple-well process, and confirmed that, in the coupling mode, the proposed architecture achieves a Vmin of 0.5 V and a power of 0.73 mW at a 1-MHz operation.

Acknowledgment

This research was supported in part by the Semiconductor Technology Academic Research Center (STARC) and KAKENHI (20360161).

References

[1] E. Sperling, “Turn Down the Heat... Please ― Interview with Tom Reeves of IBM,” EDN, July 2006.

[2] J. Kurzak, A. Buttari, P. Luszczek, and J. Dongarra, “The PlayStation 3 for High-Performance Scientific Computing,” Computing in Science and Engineering, pp.84-87, 2008. IEEE International Symposium on Quality Electronic Design (ISQED), pp. 98-102, Mar. 2008.

[3] J. Charles, P. Jassi, N. S. Ananth, A. Sadat and A. Fedorova, “Evaluation of the Intel Core i7 Turbo Boost feature,” In Proc. of IEEE International Symposium on Workload Characterization (IISWC), pp. 188-197, Oct. 2009.

[4] H. Fujiwara, S. Okumura, Y. Iguchi, H. Noguchi, Y. Morita, H. Kawaguchi, and M. Yoshimoto, “Quality of a Bit (QoB): A New Concept in Dependable SRAM,” In Proc. of IEEE International Symposium on Quality Electronic Design (ISQED), pp. 98-102, Mar. 2008.

[5] H. Noguchi, S. Okumura, T. Takagi, K. Kugata, M. Yoshimoto and H. Kawaguchi, “0.45-V Operating Vt-Variation Tolerant 9T/18T Dual-Port SRAM,” In Proc. of IEEE International Symposium on Quality Electronic Design (ISQED), pp. 219-222, Mar. 2011.

[6] M. Brandstein and D. Ward, “Microphone Arrays: Signal Processing Techniques and Applications,” Springer, 2001.

[7] T. Takagi, H. Noguchi, K. Kugata, M. Yoshimoto, and H. Kawaguchi, “Microphone Array Network for Ubiquitous Sound Acquisition,” In Proc. of IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), pp. 1474-1477, 2010.

Exploiting Bit-level Parallelism in GPGPUs: a Case Study on KEELOQ Exhaustive Key Search Attack

Giovanni Agosta, Alessandro Barenghi, Gerardo Pelosi
Dipartimento di Elettronica e Informazione (DEI)
Politecnico di Milano
Via G. Ponzio 34/5, 20133 Milan, Italy
{agosta,barenghi,pelosi}@elet.polimi.it

Abstract: Graphic Processing Units (GPU) are increasingly popular in the field of high-performance computing for their ability to provide computational power for massively parallel problems at a reduced cost. However, the programming model exposed by the GPGPU software development tools is often insufficient to achieve full performance, and a major rethinking of algorithmic choices is needed. In this paper, we showcase such an effect on a case study drawn from the cryptography application domain. The pervasive use of cryptographic primitives in modern embedded systems is a growing trend. Small, efficient cryptosystems have been effectively employed to design and implement keyless password-based access control systems in various wireless authentication applications. The security margin provided by these lightweight ciphers should be accurately examined in light of the speed and area constraints imposed by the target environment. We present a re-design of the ASIC-oriented KEELOQ implementation to perform efficient exhaustive key search attacks while fitting tightly the parallel programming model exposed by modern GPUs. Indeed, the *bitslicing* technique allows the intrinsic parallelism offered by word-oriented SIMD computations to be effectively exploited. Through proper adaptation of the algorithm implementation to a platform radically different from the one it was designed for, we achieved a $\times 40$ speedup in the computation time with respect to a single-core CPU bruteforce attack, employing only consumer grade hardware. The outstanding speedup obtainable points to a significant weakening of the cipher security margin, since it proves that anyone with off-the-shelf hardware is able to circumvent the security measures in place.

1 Introduction

In the last years, Graphics Processing Units (GPUs) have raised wide interest as sources of computational power for non-graphical applications, due to the availability of programming models such as CUDA and OpenCL that are vastly more accessible to experts of other domains than graphics rendering APIs (OpenGL and DirectX) [JDOP08]. A major strength of GPGPU-based platform are their appealing cost-performance figures of merit. In recent times even in the field of High Performance Computing there have been major investments to build GPGPU-based supercomputers.

However, there are also factors that hinder the expansion of GPGPU computing, especially the difficulty of programming efficient applications using the available programming models. Special attention must be placed to tailor the application and its algorithmic

components to the specific needs of the parallel hardware, e.g. by minimizing control flow divergence and exposing as much parallelism as possible while minimizing synchronization overheads [JDOP08]. In this paper, we show how the use of specialized techniques can lead to large speedups, thus allowing the GPU to contend on an equal or favorable base (in terms of computation throughput per euro) with solutions based on CPUs or reconfigurable hardware.

The field of cryptography has been explored since the first GPGPU attempts using graphics rendering APIs [HW07]. Especially, code breaking is attractive [BBAP09, ABSP10, ABS+09], because it requires vast amounts of computational power. We use as a case study the KEELOQ algorithm [Mic11], which is used in remote keyless entry systems (e.g., vehicle doors or building entrances) or as authentication mechanism in wireless protocols.

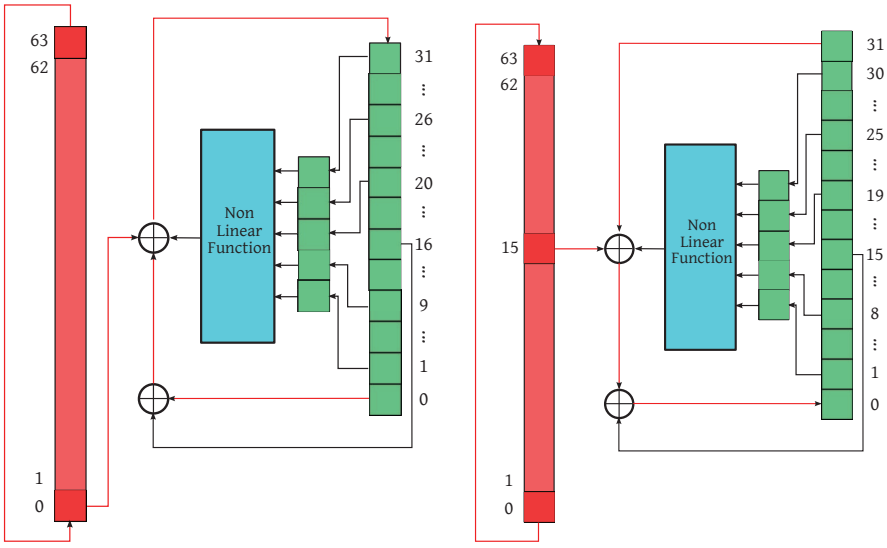
Remote keyless entry systems are based on a password based access control mechanism realized through the unidirectional transmission between a secure token (*encoder*) and a receiver (*decoder*). Unauthorized accesses are possible when the encoded password (*access code*) is fixed or it is derived from a relatively low number of possible combinations. In order to prevent this kind of threat, KEELOQ is employed in the so-called *rolling code* (also known as *hopping code*) mode of operation.

The basic idea is to have the access code change each time it is used through picking it from a sequence of codewords that cannot be predicted even knowing a very large number of previously used ones. The generation of such a sequence is based on the definition of both a uni-directional command transfer protocol and an encryption engine to provide the codewords to be transmitted.

From an operational point of view, the information transmitted by the encoder is composed by two parts: the code-hopping part (which changes each time the encoder is activated) and a second un-encrypted part, principally containing the encoder serial number, used for identifying the transmitter at a receiving decoder. To this end, the receiver decrypts the codeword, and compares the recovered counter value with its internal one, and the recovered serial number with the one received along with the codeword. If both values match, the token is granted access.

Algorithms such as KEELOQ are designed for dedicated hardware implementation, since the target devices (remote controllers) are manufactured as very low cost ASICs. So, their direct implementation in software has much lower performances – which, in principle, makes it easier to carry out an attack using configurable hardware such as FPGAs. However, we show how the introduction of a level of parallelism not commonly seen in GPGPU algorithm design, *bit-level* parallelism, can lead to a $\times 40$ speedup over a CPU core.

The rest of this paper is organized as follows. Section 2 introduces the KEELOQ cipher, while Section 3 reviews the characteristics of the NVIDIA GPU families target in this study, as well as the programming model implemented by the CUDA development tools. Section 4 describes the design of our solution and Section 5 provides the experimental evaluation on the case study. Finally, Section 6 outlines the most closely related works, while Section 7 draws some conclusions.



(a) Encryption (b) Decryption

Figure 1: KEELOQ Cipher

2 The KEELOQ Cipher

KEELOQ is the most scrutinized encryption engine used in remote keyless entry systems. It is a proprietary hardware-dedicated block cipher designed as a pair of Feedback Shift Registers (FSR) coupled with a Non Linear function (NL). Figure 1 shows the internal structure of the KEELOQ cipher: the secret key is stored in the red register on the left and is at most 64-bit wide. The key register is a FSR, and the key is mixed with the output of the state one bit per clock cycle. The 32-bit long Non Linear Feedback Shift Register (NLFSR) on the right hand side constitutes the nonlinear component of the cipher providing its effective security margin. Five bits of the NLFSR are combined together by means of a non linear function described by an equation over $\mathbb{Z}_2$ among five bits of the status register. The non linear function outputs a single bit per clock cycle, which is added to the aforementioned key bit and to b_{16} and b_0, and employed as the feedback bit of the NLFSR. To encrypt a 32-bit plaintext block, the NLFSR is initialized with the value of the plaintext, and subsequently the entire system is clocked 528 times. After the 528 updates of both registers, the content of the NLFSR is the final ciphertext.

The most common mode of operation for KEELOQ is the so-called *hopping code*, in a scenario where a remote *encoder* transmits a codeword to the authorizing *decoder* (receiver). This mode of operation involves encrypting a plaintext built out of a counter and a unique identifier (ID) of the encoding device. Every time a new 32-bit codeword (i.e. a ciphertext block) must be generated, the counter is incremented and the new plaintext is encrypted. Then, the codeword is transmitted along with the encoding device ID. The secret 64-bit key of any encoder is generated through the decoder engine as a pair of 32-bit codewords. Such a procedure implies that the decoder is able to generate the secret keys for a number

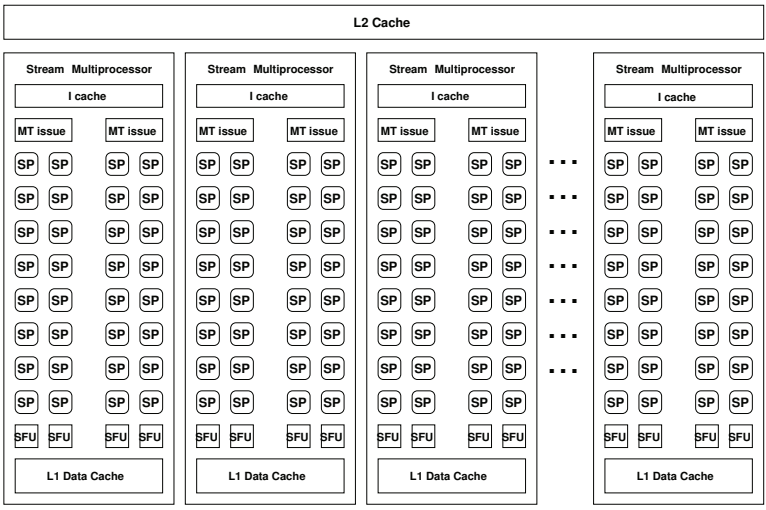


Figure 2: Overview of the NVIDIA GTX470 (Fermi) streaming processors architecture: each stream multiprocessor (SM) contains 32 streaming processors (SP), plus four special function units (SFU). A configurable L1 cache/shared memory is local to each stream multiprocessor, while L2 cache is shared among the entire set of SM. Up to 16 SM can be present in a single unit.

of encoders starting from: *(i)* an embedded 64-bit master key (which is fixed by the manufacturer of the keyless entry system), *(ii)* the ID of the encoding device, *(iii)* and a random seed composed by 32, 48 or 60 bits.

A potential attacker may retrieve the master key from the decoding device (receiver) and eavesdrop the ID of an encoder when it is transmitted along with a codeword. Therefore, the use of a secret random seed in the secret key generation phase avoid the leakage of the secret key of the targeted encoder.

A brute-forcing attack aimed at recovering the secret key of the transmitting encoder (EK) employs two consecutively transmitted codewords, each of which is bound to the encoder ID. The attacker computes a candidate 64-bit value for EK through guessing on the bits of the random seed, while the value of the remaining part of the secret key is easily derived from the specification of the key generation protocol. Subsequently, she checks the ID value resulting from the decryption of the first codeword, and whether a match is found, the output derived from the decryption of the second codeword (employing the same EK) is used as a confirmatory step.

3 General Purpose Computing with GPUs

The GPGPU devices targeted in this work are based on the NVIDIA GT200 and Fermi architectures. Figure 2 shows a sketch of the NVIDIA GTX470 (Fermi) streaming processor array. A streaming multiprocessor (SM) contains 32 streaming processors, four special functional units and a multithreaded instruction issue unit (respectively indicated

as SP, SFU and MT-Issue in Figure 2. This is a fourfold increase over the GT200 SMs. A streaming multiprocessor concurrently executes two groups of 32 threads called *warps*, for a total of 64 concurrent threads. Since each thread in a warp has its own control flow, their execution paths may diverge due to the independent evaluation of conditional statements; when this happens, the warp serially executes each path. Each multiprocessor executes warps much like the *Single Instruction Multiple Data* (SIMD) paradigm, as every thread is assigned to a different SP and every active thread executes the same instruction on different data. Finally, the Fermi architecture includes both L1 and L2 cache memories, with the L1 configurable between cache and shared memory behavior and shared by the SPs in a single SM, and the L2 shared among all SMs in the device. The earlier GT200 only has a fast shared memory shared within each SM.

GPGPU computing requires the programmer to manage a heterogeneous system (CPU host plus GPU device) as well as to handle the massive parallelism exposed by the GPU hardware. The Compute Unified Device Architecture (CUDA) [NBGS08, NVI08], proposed by NVIDIA for its graphics processors starting with the G80 series [ELM08], exposes a programming model that integrates host and GPU code in the same C++ source files. On the GPU device side, a *Single Instruction, Multiple Threads* (SIMT) programming model is exposed, where a single *kernel* is executed by a user-specified number of threads. Every CUDA kernel is explicitly invoked by host code and executed by the device, while the host-side code continues the execution asynchronously after instantiating the kernel. On the host side, a specific synchronizing function call is provided to wait for the completion of the active asynchronous kernel computation.

The CUDA programming model abstracts the actual parallelism implemented by the hardware architecture, providing the concepts of *block* and *thread* to express concurrency in algorithms. A block captures the notion of a group of concurrent threads. Blocks are required to execute independently, so that it has to be possible to execute them in any order (in parallel or in sequence). Therefore, the synchronization primitives semantically act only among threads belonging to the same block. Intra-block communications among threads use the *logical shared memory* associated with that block. Since the architecture does not provide support for message-passing, threads belonging to different blocks must communicate through *global memory*.

Note that while the OpenCL language and API [Khr11] are gaining momentum as the industry standard in programming heterogeneous platforms composed of host CPUs and programmable accelerators, including GPGPUs, the implementations provided are still not mature enough to compete, on NVIDIA devices, with the vendor-specific software development tools. However, the programming model provided in OpenCL is, as far as GPGPU programming goes, essentially based on the same principles as the SIMT model exposed in CUDA, so the techniques and results shown in this work can be easily extended to OpenCL-driven devices.

4 Adaptation to Parallel Architectures

Many-core architectures offer large amount of parallel computing power by supplying the developer with hundreds of processing cores, each endowed with limited resources. In GPGPU, key resource limitations include:

Control flow divergence as multiple divergent control flows can be handled safely from the point of view of functionality, but with major performance losses as parallelism is inhibited along the different control flows – essentially, divergent flows of control are serialized, regardless of the data dependences among the divergent threads (which may well be non-existent). This limitation is due to the hardware design of GPGPU, where the processors in a multiprocessor unit are bound to the same program counter.

Local memory availability as a limited amount of very fast local memory must be shared among numerous processing elements. While the sharing allows fast communication among the processing elements, the local memory is much more useful when used in a read-only way, or partitioned for local use by each processing element, since true shared accesses still require costly synchronization operations, and are often difficult to code.

To exploit such parallel computing power, the critical issue is to be able to express a given application or algorithm in a form amenable to parallel execution on the target device. The literature reports three main sources of parallelism, which can be exploited with different degrees of success on various types of parallel architectures:

Thread-level parallelism is obtained when two or more tasks (regions of code with independent control flow) can be executed in parallel with few or no data dependencies (in the former case, synchronizations will be needed within each task, in the latter the synchronization point will be the end of the tasks). Thread-level parallelism is exposed by complex applications, where multiple independent tasks are performed, and is best exploited on symmetric multiprocessors, where each processor is endowed with sufficient resources to executed its assigned task. It is not suited for GPGPUs, since control flow divergence is a major factor for performance reduction in these architectures.

Loop-level parallelism is found in parallel loop constructs, where each iteration of the loop is data-independent from the others (or has limited synchronization requirements). Loop level parallelism is an excellent fit for vector processors, SIMD processors and GPGPUs, since control is fixed and identical for all iterations (barring nested conditionals, which can often be transformed to predicated code).

Instruction-level parallelism is achieved at the finest of the three common granularities, where independent instructions can be parallelized. It is commonly exploited by superscalar and Very Long Instruction Word architectures, but, like Thread-level parallelism, it is unsuitable for GPGPU due to the need to executed different instructions in parallel, rather than the same instruction of different data.

It would therefore seem that Loop-level parallelism is the only viable choice for GPGPUs, but this model is not exposed by many types of codes. A typical example are encryption primitives designed for hardware implementation. In this case, parallelism is rarely available, but this is not an issue, since the implementation is performed through dedicated ASIC, and may be even considered a benefit, since software implementations are

```
#define NLF   0x3A5C742E
#define bit(x,n)     (((x)>>(n))&1)
#define g5(x,a,b,c,d,e) \
    (bit(x,a)+bit(x,b)*2+bit(x,c)*4+bit(x,d)*8+bit(x,e)*16)

uint32_t  KeeLoq_Decrypt (uint32_t data, uint64_t key){
  uint32_t  x=data, r;
  for (r=0; r<528; r++)
    x= (x<<1)^bit(x,31)^bit(x,15)^(u32)bit(key,(15-r)&63)
        ^bit(NLF,g5(x,0,8,19,25,30));
  return x;
}
```

Figure 3: Plain C KEELOQ implementation [kee11].

often aimed at *breaking* the encryption through brute force attacks. The usage of GPGPUs to perform brute force attacks is well-documented, but is often limited to mere juxtaposition of several encryption operations with different keys. However, it is possible to push the parallelization further, by introducing an entirely different level of parallelism, *Bit-level parallelism*. Here, the goal is to parallelize operations at the single bit level, thereby obtaining remarkably uniform parallel operations. This technique is know as *bitslicing* [Bih97].

Bitslicing refers to a software technique of using a general purpose CPU to implement Single Instruction Multiple Data (SIMD) operations. The strategy consists of packing the bit values belonging to different operands within a single register and of using general-purpose arithmetic/logic instructions as specialized virtual processing elements designed for SIMD operations at bit level. Most of the symmetric cryptographic primitives are designed to process input data at bit level. Therefore, the software implementations of such algorithms on not-specialized architectures may greatly benefit from the application of the bit-slicing strategy as long as the underlying hardware resources in terms of number of registers are easily available.

Figure 3 reports a public domain, plain C implementation of KEELOQ from [kee11], while Figure 4 reports our bitsliced CUDA implementation.

In the case of KEELOQ breaking, the bitslicing technique is employed through the decryption of the same 32-bit value using all possible keys. In the original code, operations on individual bits of the input are performed by means of *shift* and *mask* operations. In the optimized code, all operations work on full 32-bit words. To this end, the 32-bit input data is expanded (on the CPU host side) to a 32-word array, `bitsl_data_in`, where the i-th word in the array is `0xFFFFFFFF` if the i-th bit of the original text is set, or `0x00000000` otherwise. Input data is identical for each thread, but the same is not true for the key and decrypted output, which are stored separately for each of the $n_{threads} \times n_{blocks}$ threads. The bitsliced keys (each of 64 bits) are generated in blocks of 32 keys, starting from zero and progressively increasing its value. Each 64-word array `bitsl_key[bI][tI]` generated in this way has the five last words (corresponding to the lower bits of the original keys) always equal to the encoding of the same 32 values, which are added to a "base" key value. The base key value, in turn, increases in steps of 32. Thus, the number of parallel encryp-

```
__device__ uint32_t bitsl_data_in[32];
__device__ uint32_t bitsl_data_out[NBLOCKS][NTHREADS][32];
__device__ uint32_t bitsl_key[NBLOCKS][NTHREADS][64];

__global__ void KeeLoq_Decrypt_Bitslice() {
  int bI = blockIdx.x, tI = threadIdx.x;
  uint32_t data[32];
  #pragma unroll 32
  for (int j=0;j<32;j++) data[j]=bitsl_data_in[j];

  uint32_t key_r, nlf, rs_data;
  for (int r=0; r<528; r++) {
    key_r = bitsl_key[bI][tI][(15 - r) & 0x3F];
    nlf = NonLinearFunction(data, 0, 8, 19, 25, 30);
    rs_data = data[31] ^ data[15] ^ key_r ^ nlf;
    for (int i=31; i>0; i--) data[i] = data[i - 1];
    data[0] = rs_data;
  }

  #pragma unroll 32
  for (int j=0;j<32;j++) bitsl_data_out[bI][tI][j]=data[j];
}
```

Figure 4: Bitsliced KEELOQ CUDA kernel. The plaintext, key and ciphertext are stored in the GPU main memory at the beginning of the kernel execution, then copied to registers during the kernel itself.

tion runs is 32 per thread, as shown by the kernel in Figure 4, with configurable number $n_{threads}$ of threads per each CUDA block. Overall, a grand total of $32 \times n_{threads} \times n_{blocks}$ encryption runs are performed at every time by the GPU. It is worth noting that the shared memory is not used, since the Fermi architecture provides a large number of registers. A full analysis of the tradeoffs will be shown in the next section.

5 Experimental Results

We implemented a fully bitsliced version of the KEELOQ cipher both employing the CUDA programming model and pure C. The pure C version has been run on the host CPU to provide a reference implementation as far as throughput goes. The running environment where the bruteforcing speed tests were performed is an Intel Core i7 920 based system with 12Gb DDR3 DRAM, running Gentoo Linux AMD64. All the GPU binaries were compiled employing `nvcc` 4.0 from nVidia CUDA toolkit 4.0, while the CPU baseline versions were compiled with `gcc` 4.4.6. The bitsliced implementation of the cipher has been tested on two different GPUs, which have been mounted as the only device on the 16 lane PCI-Express 2.0 port available on the motherboard in order to test the difference in performances. The first GPU card is a GeForce GTX 260 equipped with 894 Mb of GDDR5 video RAM and 192 CUDA cores, while the second card employed for testing is a GeForce GTX 470 with 448 CUDA cores and 1280 MB of GDDR5 video RAM. An important step in the evaluation of the performances of our bitsliced implementation of

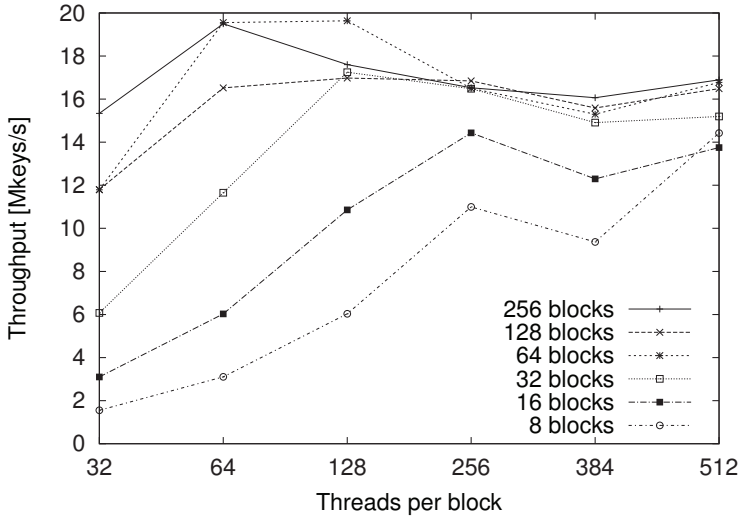


Figure 5: Throughput of the bitsliced implementation of the KEELOQ breaker on the Geforce GTX470 card, related to the number of threads per block and the number of blocks per CUDA kernel invocation

KEELOQ on CUDA is the exploration of two parameters: the number of threads composing a CUDA block and the number of blocks constituting a CUDA kernel call. The first parameter regulates the level of register pressure on the shared register file of the streaming multiprocessor and the number of warps into which a CUDA block is split. Since the basic execution unit of a streaming multiprocessor is a single warp, the choice of the number of threads should consider only multiples of 32 to achieve the best fit. The level of register pressure on the Fermi architecture is dictated by the fact that the 32768 registers are shared among the contexts of up to 3 different blocks which can be scheduled on the same streaming multiprocessor. In addition to this, the SMP issue unit of the Fermi architecture is able to dual issue warps, thus it is necessary to keep twice the contexts in the registers. Combining these data with the fact that a single bitsliced KEELOQ breaking thread employs at most 45 registers, we obtain a SMP register pressure which can be computed as $270 \times n_{threads}$. The second parameter to be chosen regulates the level of global computational load imposed on the GPU. The main point in choosing this parameter is provide at least enough computations to the GPU so that no SMPs remain idle. Moreover, since the SMP issue unit is able to interleave different blocks in order to hide global memory access latencies, it is wise to provide extra workload to the GPU to exploit this feature. These two considerations pointed to the creation of a CUDA kernel as large as possible with architectures up to the GT200, since the static scheduling of the blocks on the SMPs did not account for extra time overhead. With the introduction of a new scheduler for multiple kernels on the Fermi architecture, this consideration may not be still valid. Figure 5 reports the results of the exploration of the implementation parameter space: coherently with the previous considerations, the best solution is reached with 128 threads per block (34560

Seed Length	Single Core Core i7 920 [h]	Four Cores Core i7 920 [h]	Single GPU GTX260 [h]	Single GPU GTX470 [h]
32	2.6	0.73	0.14	0.04
48	$1.73\,10^5$	$4.84\,10^4$	$9.45\,10^3$	$3.98\,10^3$
60	$7.08\,10^8$	$1.98\,10^8$	$3.81\,10^7$	$1.63\,10^7$
Throughput [key/s]	$4.51\,10^5$	$1.61\,10^6$	$8.27\,10^6$	$1.96\,10^7$

Table 1: Expected timings and measured throughput for the exhaustive search of the KEELOQ key generation seed.

employed registers), when the number of blocks per SMP is enough to fill all the issue queues completely. Raising further the number of blocks per kernel leads to a decrease in performances which can be ascribed to the extra context switching effort imposed on the new scheduler. As expected also raising further the number of threads per block leads to a significant decrease in throughput due to the hindering of context switches caused by the frequent register spills and fills. An analogous exploration campaign has been lead also on the GTX260 card, yielding 64 threads per block as the best performing choice of the parameter. This choice is coherent with the fact that the shared register file of the GTX260 is 16384 since 64 threads per block allow the issue unit of the streaming multiprocessor to perform the context switching between the three blocks in queue without the need to spill part of the register file to the global memory. In this case, however, increasing arbitrarily the number of blocks per kernel did not induce any performance penalty as expected from the GT200 architecture. After choosing the optimal number of threads per block and blocks per kernel invocation, we evaluate the effective time needed in order to break the KEELOQ key generation mechanism, with respect to the length of the employed seed. Table 1 reports the expected running times of an attack, depending on the chosen platform to perform the exhaustive search. Taking as a reference value the throughput obtained by the bitsliced implementation of KEELOQ running on the host CPU (419430 keys/s), we notice that employing a 32 bit seed for the key generation does not yield a sufficient security margin, as the remote key can be recovered in 3 hours of computation. The bitsliced implementations running on the GTX260 and GTX470 GPUs achieve a $\times 20.5$ and a $\times 43.5$ speedup respectively, allowing a possible attacker to breach even the security of the 48 bit seed key generation mechanism in a few months. Since the exhaustive search can be split over multiple GPUs, it is possible to lower the attack time to a single week, while keeping the cost envelope of the equipment below $10000, as this budget allows an attacker to build a 20 GTX 470 cluster with the current market prices.

6 Related Work

The first cryptanalysis of KEELOQ is presented in [Bog07]. The attack is based on the *slide technique* and a linear approximation of the non-linear Boolean function used in the cryptographic engine. The attack requires 2^{52} encryptions, 16GB of storage and the entire

codebook, i.e., 2^{32} known plaintexts. In [IKD+08] the authors introduce a specific key recovery attack against KEELOQ which combines the technique of slide attacks with a novel meet-in-the-middle approach. Their method requires 2^{16} chosen plaintexts and has a time complexity of $2^{44.5}$ encryptions which results in about two days of computation employing 50 dual core CPUs at the cost of approximately €10000. The widely adoption of KEELOQ in practice, paved the way to side-channel analysis as a further viable option for attacking chips that implement it. In [EKM+08] the first successful DPA and DEMA attacks on KEELOQ implementations applied to both Identify Friend or Foe (IFF) and code hopping devices, are presented. The attack is prevented if a 60-bit seed value, with good random properties, is employed for the key derivation. Nevertheless, considering the other commonly implemented options of the cipher, the authors reported how to reveal a manufacturer key from a receiver using a few 1000 power traces, and how to recover the device key of a remote control with as few as 10 traces. In [CBW08] the authors apply algebraic techniques to cryptanalyze the cipher. This attack employs the entire codebook, 2^{27} encryptions and has an estimated success probability of 44%. The results of a brute-force attack, implemented on the FPGA-based code-breaker COPACOBANA, are reported in [NK09]. The authors claim the secret key recovery of a remote control in less than 0.5 seconds if a 32-bit seed is used and in less than 6 hours in case of a 48-bit seed. The case of a 60-bit seed needs in the worst case about 1011 days at the cost of approximately $10000. However, the technical effort needed to build an FPGA-based code-breaker, and even more one the size of the COPACOBANA, is much greater than that needed to carry out a GPU-based attack. Moreover, while FPGA-based code-breakers require specialized hardware, the GPU-based attack can benefit of a large installed base of CUDA-enabled devices, allowing distributed attacks to be carried out by groups of users, or by botnets.

7 Conclusions

In this paper, we report our experience with *bit-level parallelism* in GPGPU programming, using as a case study the brute force attack on the KEELOQ cipher. We proposed a full redesign of the computation strategy from the original hardware implementation-oriented algorithm to reach high performance in parallel software, by exploiting SIMD techniques down to the bit level. We report a speedup of $\times 40$ speedup in the computation time with respect to a CPU brute force attack, even though only consumer-grade hardware is used.

References

[ABS+09] Giovanni Agosta, Alessandro Barenghi, Fabrizio De Santis, Andrea Di Biagio, and Gerardo Pelosi. Fast Disk Encryption through GPGPU Acceleration. In *PDCAT*, pages 102–109. IEEE Computer Society, 2009.

[ABSP10] Giovanni Agosta, Alessandro Barenghi, Fabrizio De Santis, and Gerardo Pelosi. Record Setting Software Implementation of DES Using CUDA. In Shahram Latifi, editor, *ITNG*, pages 748–755. IEEE Computer Society, 2010.

[BBAP09] Andrea Di Biagio, Alessandro Barenghi, Giovanni Agosta, and Gerardo Pelosi. Design of a parallel AES for graphics hardware using the CUDA framework. In *IPDPS*, pages 1–8. IEEE, 2009.

[Bih97] Eli Biham. A Fast New DES Implementation in Software. In Eli Biham, editor, *FSE*, volume 1267 of *Lecture Notes in Computer Science*, pages 260–272. Springer, 1997.

[Bog07] Andrey Bogdanov. Linear Slide Attacks on the KeeLoq Block Cipher. In Dingyi Pei, Moti Yung, Dongdai Lin, and Chuankun Wu, editors, *Inscrypt*, volume 4990 of *Lecture Notes in Computer Science*, pages 66–80. Springer, 2007.

[CBW08] Nicolas Courtois, Gregory V. Bard, and David Wagner. Algebraic and Slide Attacks on KeeLoq. In Kaisa Nyberg, editor, *FSE*, volume 5086 of *Lecture Notes in Computer Science*, pages 97–115. Springer, 2008.

[EKM+08] Thomas Eisenbarth, Timo Kasper, Amir Moradi, Christof Paar, Mahmoud Salmasizadeh, and Mohammad T. Manzuri Shalmani. On the Power of Power Analysis in the Real World: A Complete Break of the KeeLoq Code Hopping Scheme. 5157:203–220, 2008.

[ELM08] Stuart Oberman Erik Lindholm, John Nickolls and John Montrym. NVIDIA Tesla: A Unified Graphics and Computing Architecture. *Micro, IEEE*, 28(2):39–55, 2008.

[HW07] Owen Harrison and John Waldron. AES Encryption Implementation and Analysis on Commodity Graphics Processing Units. In Pascal Paillier and Ingrid Verbauwhede, editors, *CHES*, volume 4727 of *Lecture Notes in Computer Science*, pages 209–226. Springer, 2007.

[IKD+08] Sebastiaan Indesteege, Nathan Keller, Orr Dunkelman, Eli Biham, and Bart Preneel. A Practical Attack on KeeLoq. In Nigel P. Smart, editor, *EUROCRYPT*, volume 4965 of *Lecture Notes in Computer Science*, pages 1–18. Springer, 2008.

[JDOP08] David Luebke Simon Green John E. Stone John D. Owens, Mike Houston and James C. Phillips. GPU Computing. *Proceedings of the IEEE*, 96(5):879–899, May 2008.

[kee11] C KEELOQ Implementation. [On line] `http://cryptolib.com/ciphers/keeloq/KeeLoq.c`. December 2011.

[Khr11] Khronos OpenCL Working Group. OpenCL–The Open Standard for Parallel Programming of Heterogeneous Systems. [On line] `http://www.khronos.org/opencl/`, January 2011.

[Mic11] Microchip Technology Inc. Security and Authentication Design Center–KEELOQ© 3 Development Kit. [On line] `http://www.microchip.com/stellent/idcplg?IdcService=SS_GET_PAGE&nodeId=2074`, December 2011.

[NBGS08] John Nickolls, Ian Buck, Michael Garland, and Kevin Skadron. Scalable parallel programming with CUDA. *ACM Queue*, 6(2):40–53, March 2008.

[NK09] Martin Novotny and Timo Kasper. Cryptanalysis of KeeLoq with COPACOBANA. In *Workshop on Special Purpose Hardware for Attacking Cryptographic Systems (SHARCS'09)*, 2009.

[NVI08] NVIDIA Corporation. CUDA Technology. [On line] `http://www.nvidia.com/CUDA`, September 2008.

Fast Scenario-Based Design Space Exploration using Feature Selection

Peter van Stralen, Andy Pimentel

Insititute for Informatics, University of Amsterdam
{p.vanstralen,a.d.pimentel}@uva.nl

Abstract: This paper presents a novel approach to efficiently perform early system level design space exploration (DSE) of MultiProcessor System-on-Chip (MPSoC) based embedded systems. By modeling dynamic multi-application workloads using application scenarios, optimal designs can be quickly identified using a combination of a *scenario-based DSE* and a feature selection algorithm. The feature selection algorithm identifies a representative subset of scenarios, which is used to predict the fitness of the MPSoC design instances in the genetic algorithm of the scenario-based DSE. Results show that our scenario-based DSE provides a tradeoff between the speed and accuracy of the early DSE.

1 Introduction

A significant amount of research has been performed on system-level DSE for MPSoCs [Gri04] during the last two decades. The majority of this work is focused on the analysis of MPSoC architectures under a single, static application workload. The current trend, however, is that application workloads executing on embedded systems become more and more dynamic. This is not only the time-dependent behavior of a single application, the interaction between different applications can also be hard to predict. This dynamic behavior can be classified and captured using so-called *workload scenarios* [G+09].

In this paper, we propose a novel scenario-based DSE method that allows for capturing the dynamic behavior of multi-application workloads in the process of system-level DSE. An important problem that needs to be solved by such scenario-based DSE is the rapid evaluation of MPSoC design instances during the search. Because the number of different workload scenarios can be large, it is infeasible to rapidly evaluate a MPSoC design instance during DSE by exhaustively analyzing (e.g., via simulation) all possible workload scenarios for that particular design point. As a solution, a *representative subset of workload scenarios* can be used to make the evaluation of MPSoC design instances as fast as possible. The difficulty is that the representativeness of a subset of workload scenarios is dependent on the target MPSoC architecture. But since the evaluated MPSoC architectures are not fixed during the process of DSE, we need to *simultaneously co-explore* the MPSoC design space and the workload scenario space to find representative subsets of workload scenarios for those MPSoC design instances that need to be evaluated. To this end, we pro-

pose a scenario-based DSE method combining a multi-objective genetic algorithm (GA) and a feature selection algorithm.

The remainder of this paper is organized as follows. In the next section, we briefly summarize the concept of scenario-based DSE. Section 3 describes the proposed framework to perform scenario-based DSE. In Section 4, we present experimental results in which we evaluate our scenario-based DSE approach, and also compare our method to a coevolutionary scenario-based DSE approach. Section 5 discusses related work, after which Section 6 concludes the paper.

2 Scenario-based Design Space Exploration

With the introduction of multiple applications that can execute (possibly simultaneously) on a single embedded system, the interactions on the system become more complex. If traditional DSE is applied, there are two options: 1) isolate the different applications on the MPSoC platform or 2) design the system for the situation where all the applications are running simultaneously. In both situations, the system will typically be over-designed. In case of an isolated design, each application must run on a separate part of the platform to prevent interaction at runtime. This can take up more resources on the MPSoC than required. The same is true for running all applications simultaneously, as they must meet their deadline while assuming that all applications are running. To avoid such worst case design, and to optimize the system for cost and efficiency, we propose to model the interactions of the applications using workload scenarios. Workload scenarios allow the designer to exactly describe the usage of the system. To prevent over-design, scenarios can describe which applications can be active simultaneously.

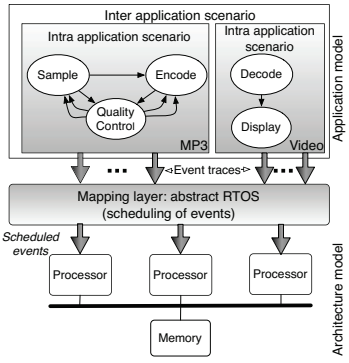


Figure 1: High level scenario-based MPSoC simulation

More precisely, scenario-based DSE bundles two concepts. The first concept is the early DSE of MPSoC based embedded systems. During such early DSE, MPSoC platform instances are quickly evaluated in order to find a set of promising candidate designs. For evaluation purposes, we are using the high-level MPSoC simulation framework Sesame [PEP06]. This framework, which is illustrated in Figure 1, enables fast performance evaluation using separate application and architectural models. An application model describes an application using a Kahn Process Network (KPN), while the architecture model models the MPSoC architecture in a cycle-approximate fashion. Subsequently, there is an explicit mapping of the application model(s) onto the architecture model, implemented using trace-driven co-simulation of the two aforementioned models. Mapping solves two aspects concurrently: 1) *allocation* and 2) *binding*. Allocation selects the architectural components used on the MPSoC platform, whereas the binding defines on which architectural component the application tasks

and communications are executed. During the evaluation of a mapping, each process in an application model generates a trace of application events, representing the application workload at a high level of abstraction. These event traces are simulated by the architecture model to obtain non-functional metrics like execution time and energy consumption.

The other concept that is used in scenario-based DSE is the deployment of workload scenarios [G+09]. In this work, we distinguish two types of workload scenarios: intra and inter application scenarios. Intra-application scenarios describe the different behaviors, or operation modes, within an application. For example, an MP3 application can play music in mono or stereo sound. An inter-application scenario describes the behavior of multiple applications. In our case, it specifies which applications can run concurrently. In the example of Figure 1, the inter-application scenario describes simultaneous execution of the MP3 application and the video application. In order to fully describe what a system is doing, a complete *application scenario* bundles the possible intra-application scenarios of all the active applications. The set of active applications is described using an inter-application scenario, whereas each intra-application scenario specifies a particular operation mode of an individual application. An example application scenario in Figure 1 is that the MP3 application is playing music in mono sound, while the video decodes at a low bitrate.

Incorporating the concept of application scenarios in the process of early DSE of MPSoCs should pave the road for exploring modern embedded systems that are executing dynamic multi-application workloads. Earlier work in this direction [vSP10a] already showed that a random pick of a set of application scenarios does not result in the best set of candidate design instances. Therefore, scenario-based DSE aims at efficiently searching for a set of candidate MPSoC design instances that perform well when considering all the potential situations that can occur in a dynamic multi-application workload.

3 Proposed Framework

In this paper, we propose a new framework for scenario-based DSE. The framework is illustrated in Figure 2. The left-hand side of the diagram shows the general flow of our scenario-based DSE. As input, the description of the target multi-application workload and a MPSoC platform are supplied, as well as search parameters for the GA.

The multi-application workload is described using KPN application models for each separate application in the multi-application workload and a scenario database in which the inter and intra application scenarios are made explicit. For the latter, we use the work from [vSP10b], in which a scenario-aware extension of the Sesame simulation framework is presented. This means that the input KPN application models specify the structure of each individual (concurrent) application allowing for mapping exploration of the application tasks, while the different possibilities for multi-application workload behavior (e.g., which applications or application modes are active at the same time) are characterized in the scenario database.

As was mentioned earlier, the number of different application scenarios can be very large. Consequently, it is infeasible to rapidly evaluate MPSoC design instances during early

DSE by exhaustively analyzing all possible workload scenarios for these particular design instances. As a solution, a *representative subset of application scenarios* can be used. The representativeness of a subset of application scenarios is, however, dependent on the target MPSoC architecture, which is in our case not fixed as we are performing MPSoC DSE. The solution is to co-explore the MPSoC design instances and the representative subset. The right-hand side of Figure 2, which zooms in on our scenario-based DSE, shows the two co-exploration processes. One process, called the *design explorer*, is similar to classical MPSoC DSE as it uses a metaheuristic, which is in our case a GA, to search for optimal mappings (i.e., optimal MPSoC design instances). The second process, referred to as the *subset selector*, tries to identify the best representative subset of application scenarios. As will be explained later on, the subset selector is implemented using feature selection. Both processes are asynchronous, and inter-process communication is performed via shared memory.

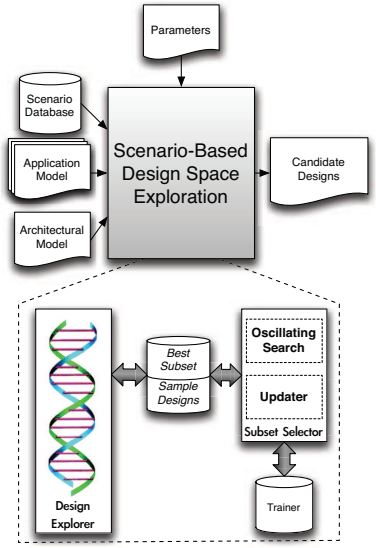


Figure 2: The framework for scenario-based DSE using feature selection

The primary reason for asynchronous execution is that both processes are independent. To provide a trade-off between DSE and subset training effort, the user can manually assign a fixed number of worker threads to the design explorer and subset selector. After termination of the design explorer, the scenario-based DSE finishes by returning a Pareto front of MPSoC design instances (in the case of this paper a trading off performance, energy consumption and cost).

3.1 Design Explorer

The design explorer is responsible for searching and identifying optimal MPSoC design instances by exploring different mappings of the multi-application workload on the MPSoC platform architecture model. To actually perform the search, a NSGA-II [D$^+$02] based multi-objective GA is used. NSGA-II is an elitist selection algorithm that uses non-dominated sorting to select the offspring individuals. An example of non-dominated sorting is given in Figure 4. Non-dominated sorting ranks all the design points based on their dominance depth [C$^+$07]. Conceptually, this dominance depth can be obtained by iteratively obtaining and removing the set of non-dominated individuals. The main reason of choosing NSGA-II is its use of the dominance depth for the optimization. In the subset selector, the scalar value of the dominance depth can easily be used for rating the subset quality independent of the number of objectives. This will be explained in the next subsection.

In order to couple the GA to the feature selection algorithm inside the subset selector, some communication is required. Figure 3 shows the extended algorithm of the GA. Before any evaluation can be done, the current representative subset must be retrieved from the shared memory. Using the obtained subset of application scenarios, the candidate mappings can be quickly evaluated using Sesame [PEP06]. The design explorer has a pool of worker threads, where each thread can simulate and evaluate a single mapping in parallel, to fully utilize multi-core systems. A sample of the simulated mappings is sent to shared memory. After the evaluation, the GA can select individuals based on their *estimated fitness*. In case of a representative scenario subset, the decisions made by the NSGA-II selector are similar to the case where the *real fitness* would have been used. The real fitness is equal to the average fitness of all possible application scenarios as stored in the scenario database. The selected individuals can be used for reproduction after which the whole procedure will repeat itself in the next generation.

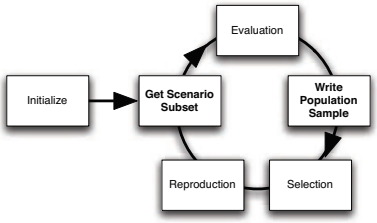


Figure 3: The GA to find the optimal mapping extended with steps to communicate data with the subset selector

3.2 Subset Selector

To obtain an estimated fitness of the MPSoC design instances in the GA population of the design explorer, a good representative subset of scenarios is required. We have chosen to select this representative subset of scenarios using feature selection. Feature selection is a technique of selecting a subset of relevant features for building classifiers. This fits our needs well since we want to select a set of application scenarios (the features) to classify the dominance depth of a mapping (i.e., a MPSoC design instance). An important consequence of this decision is that we need to have a set of training mappings of the multi-application workload from the *design explorer* to be able to score a subset of scenarios within the *subset selector*. Moreover, we need to have the real fitness in order to judge if the estimation of the dominance depth is correct.

For this purpose, we added a trainer. The trainer keeps a small set of mappings which are exhaustively evaluated with Sesame (using all application scenarios in the scenario database) to obtain their real fitness. The trainer is updated at runtime, using the current sample of mappings originating from the design explorer. In this way, we train the scenario subset exactly with respect to the part of the design space where the design explorer is currently searching. To isolate the exhaustive mapping evaluation from the GA in the design explorer, we have introduced a separate pool of worker threads in the subset selector. These worker threads are managed by the ”updater thread”. This updater thread will read out the sample mappings from the shared memory. Based on their estimated fitness, it will select a small portion of interesting mappings to add to the trainer, from which the real fitness is determined. To fully parallelize the simulation of these mappings (i.e., exploiting multi-core host systems), we do not only evaluate the different mappings in parallel,

but also the total set of application scenarios is divided in multiple chunks. In this way, a single mapping can be evaluated quicker using multiple worker threads, each simulating a different scenario for the respective mapping.

Simultaneously with the updater thread, a feature selection algorithm is executed. Every time this algorithm finds a better subset, it updates the best subset in the shared memory. We have chosen to use the sequential oscillation search [SP00] as our feature selection algorithm. Oscillating search uses a kind of hill climbing technique to improve the current subset of scenarios. We have chosen to use the oscillating search for two reasons. As it iteratively improves the scenario subset, it can provide premature results that can already be used in the design explorer. Additionally, it starts from an existing subset. As a result, we can keep using the current representative subset after the trainer has been updated.

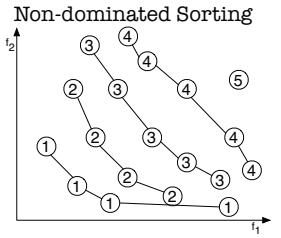


Figure 4: An example of non-dominated sorting. All design points are annotated with their dominance depth.

The *misclassification rate* is used to judge the quality of the scenario subset. The misclassification rate is equal to the percentage of incorrectly predicted dominance depths (Figure 4). The lower the misclassification rate, the better the subset is able to identify the Pareto front that consists of the non-dominated solutions. The benefits of using the dominance depth are twofold. First, the better the estimation of the dominance depth is, the better our selection algorithm NSGA-II will perform its work. Second, the quality metric of the subset is independent of the number of the objectives in the mapping space.

At the startup phase, the trainer is empty. Therefore, a random subset is created as the quality of the subset can not be judged. Once the updater thread has issued mappings to the trainer, the sequential oscillating search can be started. When a better subset is found during this search, it is communicated to the mapping space using the shared memory. Periodically, the updater thread has a new group of exhaustively evaluated mappings for the trainer. In this case, the oscillating search is temporarily halted.

4 Experiments

To verify our framework, a number of experiments have been performed with a stochastic multi-application workload with 10 different applications. The multi-application workload with 4607 application scenarios is generated using a Python tool that is loosely based on [Ors09]. We decided to use stochastic applications because it provides us full control over the workload. On top of that, stochastic applications can be instrumented to mimic the abstract behavior of real applications. The workload is mapped onto a MPSoC platform with four general purpose processors, two ASIPs and two ASICs. In addition to these processing elements, the platform contains a shared memory, a crossbar network, and four point-to-point FIFO communication channels. As our framework performs mapping DSE, handling both the allocation and the binding, not all the components in the platform are used in the final design. As a result, we consider three objectives: execution time, energy

and cost (with respect to silicon area).

The early DSE of an embedded system like this can only be done using heuristic search methods such as a GA. The number of possible mappings is exponential with respect to the number of potential architectural components. Our example multi-application workload has more than $6.8 * 10^{58}$ different mappings onto the architecture. Moreover, the exponential number of scenarios (with respect to applications) makes fitness prediction – based on a representative subset of scenarios – a necessity for a feasible early DSE. Two types of experiments have been performed. The first experiment focusses on the feature subset selection. Based on the results of this experiment, we have performed our second experiment in which we demonstrate the functionality of the scenario-based DSE.

4.1 Subset Selection

One of the parameters of our framework is the size of the scenario subset. This size involves a tradeoff with respect to quality and speed. The smaller a subset is, the faster our early DSE will be performed. On the other hand, a larger subset will result in a better fitness prediction and thus a better average outcome of the GA. The potential design space of possible subsets is huge: in our example multi-application workload there are already $6.6 * 10^{42}$ different subsets of size 15.

To investigate the exact influence of the subset size, we have applied our subset selector in isolation. For this purpose, the updater thread is disabled and replaced by a fixed trainer. The fixed trainer is selected in beforehand and contains 518 exhaustively evaluated mappings. As a result, we are able to investigate the properties of the subset selection without any external influences. We have restricted the execution time of the subset selector to 10 minutes (wallclock time). A longer execution time is likely to get better scenario subsets, but in early DSE the time available for training is limited.

4.1.1 Quality versus size

In our framework, we have decided to perform feature selection with a deterministic algorithm. To substantiate this decision, we have compared our subset selector using a deterministic feature selection (FS) algorithm to a genetic algorithm (GA) approach (comparable with the subset search method in [vSP10a]). Figure 5 shows the result of the experiment. The horizontal axis shows the wallclock time in seconds, whereas the vertical axis shows the quality of the best subset at that point in time. Quality is defined by the misclassification rate: the lower the number of incorrectly predicted dominance depths, the better the subset is. Each time a new subset is found, a marker is placed. The experiment is done for different subset sizes, where each line refers to the misclassification of a subset with a specific size.

Both approaches start with random subsets and therefore their initial quality is similar. This can be clearly seen in the figure. In general, a larger subset means a better quality. A subset with a size of 60 starts with a misclassification rate of 54 percent, whereas a subset with 15 scenarios incorrectly classifies 68 percent of the individuals. During the startup

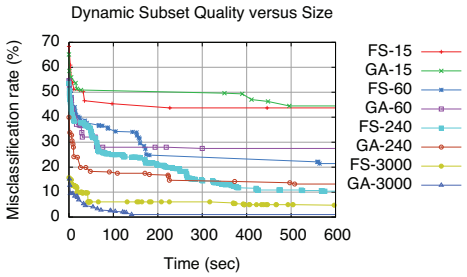


Figure 5: The relation between the quality of the subset and its size for feature selection (FS) and a genetic algorithm (GA)

phase of the search, the GA-based selector has better quality subsets than our FS-based subset selector. The oscillating search applies a hill climbing technique. As a result, the changes to the subset are minor and the subset can only be changed slowly. The GA, on the other hand, can apply crossover to quickly change large parts of the representative subset of scenarios. However, the deterministic approach of our FS-based subset selector will eventually outperform the GA. For a subset of size 15, it takes only 30 seconds. This time grows to 170 and 300 seconds for the subset size of 60 and 240 scenarios, respectively. As comparison, a large subset of 3000 scenarios is also added. Within 600 seconds, our subset selector does not outperform the GA-based approach of [vSP10a]. However, after a certain period it is expected to outperform the GA.

Since a larger subset means a larger evaluation time of the design explorer, the better eventual quality of the subset is more important than the better selection of the approach of [vSP10a] in the startup phase. In this phase, the quality improvements in the MPSoC designs (execution time, energy, etc.) are still quite significant and it is relatively easy to discriminate between different MPSoC designs. Later in the DSE, differences become smaller, but at this point in time our deterministic subset selector will have a better representative subset than is achieved with the stochastic GA-based approach. The larger the representative subset is, the more time there is to find a high quality representative subset. This large representative subset results in a longer evaluation period in the design explorer, resulting in more time for the subset selection.

Based on the above results, we have decided to use a subset with 240 scenarios. We believe that it provides a good trade-off between size (fast evaluation time in the design explorer) and quality. Compared to the large subset of 3000 that achieves a misclassification of 5%, a subset of 240 scenarios drops below 15% within 5 minutes. After five minutes, the trainer will likely be updated and the subset can quickly be adjusted for better representativity.

4.1.2 Misclassification: how does it look like

To give a feeling of how the misclassification looks like, consider Figure 6. This figure uses a density graph for the relationship between the predicted and the real values of the dominance depths. The color legend (density) shows the number of mappings that have a predicted dominance depth of x (the x-axis) and a real dominance depth of y (the y-axis). To get this relationship, we used a set of 1933 mappings which is disjunct to the set of mappings with which the scenario subset is trained. Ideally, the predicted and real dominance depth are equal. In the density graph, this would manifest itself with a straight line ($y = x$). In our experiment, this straight line is also visible. However, the line becomes wider for high dominance depths.

In our case, we are interested in good mappings. These mappings have a dominance depth that is as low as possible. Looking at the real Pareto front (the mappings with dominance depth 1), Figure 6 indicates that 84 percent is classified correctly. Looking at the other 16 percent, we see that 14 percent has a predicted dominance depth of 2. Once the population of the NSGA-II based GA is large enough, it is quite likely that also these mappings are added to the population of individuals that is carried to the next generation. The larger discrepancy between prediction and reality is mainly observed at the individuals with a high value of the real dominance depth. This is, however, not problematic for our case, as we are not interested in low quality mappings anyway.

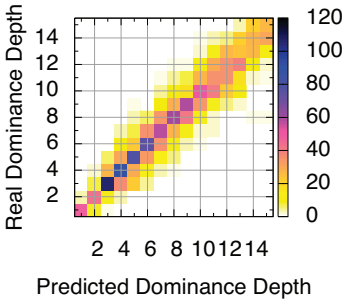


Figure 6: A density graph with the relationship between the predicted and the real dominance depth.

4.2 Design Space Exploration

Now that the subset selector has been validated, the complete framework can be validated. This is done using two experiments. First, we will show that the scenario-based DSE with feature selection can deliver results much faster than traditional early DSE. In such traditional DSE, MPSoC design points in the design explorer's GA are evaluated exhaustively using all application scenarios. Subsequently, in the second experiment, we investigate the quality of our proposed scenario-based DSE method.

The parameters of the GA in the design explorer are fixed for all experiments. The GA uses a population size of 300 and an offspring size of 100. For the fitness prediction in the experiments with scenario-based DSE, we use a subset of 240 scenarios. To compare the quality of the resulting Pareto fronts, we use the hypervolume [ZT98] metric. The hypervolume is equal to the area of the Pareto front and the larger it is, the better the Pareto front. We have normalized the hypervolume values such that the *combined* Pareto fronts of *all* performed experiments equal to a hypervolume of 1.0.

4.2.1 The evaluation time of early DSE

In the first experiment the evaluation time of traditional DSE and the proposed scenario-based DSE with feature selection (FS) is compared. The traditional DSE uses 12 threads to evaluate the individuals in the GA, whereas the scenario-based DSE with FS uses 10 threads for the design explorer (the mapping GA) and 2 threads for the subset selector. This ratio determined using calibration of time versus precision. The experiment is done on a dedicated server with 2 Intel six-core Xeon L5640 processors running at 2.26Ghz.

From Figure 7, it can be clearly seen that our framework is much faster than the traditional DSE. While the traditional DSE only performs 34 generations, the scenario-based DSE with FS can execute 300 generations. As a consequence, the resulting Pareto front of the

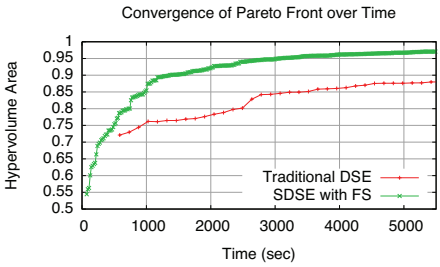


Figure 7: Time-based comparison

traditional DSE after 90 minutes is clearly outperformed by the faster scenario-based approach. We would like to note, however, that although the convergence rate of the traditional DSE looks more slowly and bumpy, this is only due to the small number of generations. The scenario-based DSE is capable of doing more generations, resulting in a smoother and faster convergence. However, it is as expected that the final result of the exact traditional DSE is better as can be seen in Figure 8.

For early DSE, it is key that a large number of candidate designs can be rapidly evaluated. Hence, the traditional DSE method takes too much time (12 hours and 40 minutes) for a quick feedback loop in which the designer is able to manipulate the application(s) and the MPSoC components based on the candidate designs. Moreover, as the number of applications grows, the larger the gap between the evaluation time of the scenario-based DSE and the traditional DSE.

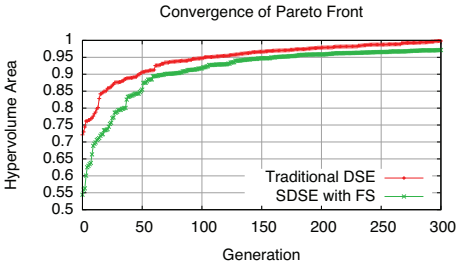


Figure 8: Generation-based comparison

4.2.2 The quality of the scenario-based DSE

In the previous experiment, we have shown that it is necessary to speed up the early DSE of a multi-application workload by using scenario-based DSE. As a next step, we show that the deterministic approach of selecting a subset of scenarios improves our result of the early DSE. To this end, we compare our approach to an alternative scenario-based DSE approach which has been proposed in [vSP10a]. In this alternative approach, a co-evolutionary GA is used to perform scenario-based DSE. The experiment is performed on a sixteen core machine and thus we have used 12 threads for the design explorer and 4 threads for the subset selector. For both the FS and the GA approach ([vSP10a]), the experiment is repeated five times.

Figure 9 shows the results of the experiments. The hypervolume is averaged over all the runs and the standard error over the different experiments is shown with an error bar. During the first 50 generations, the FS and the GA approaches are comparable. In this initial phase, the differences between the fitness of the different platform instances is still large enough to tolerate inaccurate predictions of a subset with a moderate misclassification rate.

As of approximately 50 generations, our FS method starts to outperform the GA approach, and the gap between both approaches steadily increases. After a startup period, the FS approach is able to find a better scenario subset with a higher accuracy. The higher accuracy results in a good and valid distinction between mappings which have fitness values that

are relatively close. As a consequence, the FS approach can find a higher number of better candidate designs. The GA approach is less capable of making a distinction between the highly ranked mappings and its hypervolume starts to flatten earlier.

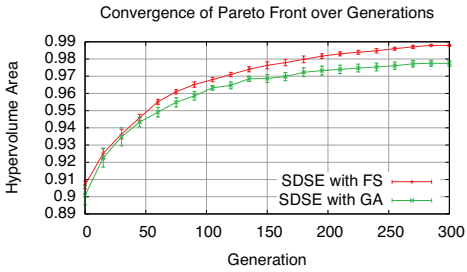


Figure 9: The effect of the deterministic subset selection on the quality of the DSE. The feature selection (FS) approach presented in this paper is compared with an approach where a GA is used to find the scenario subset.

Moreover, it can be seen that the FS approach is more deterministic than the GA approach. The standard error of the different experiments is significantly smaller after 250 generations have passed. On the final result, the standard error of the mean is 3.6 times as small as compared to the GA approach. This is because the GA approach is dependent on the stochastic nature of the GA to find good scenario subsets. A stochastic approach will eventually find a good subset, but the larger the subset is, the tougher it becomes.

5 Related work

There is a large body of research dedicated to early DSE of MPSoC based embedded systems [Gri04]. However, the majority of this work still evaluates the embedded systems with a single (fixed) application workload. Recently, research has begun to apply workload scenarios [G+09] for making the design phase scenario aware [PSZ08, P+06, S+10a]. In [vSP10a] a scenario-based DSE technique using a coevolutionary GA is described. The scenario-based DSE approach proposed in this paper has been inspired by the work of [vSP10a], but as shown in Section 4.2.2, the scenario subset selection is significantly improved and thereby the quality of the entire DSE process.

In our scenario-based DSE, fitness prediction[JB05] is used to deal with the large number of application scenarios. For this purpose feature subset selection[S+10b] is used. An example of a case where feature selection is used for ranking can be found in [L+04]. This work uses a hybrid method, which is a combination of a so called kNN classifier and a NSGA-II based GA, to estimate the Pareto front of a hydrological model. These parts are running in lockstep and not independent as in our approach.

6 Conclusions

In this paper, we have presented a framework for scenario-based DSE with feature selection. Scenarios enable the modeling of the dynamism in multi-application workloads. Our framework performs simultaneous co-exploration using a design explorer and a subset selector. The design explorer uses the current scenario subset to predict the fitness of the MPSoC design instances. The subset selector, on the other hand, uses a sample of design

instances to train a scenario subset using feature selection. Our experiments show that, based on the size of the subset of scenarios, we are able to make a tradeoff between quality and the evaluation time of the early DSE.

References

[C+07] C. Coello et al. *Evolutionary Algorithms for Solving Multi-Objective Problems Second Edition*. Springer US, 2 edition, 2007.

[D+02] K. Deb et al. A Fast and Elitist Multiobjective Genetic Algorithm: NSGA-II. *IEEE Trans. on Evolutionary Computation*, 6(2):182–197, April 2002.

[G+09] S. V. Gheorghita et al. System-scenario-based design of dynamic embedded systems. *ACM Transactions on Design Automation of Electronic Systems*, 14(1):1–45, 2009.

[Gri04] M. Gries. Methods for evaluating and covering the design space during early design development. *Integration, the VLSI Journal*, 38(2):131–183, 2004.

[JB05] Y. Jin and J. Branke. Evolutionary Optimization in Uncertain Environments-a Survey. *IEEE Transactions on evolutionary computation*, 9(3):303–317, 2005.

[L+04] Y. Liu et al. A Hybrid Optimization Method of Multi-objective Genetic Algorithm (MOGA) and K-Nearest Neighbor (KNN) Classifier for Hydrological Model Calibration. In *Intelligent Data Engineering and Automated Learning*, volume 3177, pages 546–551. 2004.

[Ors09] H. Orsila. `http://zakalwe.fi/~shd/foss/kpn-generator/`, 2009.

[P+06] J. M. Paul et al. Scenario-oriented design for single-chip heterogeneous multiprocessors. *IEEE Trans. on VLSI Syst.*, 14(8):868–880, August 2006.

[PEP06] A. D. Pimentel, C. Erbas, and S. Polstra. A Systematic Approach to Exploring Embedded System Architectures at Multiple Abstraction Levels. *IEEE Trans. on Computers*, 55(2):99–112, 2006.

[PSZ08] G. Palermo, C. Silvano, and V. Zaccaria. Robust Optimization of SoC Architectures: A Multi-Scenario Approach. In *Proceedings of ESTIMedia 2008*, October 2008.

[S+10a] A. Schranzhofer et al. Dynamic and adaptive allocation of applications on MPSoC platforms. In *Proceedings of the ASP-DAC*, pages 885–890, 2010.

[S+10b] P. Somol et al. *Pattern Recognition Recent Advances*, chapter Efficient Feature Subset Selection and Subset Size Optimization, pages 75–97. In-Tech, February 2010.

[SP00] P. Somol and P. Pudil. Oscillating Search Algorithms for Feature Selection. *International Conference on Pattern Recognition*, 2:406–409, 2000.

[vSP10a] P. van Stralen and A. D Pimentel. Scenario-Based Design Space Exploration of MPSoCs. In *Proceedings of IEEE Int. Conference on Computer Design (ICCD '10)*, October 2010.

[vSP10b] P. van Stralen and A. D. Pimentel. A Trace-based Scenario Database for High-level Simulation of Multimedia MP-SoCs. In *Proc. of the Int. Conference on Embedded Computer Systems: Architectures, MOdeling and Simulation (SAMOS '10)*, July 2010.

[ZT98] E. Zitzler and L. Thiele. Multiobjective optimization using evolutionary algorithms — A comparative case study. In *Parallel Problem Solving from Nature*, volume 1498, pages 292–301. 1998.

Sub-word Handling in Data-parallel Mapping

Georgia Psychou[1], Robert Fasthuber[2], Jos Hulzink[3], Jos Huisken[3], Francky Catthoor[2]

[1] Comp. Eng. and Inform. Dep., Univ. of Patras, Greece
(currently at Electr. Eng. and Comp. Syst. Dep., RWTH Aachen Univ. Germany);
psychou@eecs.rwth-aachen.de
[2] IMEC, Leuven, Belgium; {fasthuber,catthoor}@imec.be
[3] Holst Center/IMEC, Eindhoven, The Netherlands; {hulzink,huisken}@imec-nl.nl

Abstract: Data-parallel processing is a widely applicable technique, which can be implemented on different processor styles, with varying capabilities. Here we address single or multi-core data-parallel instruction-set processors. Often, handling and reorganisation of the parallel data may be needed because of diverse needs during the execution of the application code. Signal word-length considerations are crucial to incorporate because they influence the outcome very strongly. This paper focuses on the broader solution space of selecting sub-word lengths (at design time) including especially hybrids, so that mapping on these data parallel single/multi-core processors is more energy-efficient. Our goal is to introduce systematic exploration techniques so that part of the designers effort is removed. The methodology is evaluated on a representative application driver for a number of data-path variants and the most promising trade-off points are indicated. The range of throughput-energy ratios among the different mapping implementations is spanning a factor of 2.2.

1 Introduction

Today designers are using a very simplified methodology to exploit the available signal quantisation freedom. That is in general the case for mapping applications to instruction-set processors, which is our target in this paper. Exceptions are present for cost-sensitive custom ASIC or FPGA oriented designs, but those are not our focus here. Usually, the width of the available data-path (e.g. 32 bit) is used for representing all the signals in a fixed-point notation. Alternatively, all data are just left in floating-point notation when the processor platform supports it. That is the easiest and the most effective approach from a design-time reduction point of view. Many processor vendors explicitly use that argument to motivate the support of expensive floating-point arithmetic in their processors, even in the embedded DSP domain where processor cost overhead is seen as undesirable (see e.g. TI C6x DSPs). In the best case, designers use a quantisation technique that reduces the word-length of all signals in a uniform way to a few power-of-2 values (e.g. [Ha04, Wi11]). This allows to better exploit SIMD or sub-word parallel operation. Usually, that leads to a division of the signals in a few categories: 8, 16 and 32 bit data.

However, we believe that recently introduced and still developing techniques for advanced quantisation exploration [Pa10, No10a] allow to change this state-of-the-art paradigm. With these new emerging techniques, a methodology is enabled where the minimal word-length for all signals is determined individually based on quality of service (QoS) require-

ments. Examples are the bit-error rate (BER) in a wireless system or the signal-to-noise ratio (SNR) for a multi-media system. As a result, a wide range of signals with a very non-uniform distribution is produced, e.g. ranging from 6 to 24 bit [No10a]. So, we can exploit a much wider variation of SIMD groupings and this motivates in its turn to go beyond state-of-the-art hardware-based SIMD concepts where only a few power-of-2 word-lengths are supported. In an ad hoc way, custom ASIC designers are sometimes using this fine-grain quantisation information already, but the resulting range of word-lengths is then used to hardwire different operators and not used for a time-multiplexed SIMD processor. We are trying to explore such solutions with our methodology. Moreover, the methodology is not only applicable on single processing elements (PEs) but also on multi-core processors with multiple homogeneous PEs which are organised in a SIMD fashion.

Our first contribution is the systematic exploration of selecting data word-lengths on different SIMD platform options, with a given distribution over the signals of minimal word-length requirements, including especially the attractive hybrid options. Secondly, we also show how we can obtain a broad range of near-optimal mapping results in terms of performance, energy and area trade-offs for realistic embedded applications on an single or multi-core SIMD processor.

The structure of the paper is organized as follows: Section 2 presents related work, while Sect. 3 exhibits the proposed systematic approach for data organisation within the data-parallel processing context. A case study in Sect. 4 is used to illustrate the potential use of the methodology and to show the benefits through some high-level comparisons with alternative techniques. Section 5 concludes the paper.

2 Related work

In the literature, a number of mapping solutions on data-parallel platforms have been proposed, especially for vector or conventional hardSIMD architectures. Regarding SIMD implemented in software, a number of instantiations are discussed in [BD06, La07, No10b] but without any systematic classification or selection process.

In order to exploit the data-level parallelism present in the application, often the data layout (the order in which the data is stored in the memory) has to be modified and the data sub-words that will be operated in parallel have to be (re-)packed together into words. This process, together with the restructuring of the application code to perform the data parallel execution, is called vectorization or SIMDization. Vector compiler-focused work has been published in [LA00, Te05].Various SIMDization and transformations techniques have been proposed in the general or embedded compiler literature [KL00, XOK07]. They are focused on the basic parallelisation, potentially together with some cost functions to reduce the overhead of dealing with various sub-word lengths. But they do not try to describe or explore the broad search space that is available with non-conventional techniques or hybrids for effectively handling sub-words with different word-lengths.

Many researchers have addressed SIMD mapping on homogeneous SIMD architectures in the past (see e.g. [CF04, Qi09]). They have focused however only on the pure hardware-supported SIMD styles, and very seldom even on the sub-word parallel processing. Exceptions to this are the papers [Fr05, CVE09] which do address sub-word processing but again without covering any design space exploration of hybrids.

3 Systematic exploration of sub-word handling hybrids

3.1 Target platforms description

Typically, the elements, that can be found in sub-word parallel SIMD architectures, are the data memory and the register file, arithmetic operators such as shift-add units, and a sub-word rearrangement unit such as a shuffler (if repacking or other operations are necessary). Traditionally, within SIMD, the arithmetic hardware is capable of handling the sign and carry (or other bits) propagation for each of the operands during the calculations, so that they do not pollute each other (called hardSIMD further on). More recently, a software-controlled technique has been introduced, named softSIMD [BD06]. SoftSIMD does not include dedicated circuitry in the arithmetic units to support the data-parallel operations. When operations move operands across their limits, techniques like resizing of the sub-words (repacking) and masking through placing guard bits [BD06] then have to be employed. The difference between hardware-implemented SIMD and the software version, lies in the shuffler operator that now handles the bits on the MSB or LSB side that can potentially pollute the adjacent sub-words. This sub-word parallel operation with the corresponding support can be realized in a single core processor but also in a homogeneous SIMD multi-core processor. In that case, the overall data parallel solution is constructed of a hybrid between sub-word processing inside the cores and vector SIMD across the cores. For instance if we have 8 cores of 64 bit each and we have support for sub-words of 8, 16 and 32 bit, then the mapping can contain 64 parallel operations on 8 bit, 32 operations on 16 bit and 16 operations on 32 bit.

3.2 Systematic exploration of sub-word lengths

For this analysis, a set of proposed word-lengths for each variable in the application code is taken as input. The set comes from emerging systematic and hence automatable quantization techniques such as [Me10]. It includes information about hard and soft constraints regarding the word-lengths. This allows the mapping stage to effectively exploit the word-lengths that are reduced compared to conventional worst-case techniques.

The methodology to select the appropriate sizes for the signals of an application is illustrated through a compact classification scheme (Fig. 1). This scheme can be used by designers to find optimized options for the application word-lengths in a systematic way, which makes sure that no important option is missed. Some of these solutions are well-known but a number of them are novel (to the best of our knowledge), since the techniques for exploring reduced word-length are not so widely applied yet. More importantly, this top-down split-based tree organization allows a number of very promising hybrid solutions to pop up which are not so easy to find simply by trying out different combinations. Hybrids come up by combining options. One such more obvious illustration is that software SIMD can be used to improve the capabilities of a hardware SIMD platform. An example mentioned in [BD06] is that softSIMD could be used in DSPs when, for instance, hardware support for 16-bit SIMD operations is provided, but not for operands with a lower word-length. In that case, 16-bit hardSIMD instructions are combined with lower word-length softSIMD instructions inside the 16-bit segment. But the searching space is broad and this is what we are trying to explore here.

Initially, the quantization information of all variables in the application code is collected. Then, an important distinction has to be made between selecting signal sizes for applications that allow accuracy loss and those that do not (split **a** in Fig. 1). The information about which direction we follow in the diagram, comes from the application designers, who code the programs.

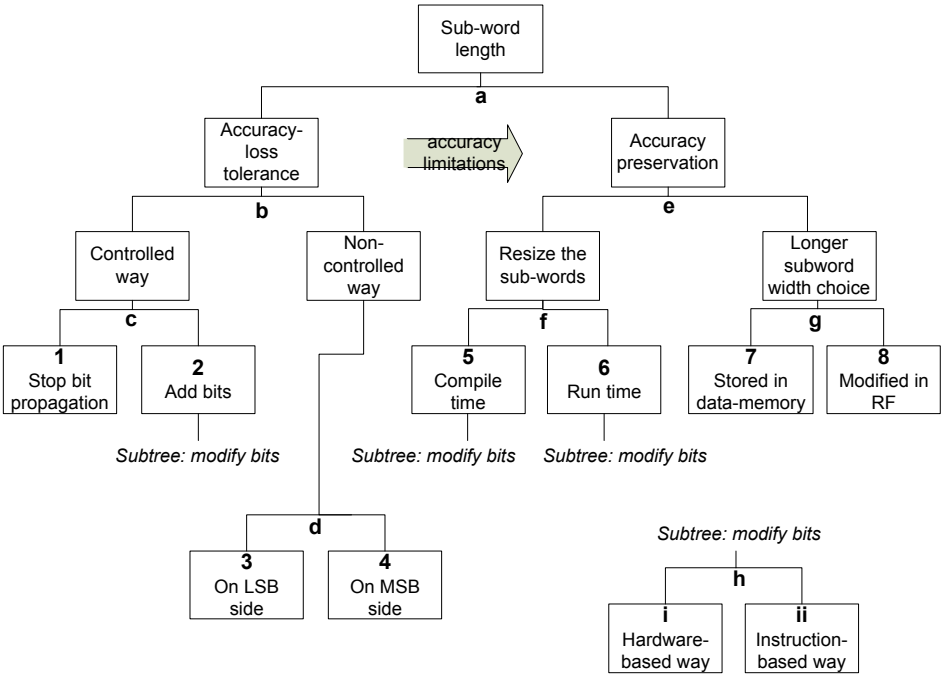


Figure 1: Sub-word length selection

3.2.1 Accuracy loss tolerance.

When the type of application allows that data have some accuracy loss, two potential cases can be found (split **b**). In the first case (left-hand side of split **b**), at compile time it can be known exactly how many bits can be discarded from each data item, namely accuracy loss takes place in a completely controlled way. In the second case (right-hand side of split **b**), the loss of accuracy can not be fully determined until run-time. The bit loss is bounded by a pre-analysed upper limit, to ensure correct functionality. This is the non-controlled subcategory. Whether we are in the controlled or non-controlled branch depends on whether we have sufficient knowledge of how the application evolves during execution time. An example application in the controlled accuracy loss subcategory is a video encoder. Since the code and typical (natural) data streams of the video encoder and the compression standards are predefined, it can be analysed at design time how much accuracy loss will be present. This happens by profiling representative video input sequences and extrapolating from these information regarding the accuracy loss. On the other hand, the subcategory of uncontrolled accuracy loss solutions can be applied in cases like data transmission applica-

tions that are best effort-oriented, such as Skype-like protocols. Since it can not be known exactly at design time how many users will appear, namely the load changes at run-time, it is impossible to accurately preanalyse the data.

An example can be used to illustrate more effectively these cases. We use simple operations as additions and right shifts, which are representative cases of handling bits both on the MSB and LSB side. In this example, the operands, resulting from the quantization analysis, are 5-bit long and the data-path is 30 bits wide. Each of the operands takes part in an addition and/or a right shift. Given a 30-bit data-path, we can operate 6 sub-words of 5 bits in parallel.

In the controlled approach (left-hand side of split **b**), the accuracy loss can be guaranteed either by stopping bit propagation (carry or LSBs) by means of hardware (leaf **1**) or by adding bits (leaf **2**). This guarantee refers to making certain that the operands do not cross their boundaries and pollute their neighbouring ones. Thus, the results remain valid. Leaf **1** applies both to hardSIMD and softSIMD. For softSIMD, the bit cut off can be similar to the hardSIMD case, but with a different internal hardware structure to create less overhead especially when many (non power of 2) sub-word sizes exist. Adding bits can be accomplished by extending the word-lengths and placing guard bits or modifying bits in the existing space (applies for the softSIMD case). In the example shown in Fig. 2a, the accuracy loss is being controlled leading to only 1-bit loss (instead of potentially 2 due to the right shift). This happens by extending the 5-bit operands to 6 bits and zeroing out their LSB before they are shifted.

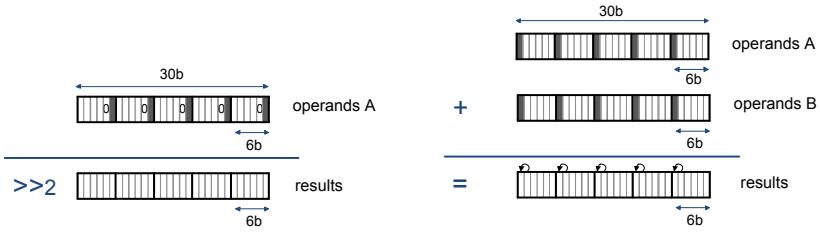


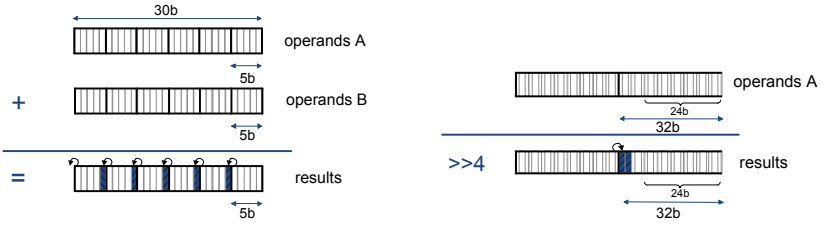
Figure 2: Extended sub-words for controlled accuracy loss and preservation

The placing of the guard bits can either take place in a hardware-based fashion (leaf **2i**), for instance by shuffling of bits, or in an instruction-based way (leaf **2ii**). In the last case, the data-word to be masked is entered as one input of the masking unit. Another word, with the desired composition of guard bits, needs to have been stored upfront in the data memory. The (re)placement occurs through a logic operation.

When accuracy loss occurs in the non-controlled way (right-hand side of split **b**), no special care is taken for the sub-words of the output (after the addition or the shift). Only an upper limit exists for the bit loss (see beginning of Subsect. 3.2.1). In the example, the 5-bit values of the output may contain inaccurate data. Here, the case that there is pollution of data of the neighbouring sub-words on the LSB side (leaf **3**) or on the MSB side (leaf **4**) has to be discussed separately. Regarding leaf **3**, inaccurate data appears (as shown in Fig. 3a) when an overflow occurs during an addition. If no precaution is taken,

the overflow bit moves out of the sub-word and generates an inaccurate result for both this and its neighbouring sub-word (the MSB enters the left-hand sub-word).

Also right-shift operations can lead to accuracy loss in an uncontrolled way (the shifted bits -LSB- enter the right-hand sub-word). These instances are relevant to leaf **4**. Since the MSBs are influenced, it is possible that the results come out totally wrong. However, cases exist that this does not happen, as the one illustrated in Fig. 3b. Here, the operands are two 32-bit sub-words. The data values needed to be represented are only 24 bits. If these 24 bits are chosen to be positioned on the right-hand side (LSB side) of the sub-words, then the 4 bits that are shifted in from the left-hand side do not influence the results on the MSB side. In this way, the results remain valid and no overhead operations need to take place.



(a) during an addition (b) during a right shift

Figure 3: Accuracy loss that occurs in a non-controlled way

3.2.2 Accuracy preservation.

The second category (right-hand side of split **a**) considers the case where a hard guarantee exists for the accuracy preservation. This is true in mission critial applications where not meeting these hard requirements on the results will have an unacceptable impact.

The accuracy preservation can happen either by enlarging the sub-words to prevent bit loss during the operations (split **e** left-hand side) or by upfront (during the word-length decision phase) selecting a bigger word-length before the beginning of the execution (right-hand side of split **e**). In Fig. 2b although the operands are 5-bit long, 6 bits are reserved for each one of them. The 6th bit (MSB) provides space for the overflow bit.

In the case of resizing (left-hand side of split **e**), sub-words are extended to prepare for the operations that follow. The decision on the resizing can be made either at compile (leaf **5**) or at run-time (leaf **6**). The run-time mechanism could be realised with preinstalled hardware that supports precoded senario-based mapping options. The latter are then selected during run-time. For the addition case of our example, this means 6-bit sub-words are selected before the beginning of the operation. For the right-shift operation, (5+n)-bit sub-words are selected before the beginning of the shift by n. In this case, less sub-words are processed in parallel, i.e. only 5 instead of 6 in the case of the addition. Still, every sub-word of the output has a correct result and no sub-word is affected by its neighbouring one. Thus, at the cost of extra resizing operations and a reduced number of parallel sub-words, a hard guarantee can be provided for the accuracy. To achieve an efficient resizing for the softSIMD case, guard bits are used. The placing of the guard bits can take place in a similar way as for leaf **2** (subtree **h**).

At the right-hand side of split **e**, during the initial selection of the word-lengths, we choose a bigger word-length in order to provide enough space for the extra bits i.e. the overflow bit in the case of the addition. In our example instead of the minimal length of 5 bits, we select 6-bit word-lengths. In that case, the correctness of the results is maintained, and extra resizing is avoided, but we operate only 5 sub-words in parallel. The bigger word-lengths can either have been stored in the data-memory in the intended extended form (leaf **7**) or the extension can take place during their stay in the register file (leaf **8**). Within softSIMD, the extra space provided by the bigger word-length selection is filled with guard bits on the one or both sides of the sub-words depending on the next operations.

3.2.3 Hybrid solutions.

Most importantly, many hybrid solutions can be derived in a fully systematic way by combining characteristics of any subset of leaves in Fig. 1. All these leaves are namely potentially compatible with each other. Here, we will describe only a few to illustrate this property. These combinations lead to new, non-obvious choices.

One such case is that within the controlled accuracy loss approach (left-hand side of split **b**), the stopping of bit propagation (leaf **1**) can be combined with the addition of bits at specific places (leaf **2**). In that way, space can be provided for the MSBs by cutting off LSBs. The guard bits for this case are placed at the MSB side of the sub-words. In Fig. 4 such an example is shown. The LSB of the 6-bit operands is stopped by means of hardware during the right shift by 1. In combination to that, a guard bit is placed on the MSB of the sub-words to potentially prepare them for an addition. If we would choose to stop bit propagation on the MSB side as well (during the addition), then the neighbouring sub-words would still be protected but the accuracy of the current sub-words would be altered (as they cannot expand to the extra bit any longer). The other option to preserve accuracy would be to resize the sub-words to a bigger word-length before the execution of the operations. But this potentially means less parallelism exploitation. This hybrid approach provides a way to avoid the use of a bigger word-length. A second hybrid

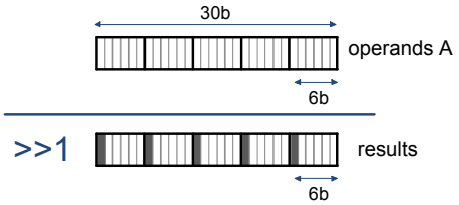


Figure 4: Hybrid approach for controlled accuracy loss

illustration considers the combination of the resizing option (left-hand side of split **e**) with controlled loss of accuracy (left-hand side of split **b**). This could be considered when the loss of accuracy would be too big if no resizing takes place but a flawless accuracy preservation is also not preferable (e.g. because it leads to lower parallelism exploitation). Many other promising hybrids exist in the overall search space.

It becomes obvious that this diagram can be effective for the selection of the platform characteristics, since certain platform options can lead to more optimized mapping results depending on the application domain and the overall system requirements. However, even

when the platform is already defined this diagram remains partly relevant. For example, instruction-based masking can be performed as long as the basic logical operations are supported on the platform. This is common for conventional instruction-set processors.

The way to select and evaluate the different possibilities is dependent of course on the application and platform characteristics. But also trade-offs are present. We briefly discuss here the main issues that play a role in this decision making. A more systematic evaluation is part of future work. The total number of groups of lengths should be small so that the overhead involved with resizing the sub-words is minimized (e.g. in the shuffler). If, for example, in an application among other word-lengths needed, 13- and 16-bit data occur at similar frequencies, it would make sense to use the 16-bit value for both cases. In that way, the support of an extra word-length is avoided. In addition, a resizing operation should be avoided when the accuracy loss is acceptable compared to the gain due to more difficult masking and repacking. For example, if an application allows that some LSBs are lost then it is better to avoid repacking to preserve accuracy and make use of this tolerance of the application so that energy consumption is further minimized. A specific example illustrating trade-offs regarding this selection decision is discussed in Subsect. 4.2.

4 Evaluation of the proposed methodology

4.1 Driver algorithm

An image filter is used as driver for the evaluation of the proposed sub-word handling techniques. The illustration regards the mapping of a 1-dimensional Gauss loop filter applied to an input frame. Different architectures are considered for mapping the Gauss loop. However, due to space limitations, only one mapping implementation will be discussed more thoroughly now and the others will be summarized and compared in Sect. 4.3. The

program 1 Optimized Gauss loop (1D)

for $x = 1 \rightarrow N - 1$ **do**
  for $y = 2 \rightarrow M - 1$ **do**
    Op1: $MulRes0 = imsub[x-1][y] + imsub[x+1][y]$;
    Op2: $MulRes7 = MulRes0 * Gauss[0][1]$;
    Op3: $MulRes4 = imsub[x][y] * Gauss[0][2]$;
    Op4: $imgauss\_x[x][y] = MulRes4 + MulRes7$;
  end for
end for

optimized code of the 1D filter is depicted in program 1. The operations involved are two additions (Op1, Op4) and two constant multiplications (Op2, Op3). The filter coefficients are shown in Table 1. The multiplications can be performed either by using a multiplier unit or by a shift-add unit (by applying strength reduction). For the current illustration, when the multiplications are performed in a shift-add way, the right-shift approach is chosen and for the constants the Canonical Signed Digit (CSD) [Ba07] encoding is considered (see Table 1). The first multiplication is performed in three shift-add/subtract steps by 4, by 3 and by 2. Similarly, for the second multiplication right shifts by 3, by 2 and by 5 and the corresponding additions/subtractions need to take place.

	Gauss[0][1]	Gauss[0][2]
Decimal	655	983
Binary	1010001111	1111010111
CSD coding	101001000-	10000-0-00-

Table 1: Gauss coefficients

Signal	Length (bits)	Signal	Length (bits)
imsub	7	imgauss_x	12
MulRes7	10, 11, 13	MulRes4	9, 10, 13

Table 2: Gauss loop signals

4.2 Application of the methodology

We will illustrate the mapping of the code using the softSIMD related leaves in Fig. 1. SoftSIMD makes sense to use while performing simpler operations that do not lead to a potential explosion of the operands' word-length [BD06]. Thus, for the multiplications, the shift-add scheme is employed. We consider a 48-bit data-path that consists of a shift-add-subtract unit (SAS) for the arithmetic operations, a shuffler unit for the repacking operations and intermediate registers (softSIMD data-path).

For the mapping, minimal quantization-based word-lengths (for the input signals, the intermediate results of the constant multplications and the final results), as shown in Table 2, are our starting point. Then, in correspondence to the Fig. 1, the word-length choices of this Gauss filter code occur as follows:

1. The first operation (*Op1*) is an addition of 7-bit values (*imsub*). The result needs to be stored in 8-bit values. To avoid a repacking operation from 7 to 8 bits, the signals are stored as 8-bit in the data memory (leaf **7**). That means 6 sub-words are in parallel in the 48-bit data-path. The result of this operation (MulRes0) is used in the second operation which is a multiplication with a Gauss coefficient. The second multiplication starts with a right shift by 4. Since the operands need to be shifted by 4, they need to be extended to 12-bit values, before they enter the data-path. So that accuracy is preserved during the right shift, 1 LSB is cut-off from the first addition result (leaf **1**) and guard bits are added (leaf **5i** or **5ii**) so that the sub-words are 12-bit long to prepare for the next multiplication. Combining these leaves is possible as explained in Subsect. 3.2.3.

2. During the second operation (*Op2*), after the first shift-add operation by 4 (explained before) and prior to the second shift operation by 3, a repacking operation of the variable to 16 bits needs to take place so that accuracy is preserved (leaf **5i** or **5ii**). Because two guard bits are required at the left-hand side of the repacked multiplicand (one for the additions within the multiplication and one extra for the addition of the fourth operation of the Gauss loop), the result is shifted out of the sub-words during the last shift by 2. In this case, care must be taken so that the extra bits are cut off and do not enter the adjacent operand (leaf **1**). This choice is a hybrid combination of the resizing option (leaf **5i** or **5ii**) and the stopping of bit propagation (leaf **1**), that allows us to exploit the given word-length in an efficient way while we are also obeying the accuracy limitations. If the guard bits are not placed, potential overflows coming from the additions can corrupt the results. On the other hand, if the 2 LSBs are not cut-off, right-hand sub-words may be polluted. Another alternative would be that an extra word-length is then supported, for example the 18-bit or 24-bit word-length. But this would increase the hardware overhead (for example in the shuffler because an additional repacking operation would have to be supported) and would limit the parallelization potential. With the 18-bit or 24-bit option only two sub-words

could operate in parallel in the 48-bit data-path. This allows this hybrid combination to appear as an optimal option.

3. For the third operation (*Op3*) similar choices with the previous operation are made. The imsub data are stored as 8-bit in the data memory (leaf **7**) and are resized to 12-bit by the register file (leaf **8**) before they enter the data-path. As before, the sub-words operate as 12-bit during the first cycle and as 16-bit during the two subsequent cycles.

4. The fourth operation (*Op4*) is an addition of the multiplication results (*Op2*, *Op3*) stored into a 12-bit value (according to quantization results). In this case, 4 sub-words are working in parallel.

More detailed information on the exact mapping is available in [Ps10].

4.3 Comparison with mapping on alternative data-paths

Alternative data-paths. Besides the softSIMD discussed above, alternative data-paths are considered. Like softSIMD, hardSIMD data-paths also include a shift-add-subtract (SAS) unit and a shuffler. The differences with softSIMD have been discussed in Sect. 3. The cases considered are one more unconventional hardSIMD option (HardSIMD SAS, 3-wl) that supports all three sub-word sizes (8-, 12-, 16-bit), one more conventional (HardSIMD SAS, 2-wl) that supports only power of 2 sub-word sizes (in this case 8-, 16-bit). Moreover, the SIMD case where only one sub-word size (24-bit) is considered (SIMD SAS, 1-wl). In the last case, no shuffler is needed and the SAS unit does not need to handle the carry and other bits (the operands remain within their boundaries).

As further reference, we also consider hardwired multiplier data-paths. In this case, a multiplier is used in the position of the SAS unit. Operands' width during the multiplications is calculated as follows. The input operands (*imsub[x...]*) are 7-bit and the Gauss coefficients are 10-bit long. That means that the width needed for each multiplication result is at least 17 bits, and when rounded to the next power of 2, this becomes 32 bits. For the multiplier approaches discussed in this paper, a more optimized choice of 24 bits is used as an optimistic comparison reference. Moreover, for the additions existing in the application an adder is needed and when more than one word-length is supported, a shuffler unit is present. In this category, two cases are considered, one (HardSIMD Mult, 3-wl) where three sub-word sizes (8-bit for the initial addition, 24-bit for the multiplications and 12-bit for the final addition) are supported and one (SIMD Mult, 1-wl) where only one size is supported. In the last case, six groups of two 24-bit operands are operating in parallel and operands remain within their limits during the operations.

Relative area, energy consumption and throughput. The different data-path options with the required functionalities have been implemented in TSMC 40 nm std. cell technology, using Cadence RTL Compiler and SoC Encounter for synthesis and place and route. To obtain the area, energy and throughput numbers for the components, a similar estimation flow as in [Fa09] has been applied. Each data-path is synthesized for the maximum clock frequency; no pipelining in the arithmetic data-path is applied. The algorithm is manually scheduled and optimized for each data-path variant. Software pipelining is applied in cases where more than one operator is present. The number of required clock cycles (to complete one Gauss iteration; 12 pixels) together with the critical path is then

used to determine the relative throughput. For the energy estimation, the actual activation count for each data-path unit is considered. Table 3 summarizes the results.

Data-path option	Accesses of operators	Area	Energy	Through-put	Through. /energy
SoftSIMD	27*SAS + 19*Shuf	1.63	0.80	1.69	2.12
HardSIMD SAS, 3-wl	27*SAS + 11*Shuf	1.48	0.72	1.27	1.76
HardSIMD SAS, 2-wl	30*SAS + 4*Shuf	1.28	0.72	1.33	1.85
SIMD SAS, 1-wl	48*SAS	1.00	1.00	1.00	1.00
HardSIMD Mult, 3-wl	5*Adder + 12*Mult + 9*Shuf	7.84	2.05	2.00	0.96
SIMD Mult, 1-wl	12*Adder + 12*Mult	6.63	2.00	2.29	1.15

Table 3: Estimation of the total relative area, energy consumption, throughput and throughput-energy quotient for one Gauss iteration (data-path = 48 bits).

As it can be seen in Table 3, SIMD SAS 1-wl requires the lowest area, which is reasonable since no bit handling among sub-words needs to take place. However, the delay significantly increases in this case. The softSIMD and hardSIMD SAS (3-wl) data-paths, that exploit maximally the different solutions presented in Sect. 3, are among the near-optimal solutions in the table. SoftSIMD achieves a higher throughput, compared to the other SAS implementations (because of the small critical path and the dense scheduling). More importantly, it achieves the highest throughput-energy ratio. HardSIMD SAS data-paths exhibit energy-efficiency because of the simpler shuffler unit. HardSIMD SAS (2-wl) has a more optimal throughput-energy combination than the hardSIMD (3-wl) because of the slightly smaller critical path. The multiplier data-paths have a big area and energy overhead but achieve the highest throughput. The throughput-energy ratio is a viable metric because at the circuit level, it is the most relevant trade-off. A faster circuit indeed requires more energy to perform a computation than a slower one, due to buffer sizing. For instance, the softSIMD circuit could be made slower and therefore it will also consume less energy. Note that these results scale as well to a multi-core processor composed of a homogeneous set of PEs with sub-word parallel support, as indicated already in subsection 3.1. In that case the mapping above is simply copied for each of the PEs due to the homogeneous overall mapping principle.

The table demonstrates a wide range of options in the total space, which lead to a wide range of quantitative results. According to the objectives, the options are worth exploring. For instance, when high timing constraints are present using a multiplier data-path is a better option. When design time is not a hard limitation, softSIMD as well as hardSIMD which supports non power-of-2 sub-word sizes are worth exploiting, since they exhibit the best energy-efficiency.

5 Conclusion

In this paper, the broad solution space of organizing parallel data with minimal data word-length requirements in domain-specific processors during mapping (at design time) is explored in a systematic way. These processors can consist of single or multi-core architectures organized in a homogeneous SIMD fashion. A methodical way reveals a broad number of options and can potentially save design time. The application of the proposed

sub-word handling analysis on a driver algorithm has substantiated a wide mapping solution space. Our goal is to facilitate an intermediate step of the mapping process, after having selected the minimal word-length requirements and before the compiler tool decides and applies the final mapping. This precompiler methodology can provide pragma or intrinsic-based guidance to support the traditional compilers.

References

[Ba07] E.Backenius, Reduction of Substrate Noise in Mixed-Signal Circuits, PhD thesis, Linkoping University, 2007.

[BD06] Emulate SIMD in software, `http://www.bdti.com/InsideDSP/2006/12/06/Bdti`, 2006.

[CF04] G.Cichon, G.Fettweis, Mouse: a shortcut from Matlab source to SIMD DSP assembly code, *SAMOS'IV*, Greece, July 2004.

[CVE09] J.Corbal, M.Valero, R.Espasa, Exploiting a new level of DLP in multimedia applications, *MICRO conf*, New York, Dec. 2009.

[Fa09] R.Fasthuber et al, Novel energy-efficient scalable soft-output SSFE MIMO detector architectures, *Proc. Intnl. Conf. (IC-SAMOS)*, July 2009.

[Fr05] F.Franchetti et al, Efficient utilization of SIMD extensions, *Proc. Of the IEEE*, Vol.93, 2005

[Ha04] K.Han et al, Data wordlength reduction for low-power signal processing software, *Proc. IEEE Wsh. on Signal Proc. Syst. (SIPS)*, Austin TX, IEEE Press, pp.343-348, Oct. 2004.

[KL00] A. Krall, S.Lelait, Compilation techniques for multimedia processors, *Int. Journal on Parallel Programing, 28(4)*, 2000.

[LA00] S. Larsen, S.Amarasinghe, Exploiting superword level parallelism with multimedia instruction sets, *ACM SIGPLAN Notices, 35(5):145156*, 2000.

[La07] A.Lambrechts et al, Enabling word-width aware energy optimizations for embedded processors, *Proc. ODES*, San Jose CA, pp.66-75, March 2007.

[Me10] D.Menard et al, Quantization mode opportunities in fixed-point system design, *Proc. 18th Eur. Sign. Proc. Conf.(EUSIPCO)*, Denmark, pp.-, Aug. 2010.

[No10a] D.Novo et al, Exploiting Finite Precision Information to Guide Data Flow Mapping, *Proc. 47th ACM/IEEE Design Automation Conf. (DAC)*, Anaheim CA, pp.248-253, June 2010.

[No10b] D.Novo et al, Ultra Low Energy Domain Specific Instruction-set Processor for On-line Surveillance, *Proc. SASP co-located with DAC*, Anaheim CA, pp.30-35, June 2010.

[Pa10] K.Parashar et al, Fast performance evaluation of fixed-point systems with un-smooth operators, *Proc. IEEE Intnl. Conf. Comp. Aided Des. (ICCAD)*, San Jose, pp.9-16, Nov. 2010.

[Ps10] G.Psychou, Optimized SIMD scheduling and architecture implementation for ultra-low energy bioimaging processor, M.Sc. thesis, Univ. of Patras and IMEC NL, 2010.

[Qi09] M.Qiu et al, Energy-aware loop scheduling and assignment for multi-core, multi-functional unit architecture, *J.Signal Processing Systems*, Vol.57, pp.363-379, 2009.

[Te05] C.Tenllado et al, Improving superword level parallelism support in modern compilers, *Wsh. Codes-ISSS*, NY, pp.303-308, Oct. 2005.

[Wi11] Wikipedia, `http://en.wikipedia.org/wiki/Fixed_point_arithmetic`, read Sep.2011.

[XOK07] L.Xue, O.Ozturk, M.Kandemir, A memory-conscious code parallelization scheme, *Proc. 44th Conf. Design automation*, pages 230–233, NY, USA, 2007. ACM Press.

Concurrent Phase Classification for Accelerating MPSoC Simulation

Melhem Tawk, Khaled Z. Ibrahim*, Smail Niar

University of Valenciennes, 59313 Valenciennes, France
*Computational Research Division
Lawrence Berkeley National Laboratory, Berkeley, CA 94720
melhem.tawk@univ-valenciennes.fr, smail.niar@univ-valenciennes.fr, KZIbrahim@lbl.gov

Abstract: To rapidly evaluate performances and power consumption in design space exploration of modern highly complex embedded systems, new simulation tools are needed. The checkpointing technique, which consists in saving system states in order to simulate in detail only a small part of the application, is among the most viable simulation approaches. In this paper, a new method for generating and storing checkpoints for accelerating MPSoC simulation is presented. Experimental results demonstrate that our technic can reduce simulation time and the memory size required to store these checkpoints on a secondary memory. In addition, the necessary time to load checkpoints on the host processor at runtime is optimized. These advantages speedup simulations and allow exploration of a large space of alternative designs in the DSE.

1 Introduction

Several studies showed that the time needed for designing and verifying next high performance embedded systems will be more and more important due to the increasing system complexity [VAM+06, SBR05, GVH01]. The use of DSE and high level simulation technique for the realization of high performance embedded systems represents an important solution to this dilemma. Nevertheless, several obstacles stand in the way to allow MPSoC simulations being efficient. Among these obstacles, we find: 1) the significant number of parameters to be taken into account in the exploration phase of the MPSoC design 2) The increasing complexity of systems to be simulated. Consequently, the more the architecture is sophisticated and provides more features, the slower is the simulation. For instance, in the MPSoC platform used in our experiments, the simulation speed is slowed down by a factor of 6 when the number of processors increases from 1 to 12.
The work, we present in this paper, aims at the design of a tool for rapid and accurate MPSoC simulation. Our contribution involves supporting the application sampling technique as a viable and competitive method to quickly simulate MPSoC architectures. In this paper, we extend one of the most used sampling methods, namely the Simpoint [SPH02] method, to be efficient for simulating MPSoC. Our proposition consists in adapting of the checkpointing approach to MPSoC by transforming Basic-Block Vectors (BBV) analysis to Basic-Block Matrices (BBM) analysis. This extension makes Simpoint efficient for

MPSoC simulation. We demonstrate that, by our trace analysis and checkpointing, simulation time and memory space overheads can be greatly reduced.
The rest of this paper is organized as follows. An overview of related work on simulation acceleration techniques is presented in Section 2. In this section, the multi-granularity sampling (MGS) approach [TIN08], on top of which the proposed checkpointing technique is evaluated, is presented in detailed. In Section 3, two existing techniques for building the checkpoints before a sample run are discussed. The base line implementation used in the original MGS proposal is also outlined in this section. Our new method of checkpointing based on BBM analysis for MPSoC simulation acceleration is introduced in Section 4 and its performances are given Section 5. Finally, we present our conclusion and perspectives in Section 6.

2 State of the Art

In order to find the most adequate embedded system architecture configuration for a given application or group of applications, it is necessary to explore a large set of possible architecture configurations. The challenge is to reduce this exploration phase in the Design Space Exploration (DSE) in order to reduce the time-to-market. Accelerating the DSE can be done at two levels; either by reducing the number of architectural alternatives to evaluate [PSZ09] and/or by accelerating the evaluation process associated with each alternative. In this paper, we deal with the second alternative. In the literature several different approaches have been proposed for this purpose. Higher abstraction modeling level instead of the cycle description level (such as Transaction Level Modeling TLM) [Don04], analytic modeling [CHE11] , and emulation on FPGA [CPN$^+$09] are among the widely used methods. This paper is based on the application sampling method.

In this approach [SPH02, WWFH03], application execution is first divided in several intervals. From these intervals, some samples are chosen to represent application repetitive phases. The samples have a reduced instruction count compared to the total execution and approximate the behavior of the whole application on the hardware platform. The choice of the samples could be either obtained periodically, as in SMARTS [WWFH03], or by interval analysis as in SimPoint [SPH02]. In the exiting methods, simulation acceleration is obtained by *commuting* the simulator between two states: 1) Detailed-simulation used during the sample execution and 2) Moving to the next sample to simulate in detail. In this case, intervals with known performance estimates are either skipped, thus requiring checkpoints of the system state after the skipped intervals, or simulated with functional simulation. Checkpointing is usually associated with large disk space overhead, while fast-forwarding is usually associated with additional simulation cycles. In this paper, we propose a new checkpointing method for MPSoC sampling-based simulation.

While the detection of identical simulation phases of an application can be accurately done using phase classification technique (such as Simpoint), concurrent applications are more problematic because phases from different applications may not perfectly overlap. In [TIN08], we introduced the Multi-Granularity Sampling (MGS) approach, to addresses this problem. MGS relies on detecting execution phases that are similar and executing one

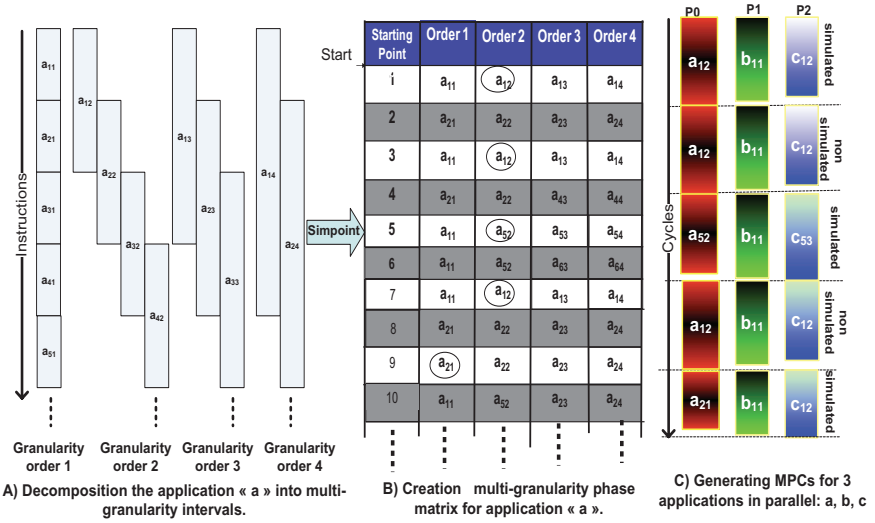


Figure 1: The two stages of the method MGS with 3 processors running 3 applications (a, b, c). The two figures A and B correspond to the first stage in MGS and Figure C is the second one. Notation $A_{(ij)}$ is the $i^{(th)}$ interval of granularity j. Steps A and B are also applied for P2 and P3 to obtain 3 multi-granularity phase matrices for the 3 concurrent applications a, b, and c [TIN08]

representative sample of each of the execution phases in detail. When an execution phase is encountered later during the simulation, it is skipped necessitating loading a checkpointed image of the system state after this phase. The MGS [TIN08] method involves two steps:

1) Multi-Granularity Phase Matrix MGPM Creation: In this step phase-ID traces for each application to run on the MPSoC simulator is individually generated using a rapid functional simulation. These traces are neither dependent on the concurrent processes on the other processors, nor they dependent on a specific MPSoC architectural configuration. Each application is decomposed into intervals of *K* instructions. This value is called here granularity order of 1 and corresponds to 50K instructions in the experiments. For each interval, a basic block vector (BBV), containing the frequencies of executed basic blocks in the corresponding interval, is generated. A phase-ID trace is then generated by examining the similarity between these BBVs using SimPoint [SPH02] classification. Using the same starting points (i.e. a discretization point of 50K instructions), overlapping samples of coarser granularity 100K, 150K, etc. SimPoint is used repeatedly to detect similarity for these coarse granularity phases.

Simpoint is used to detect the similarity for each granularity in order to generate Multi-Granularity Phase Matrix MGPM (1.B) . Each line in the MGPM corresponds to one starting point in the execution of the application. This first stage is executed once and statically for all the granularities expected during runtime and a rapid functional simulation is used to obtain phase matrices in a few minutes. Granularities up to 20 seemed sufficient

Table 1: A multi-phase cluster table (MPCT) containing 4 MPCs and their metrics. This MPCT corresponds to figures 1.b and 1.c. g_i corresponds to the phase granularity (in K inst) for processor i. Here we assume having 3 parallel applications (a, b, c) on 3 processors (P0, P1, P2). For example, the first cluster that is met in the execution consists of 3 intervals: interval 1 of granularity_2 for P0, interval 1 of granularity_1 for P1 and interval 1 of granularity_2 for P3

MPC	g_1	g_2	g_3	K-Cycles	Energy(mJ)	Repetition
a_{12}, b_{11}, c_{12}	100	50	100	280	104	3
a_{52},b_{11},c_{53}	100	50	150	487	190	1
a_{21},b_{11},c_{12}	100	50	150	277	150	1

in our experiments, requiring less than 10 minutes of profiling per application.

2)Multi-Phase Cluster (MPC) generation: The second stage uses the MGPMs that have been generated in the first step. Phases that are executed in parallel by processors are combined together to form a multi-phase cluster (MPC). These parallel phases are determined dynamically at runtime as follows: An MPC contains *p* parallel phases, where *p* is the number of processors in the MPSoC. Each new MPC is assigned an entry in the multi-phase cluster table (MPCT)(see Table 1). MPC containing the same parallel phases have the same behavior, and thus can be skipped during simulation after estimating their performance once. Discretization of the overlapping execution phases is adopted. In order to generate the MPCs, the granularity of each of the *p* phases is determined dynamically. Whenever a process reaches the boundary of 50K instructions, a decision is made as to whether to continue the simulation or to stop to form an MPC. The number of cycles needed to finish the current interval for each processor is calculated. If the maximum number of cycles is less than a certain predefined threshold TWSS[1], then the approach enters the cluster formation mode, in which each process tries to finish the current phase and then waits for the other processes. In the MPCT, the combined phase identifiers, which are executed in parallel, represent a unique MPC identifier and MPCs with the same identifiers are assumed to have the same performance.

In Table 1, after simulating the first MPC (a_{12};b_{11};c_{12}), this MPC recurs 3 times. Since it already exists in the MPCT, the repetition entry is simply incremented, and the repeated MPCs are not simulated in details.

In the work we present in this paper, the purpose is to make the simulation by sampling faster, than MGS. For this reason, we propose to eliminate the fast-forwarding phase and the checkpointing method is used to minimize the simulation time. We extend phase trace generation and classification technique based on Simpoint [SPH02], by analysis of all the granularities that can be encountered at runtime. Therefore, the same start point of program execution is associated with multiple classifications based on the granularity of the phase (instruction count), as will be outlined in the following discussion.

[1]Threshold Waiting at Simulation Synchronization is denoted by TWSS.

3 System context construction for simulation acceleration

The use of application sampling for simulation acceleration requires addressing the following two issues: 1) Restore the exact image of the application context also named *starting sample image* and 2) Restore the micro-architectural state also called *warming-up sample*. The architectural state is represented by the values of data found in registers of different processors and in private or shared memories in the case of multi-processor architectures. These data represent the context of the application and are independent of the architecture configuration. On the other hand, the micro-architectural state corresponds to content of caches, the branch prediction table, pipeline, etc [BCE06, WWFH06]. The previous two tasks must be realized before starting the detailed simulation of each interval. They are usually made either by a functional simulation or by storing/loading checkpoints. In contrast to functional simulation, checkpointing reduces the simulation time by eliminating the time consumed by FFW [WWFH06] between samples. It consists in saving an image containing the two states mentioned above, for each cluster. The checkpoints are collected offline by simulating in details the entire application once for the entire operation of the DSE. The disadvantage of checkpointing is the large memory size required for storage on disk. This reduces the simulation acceleration, since additional time is needed to access the disk and to load the checkpoint.

4 Basic Block Matrices for MPSoC checkpointing

For MPSoC platform simulation with important number of processors is increased, the number of checkpoints and the memory space to store the checkpoints could be very important . The method for selecting checkpoints that we present in this paper aims to solve this problem. Our method reduces the number of checkpoints by static analysis, prior to execution, of different intervals for different granularities. It allows the user to set the number of checkpoints to be stored depending on the memory available on the host. The idea is to combine the different basic block vectors (or BBV) of different granularities associated with a starting point of the simulation in a matrix. This matrix is called matrix of basic blocks or BBM. Subsequently, a search of the similarity between the BBM of all starting point is made. This analysis is independent from micro-architectural configuration and allows detecting multi-granular phases of the application. In fact, when two BBMs are identical, this corresponds to intervals of the same multi-granular phase of the application. These phases are denoted in the following by MPhases, for Multi-granular phase. When two BBM belong to the same MPhase, this means that the instructions executed in intervals of both BBM, taken in pairs, are similar. In other words, whatever the considered interval size the work done is the same. Therefore, the execution of one BBM replaces the other, and more importantly a checkpoint of one BBM can be used to start the other. Thus, all BBMs will be provided to a classification tool in order to select a number of MPhases set by the user and representing, in the best way, all the BBMs of the application. The classification tool selects one BBM for each MPhase of the application.

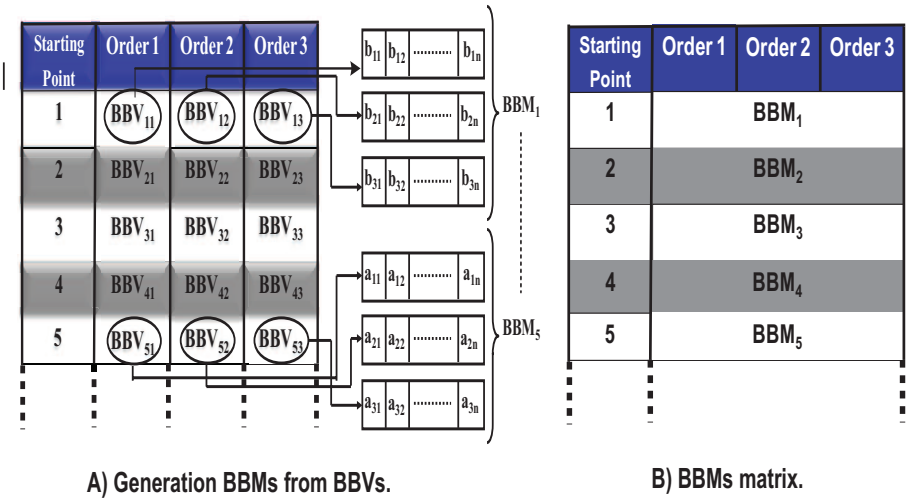


Figure 2: Generation of BBM from BBV. Here we suppose that the number of basic blocs is n and bij corresponds to the number of occurrences of basic bloc j of size of granularity i. BBMi corresponds to the concatenation of columns: BBVi1, BBVi2, ,BBVi3., .etc

4.1 Generation of Basic Block Matrices

Figure 2.A shows the construction of BBMs from BBVs with a granularity equals to 3. There are BBMs as many as rows in the phase matrix. All the constructed BBMs form the "BBM vector" represented in Figure 2.B Each BBM contains gxn elements; g and n correspond respectively to the considered granularity and to the total number of basic blocks of the application. In Figure 2, g corresponds to 3.
To detect the MPhases of the application, the BBMs are compared by a profiling tool such SimPoint.

4.2 Classification of Basic Block Matrixes

Once the similarity degree is calculated, the similar (or almost similar) BBMs are gathered in one group, called *MPhase*. One BBM, called *representative BBM*, is chosen to represent each MPhase. This allows us to represent the behavior of the whole application by the representative BBMs. Indeed, the classification aiming at distributing the BBMs into MPhases so that BBMs belonging to the same MPhase are very similar, and BBMs belonging to distinct MPhases are different from each other.
Thus, we are inspired from the SimPoint methodology to achieve BBMs classification. This is based on the k-means algorithm. However, any classification methodology, such as [JH08], can be used to realize this classification. Subsequently, a classification algorithm, such as *k-means*, is applied to BBMs . This algorithm achieves a distribution of

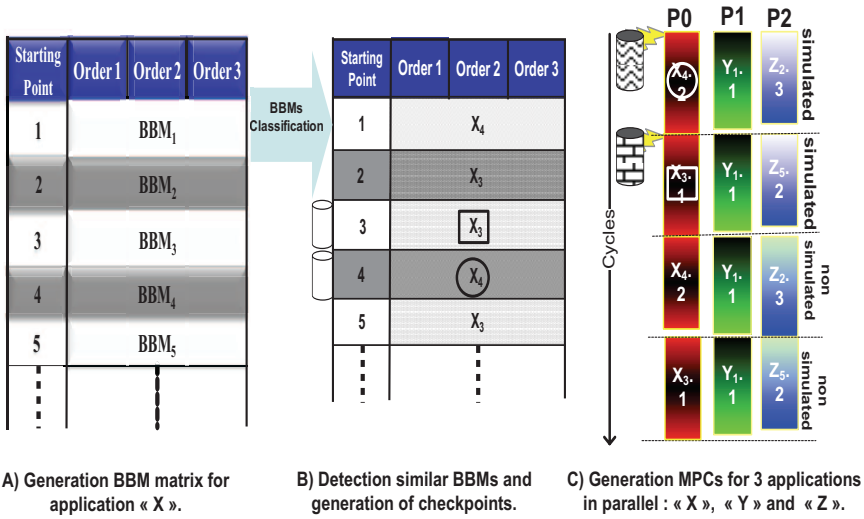


Figure 3: The similar BBMs in Figure B have the same identifier. The identifier is composed of Mphase application name and the index of its representative BBM. The cylinders represent the collected checkpoints. C) $X_i.J$ means that the instructions executed belong to the Mphase X_i and to the order of granularity j.

BBMs in k different MPhases ($1 \leq k \leq K_{max}$). To choose the value of k, SimPoints uses the *Bayesian Information Criterion* (BIC) [12] to compare the quality of the classification of different values of k. Indeed, the classification must provide two things, first, the smallest possible value for k, and the second one is all the BBMs belonging to the same MPhase must be very similar. Here, the chosen classification is the one that gives the smallest value of k with a BIC score at least 90% of the best BIC score obtained. BIC simply indicates the degree of similarity of all BBMs in an MPhase. The last step is to elect a representative BBM of each MPhase. This represents the behavior of the whole MPhase. The centroid of the BBM is calculated as the average of all BBMs of the MPhase. In these cases, the representative BBM of the MPhase is one that has the closest distance from the centroid of MPhase. Since there are as many representative BBMs as MPhases, *k* BBMs are selected to represent the behavior of the entire application and representative checkpoints are collected at starting points of the *k* representative BBMs. Figure 3.A presents an application X as a suite of BBMs that are collected offline. Figure 3..B shows the classification of BBMs of the application X. The similar BBMs have the same MPhase identifier. Here, the identifier of the MPhase corresponds to the name of the application (X) followed by the index of the representative BBM of the considered MPhase. For example, the first and fourth BBM are considered similar by the static analyzer and thus have the same identifier MPhase. The fourth BBM, called BBM_4, was selected as representative BBM of that MPhase. Thus the identifier of that MPhase is X_4 in the Figure. Figure 3.B shows two MPhases and two representative BBMS BBM_3 and BBM_4. As the checkpoints are collected at starting points of representative BBMs, checkpoints in Figure 3.B are collected

at the third and fourth starting point. These checkpoints are represented by the different cylinders in Figure 3.B.

4.3 Use of BBM by MGS to Accelerate Simulation

We illustrate the use of our checkpointing method in Figure 3. Checkpoints are collected at starting points 3 and 4 for the application X and there are as many checkpoints as representative BBMs for each of the concurrent applications, as shown in Figure 3.B. For each processor of MPSoC architecture, the associated checkpoint with the representative BBMs of the phases to be executed, is loaded in the memory of the host. In Figure 3.C, P0 begins with the detailed simulation of the MPhase X_4. The representative BBM of that MPhase is the fourth BBM in the matrix (see Figure 3.B). It corresponds to the starting point 4. Thus, the checkpoint of that BBM is loaded for P0 at the beginning of the first MPC and the fourth BBM is executed. As the granularity of instructions executed in an MPC is determined dynamically by MGS, it is impossible to know a priori the granularity of the phases. The first MPC corresponds to a phase of granularity 2 for P0. Subsequently, the second interval of P0 starts at the point 3. As for the second MPC, P0 load the checkpoint of the representative BBM corresponding to point 3 of the execution. In this example the representative BBM at this point of execution is the same, denoted X_3 The same technique is applied for processors P1 and P2 in our example. During the simulation, an MPC is considered similar to an MPC already simulated if the MPhase in the simulated MPC and that in the trace of MPhases are similar. Thus in Figure 3.C, the third MPC is considered similar to the first MPC. Similarly the fourth MPC is considered similar to the second MPC. The advantages of the method of checkpoint generation by BBMs versus the method of similarity lines proposed in [TIN08] is the fact that the representative BBMs correspond to the centers of MPhases. This allows a more accurate performance approximation than taking the first occurrence of the MPhase during the simulation.

5 Experimental Results

In this section, we quantify the amount of saving in checkpointing storage when the the BBM technique is used in conjunction with MGS. Note that since trace analysis and checkpointing generation is made only once and offline, simulation acceleration factors are at least identical to those given by MGS [TIN08].

5.1 Checkpoints construction

To reduce required memory size for storing checkpoints, the Touched Memory Image (TMI) was implemented in conjunction with MGS and BBM-Based checkpointing. In TMI, checkpoints, which do not contain memory regions that are not modified by the application phases, are not integrated in the checkpoints. In addition, in a checkpoint state instead of having an address for every memory position, only one address is stored

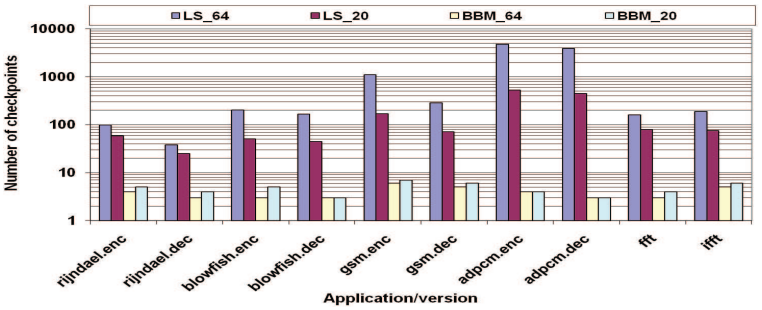


Figure 4: Variation of the number of checkpoints to store checkpoints in the similar line method (LS) and in the BBM method (BBM) with G_{max} 20 and 64

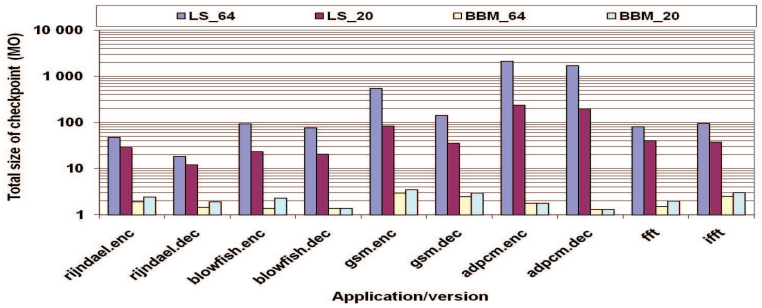
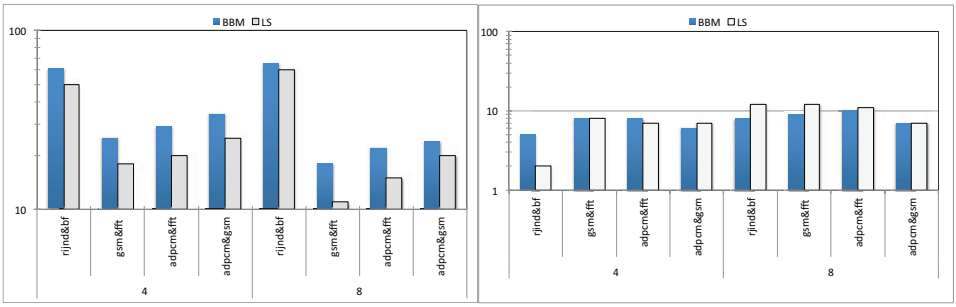


Figure 5: Variation of the total memory size in MB to store the checkpoints for LS and for the BBM with G_{max} 20 and 64

to represent a sequence of consecutive data in the memory. We also used the Memory Hierarchy Sate (MHS) technique [BCE06]. In MHS, cache contents are created from the largest reference cache (64KB in our case). Thanks to this method, the size of the architectural state of each processor in each checkpoint is limited to 200KB, for the studied benchmarks.

5.2 Number of checkpoints reduction with BBM-based checkpointing

In this section, the required memory size for BBM-based checkpointing is compared to the required size when line similarity (LS) and BBV [TIN08] are used. We conducted the experiments with G_{max} of 20 and 64 . These values are the largest observed granularity for the MiBench application combinations [GRE$^+$01]. As in [TIN08], we used an interval size of 50K instructions as the order-1 granularity. Finally, we fixed K_{MAX} to 10. Figure 4 gives the number of checkpoints for the two approaches BBM and LS. In this figure, we notice that the number checkpoints increases with granularity in LS. This is due to the fact that similarity between lines diminish with the increase of the line size. For example, for the adpcm application and granularity of 64, more than 3000 checkpoints must be stored.



(a) Simulation speedup factor

(b) Estimated IPC error

Figure 6: Simulation speedup factor and estimation error (in%) for 4 combinations of benchmarks provided by MGS with BBM-Based Checkpointing and compared with Line Similarity technique [TIN08]. The number of processors is 4 and 8.

In contrast with BBM, this number depends only on K_{Max}. For K_{Max} of 10, the number of checkpoints is less than 10. We also notice that the number of checkpoints is constant with G_{max}. Even better, for the majority of the executed applications, this number slightly decreases because the BBM contains intervals that partially recur. Indeed, when phase granularity increases, the degree of similarity between intervals and consequently between BBM also increases.

5.3 Required memory size reduction with BBM-based checkpointing

Figure 6b gives the required memory size to store all collected checkpoints with LS and BBM. For instance, with adpcm and a granularity of 64, the maximum memory size is 3MB for each processor when BBM is used. In the same conditions, LS requires a 1000 MB memory size. If we assume a 12-processor MPSoC, 12 GB of RAM is needed for LS while 16 MB of RAM is sufficient in the case of BBM. The advantage of fewer and smaller checkpoints allows holding all of these checkpoints on the physical memory (RAM) instead of streaming them from the hard-drive. Consequently, the time overhead for warming the systems by checkpoints restoring is reduced thus allowing to increase the simulation speedup.

5.4 Performance Analysis of MGS with BBM-Based Checkpointing

In this section, we present the performance in terms of simulation factor and the *IPC* estimation accuracy while using *MGS* with *BBM*-Based Checkpointing.

Figure 6a gives the simulation acceleration for 4 different benchmark combinations, for 4 and 8 processors using *MGS* with *BBM*-Based Checkpointing. In these experiments different concurrent applications are executed. For example with Rijndael-Bf benchmark on

4 processors, Rijndael-encrypt, Rijndael-decrypt, Blowfish-encrypt and Blowfish-decrypt are executed in parallel on each of the processor [GRE+01]. The simulation factor for concurrent applications varies generally between 20x and 60x (for rijndael and blowfish). The average simulation factor is about 35x for all the studied applications. It is clear that the simulation factor obtained by our method is significantly important without sacrificing accuracy as shown in Figure 6.B.

In [TIN08] it has been mentionned that the *IPC* estimation error has two sources. The first source corresponds to the added waiting cycles that are injected at simulation discretization points. The second source of error is the association of the *IPC* of recurring *MPC*s with the *IPC* of the first simulated *MPC*. This is due to the fact that intervals classified as similar may not have exactly the same performance. In the case of *BBM*-Based checkpointing the representative *BBM*s correspond to the centers of *MPhases*. The representative *BBM*s based on selecting the centers of *MPhases* is more accurate than taking the first occurrence of the *MPhase* during the simulation. This approach reduces the error in the performance estimation. This explained the low error in *IPC* obtained by our method. Based on the last figure, the error is smaller than 10%. Additionally that error does not significantly increase with the number of processors increase.

6 Conclusion

Simulation Acceleration for embedded systems is crucial to explore large design space under stringent time-to-market constraints. Multi-granularity sampling technique provides an attractive solution for the simulation of heterogeneous concurrent applications by analyzing each application for multiple granularity level, thus allowing to overlap samples of phase executions in an efficient and accurate manner. Checkpoints of the system state are required to skip repeated phases, but they require large storage space that makes simulation speed limited by the speed of the hard disk of the system. In this work we introduce a space efficient mechanism to select the most representative checkpoints. We propose to use classification methods, such as k-means, to identify the checkpoints that are most representative for a large set of checkpointed states. The classification scheme allows the user to control the maximum checkpointed states of the system, with little sacrifice in accuracy. In contrast, the base checkpointed states grow quickly with the number of concurrent applications, in the case of multi-granularity sampling technique. We showed that the space needed of the checkpointed states can be reduced by orders of magnitude compared with the base scheme for multi-granularity sampling, thus allowing to keep all needed checkpoints in the memory of the host machine. Simulation acceleration are significantly speedup, without impacting accuracy, when using multi-granularity sampling in conjunction with the introduced representative checkpoints.

References

[BCE06] Michael Van Biesbrouck, Brad Calder, and Lieven Eeckhout. Efficient Sampling Startup for SimPoint. *IEEE Micro*, 26(4):32–42, 2006.

[CHE11] Trevor E. Carlson, Wim Heirman, and Lieven Eeckhout. Sniper: exploring the level of abstraction for scalable and accurate parallel multi-core simulation. *SuperComputing Conference*, Mar. 2011.

[CPN+09] E. Chung, M. Papamichael, E. Nurvitadhi, J. Hoe, K. Mai, and B. Falsafi. ProtoFlex: Towards Scalable, Full-System Multiprocessor Simulations Using FPGAs. *ACM Trans. Reconfigurable Technol. Syst.*, 2(2), 2009.

[Don04] A. Donlin. Transaction Level Modeling: Flows and Use Models. *Conf. on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*, pages 75–80, Sep. 2004.

[GRE+01] M. R. Guthaus, J. S. Ringenberg, D. Ernst, T. M. Austin, T. Mudge, and R. B. Brown. MiBench: A Free, Commercially Representative Embedded Benchmark Suite. *Worksh. on Workload Characterization (WWC)*, pages 3–14, Dec. 2001.

[GVH01] T. Givargis, F. Vahid, and J. Henkel. System-level Exploration for Pareto-optimal Configurations in Parameterized System-on-a-chip. *Conference on Computer-aided design (ICCAD)*, pages 416–422, Nov. 2001.

[JH08] J. Johnston and G. Hamerly. Improving simpoint accuracy for small simulation budgets with edcm clustering. *Worksh. on Statistical and Machine learning approaches to ARchitectures and compilaTion (SMART08)*, Jun. 2008.

[PSZ09] G. Palermo, C. Silvano, and V. Zaccaria. ReSPIR: a response surface-based Pareto iterative refinement for application-specific design space exploration. *Trans. Comp.-Aided Des. Integ. Cir. Sys.*, 28(12), 2009.

[SBR05] J. Schnerr, O. Bringmann, and W. Rosenstiel. Cycle Accurate Binary Translation for Simulation Acceleration in Rapid Prototyping of SoCs. *The Design, Automation and Test in Europe (DATE)*, pages 792–797, Mar. 2005.

[SPH02] T. Sherwood, E. Perelman, and G. Hamerly. Automatically Characterizing Large Scale Program Behavior. *The 10th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, pages 45–57, Oct. 2002.

[TIN08] M. Tawk, Khaled Z. Ibrahim, and S. Niar. Multi-granularity sampling for simulating concurrent heterogeneous applications. *Proceedings of the 2008 international conference on Compilers, architectures and synthesis for embedded systems (CASES08)*, 2008.

[VAM+06] P. G. Del Valle, D. Atienza, I. Magan, J. G. Flores, J. M. Mendias E. A. Perez, L. Benini, and G. De Micheli. Architectural Exploration of MPSoC Designs Based on an FPGA Emulation Framework. *Conf. on Design of Circuits and Integrated Systems*, pages 12–18, Dec. 2006.

[WWFH03] R. E. Wunderlich, T. F. Wenisch, B. Falsafi, and J. C. Hoe. Smarts: Accelerating Microarchitecture Simulation via Rigorous Statistical Sampling. *Symposium on Computer Architecture (ISCA)*, pages 84–97, Jun. 2003.

[WWFH06] T. Wenisch, R. Wunderlich, B. Falsafi, and J. Hoe. Simulation Sampling with LivePoints. *Symposium on Performance Analysis of Systems and Software (ISPASS)*, pages 2–12, Mar. 2006.

PASA 2012

10th Workshop on Parallel Systems and Algorithms

organized by

Jörg Keller, FernUniversität in Hagen, Germany
Rolf Wanka, Universität Erlangen-Nürnberg, Germany

Resilient data encoding for fault-prone signal transmission in parallelized signed-digit based arithmetic

David Neuhäuser and Eberhard Zehendner

Institute of Computer Science
Friedrich Schiller University
D-07737 Jena, Germany
{david.neuhaeuser,nez}@uni-jena.de

Abstract: When arithmetic components are parallelized, fault-prone interconnections can tamper results significantly. Constantly progressing technology scaling leads to a steady increase of errors caused by faulty transmission. Resilient data encoding schemes can be used to offset these negative effects. Focusing on parallel signed-digit based arithmetic frequently used in high-speed systems, we propose suitable data encodings that reduce error rates by 25%. Data encoding should be driven by the occurrence probabilities of digits. We develop a methodology to obtain these probabilities, show an example fault-tolerant encoding, and discuss its impact on communicating parallel arithmetic circuits in an example error scenario.

1 Introduction

In times of billion-transistor processors being commercially available and transistors being processed in 22 nanometer CMOS process [ITR11], it becomes more and more difficult to design fault tolerant [NSF01, RSKW07] and mixed critical systems [PMN+09]. More complex circuits require increased inter- and intra-circuit connections which become increasingly fault-prone.

Focusing on fast, parallelized, signed-digit based arithmetic, used extensively for instance in CORDIC arithmetic, we propose a data encoding which can significantly lower transmission error rates. Our data encoding principle is based on occurrence probabilities of digits. We show that digit probabilities in signed-digit arithmetic converge when results of addition operations are iteratively reused as input to other addition operations. Digits with the highest limit probability should have more than one bit level encoding. Some errors at bit level would result in unchanged values at digit level. We apply our methodology exemplarily to 2-bit encodings and provide an error rate optimal encoding.

Alternative approaches like using check symbols have been proposed [COP+06], which are less efficient in terms of latency, since every arithmetic operation has to be done multiple times to obtain error information.

In the following section we discuss the signed-digit arithmetic used. In Section 3 we show our methodology to obtain digit probabilities for signed-digit encoded data. In Section 4

we discuss a possible communication error scenario, where fault tolerant data encoding can reduce error probabilities, and give recommendations for error resilient encoding. Applying our methodology, we provide accurate data word probabilities for common signed-digit adder cell implementations in Section 5 and present error rates for different encoding schemes. We conclude in Section 6 and give an outlook to future work.

2 Signed-digit arithmetic

A special case of a signed-digit [Avi61] number system is a signed-binary number system, where each digit is limited to $\{-1, 0, 1\}$. In the following we focus on signed-binary number systems. A signed binary number is defined as

$$Z_{sb} = (z_{n-1}, ..., z_0), z_i \in \{-1, 0, 1\}, 0 \leq i < n \quad (1)$$

$$I(Z_{sb}) = \sum_{i=0}^{n-1} 2^i \cdot z_i \quad (2)$$

where $z_i \in \{-1, 0, 1\}$ and $I : \{-1, 0, 1\}^n \to \mathbb{Z}$ is the interpretation function.

A signed-binary adder (SBA) calculates $S_{sb} = A_{sb} + B_{sb}$ which corresponds to $S = A + B$ with $A, B, S \in \mathbb{Z}$, $I(A_{sb}) = A$, $I(B_{sb}) = B$, and $I(S_{sb}) = S$. We decompose this into digit operations:

$$S_{sb} = A_{sb} + B_{sb} \quad (3)$$

$$\sum_{i=0}^{n-1} s_i = \sum_{i=0}^{n-1} 2^i \cdot a_i + \sum_{i=0}^{n-1} 2^i \cdot b_i; s_i, a_i, b_i \in \{-1, 0, 1\} \quad (4)$$

$$= \sum_{i=0}^{n-1} 2^i \cdot (a_i + b_i) \quad (5)$$

Figure 1 shows this decomposition. One operation at digit i calculates $s_i = a_i + b_i$. Since $a_i, b_i \in \{-1, 0, 1\}$, $a_i + b_i \in \{-2, -1, 0, 1, 2\}$, but $s_i \in \{-1, 0, 1\}$, we need some carry to propagate $\{-2, 2\}$ to the digit at $i + 1$. Focusing on a 3-level design as in Chow and Robertson [CR78], we introduce $c_i \in \{-1, 0\}$ and $d_i \in \{0, 1\}$ as a solution. We include c_i and d_i in Equation 5:

$$S_{sb} = \sum_{i=0}^{n-1} 2^i \cdot (a_i + b_i + c_i + d_i - 2 \cdot c_{i+1} - 2 \cdot d_{i+1}) \quad (6)$$

Here c_0 and d_0 are the carry-ins of the whole adder, set to 0 in normal operation. The carry-outs of the whole adder are c_n and d_n. For any i the signed-binary adder cell (SBAC) calculates:

$$s_i + 2 \cdot c_{i+1} + 2 \cdot d_{i+1} = a_i + b_i + c_i + d_i, \quad (7)$$

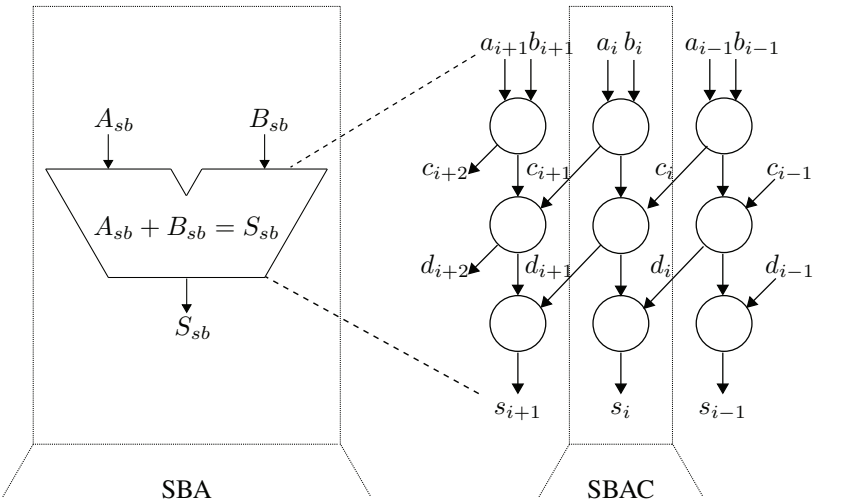


Figure 1: Signed-binary adder (SBA) consisting of three level signed-binary adder cells (SBAC) shown at numerical level, see [CR78, S.111].

The calculation of c_{i+1} must be independent ($\perp$) from c_i and d_i. the calculation of d_{i+1} must be independent from d_i, see again Figure 1. By enforcing these independencies, the remaining carry chain is locally constraint, the calculation of any s_i depends only on $a_i, b_i, a_{i-1}, b_{i-1}, a_{i-2}, b_{i-2}$, see also [Zeh92].

3 Digit probabilities

We now describe a SBAC through atomic operations that are in accordance to Equation 8.

$$e_i = a_i + b_i \tag{8}$$

$$c_{i+1}(t, a_i, b_i) = \begin{cases} 0 & \text{when } e_i > 0, \\ -1 & \text{when } e_i < 0, \\ \gamma(t, a_i, b_i) & \text{when } e_i = 0. \end{cases} \tag{9}$$

$$f_i = e_i - 2 \cdot c_{i+1}(t, a_i, b_i) \tag{10}$$

$$g_i = f_i + c_i \tag{11}$$

$$d_{i+1} = \begin{cases} 0 & \text{when } g_i \leq 0, \\ +1 & \text{when } g_i > 0. \end{cases} \tag{12}$$

$$h_i = g_i - 2 \cdot d_{i+1} \tag{13}$$

$$s_i = h_i + d_i = a_i + b_i + c_i + d_i - 2 \cdot c_{i+1} - 2 \cdot d_{i+1} \tag{14}$$

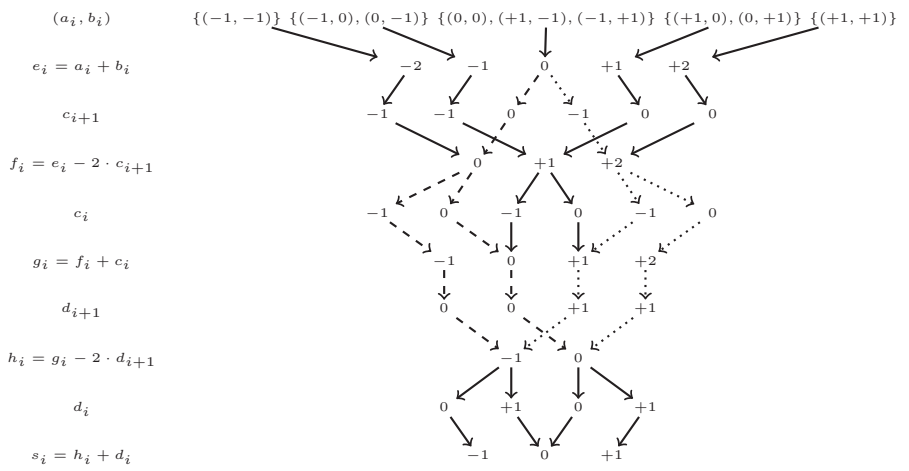


Figure 2: Signed-binary adder cell decision graph. For $a_i + b_i = 0$, the dashed graph denotes a choice of $c_{i+1} = 0$, the dotted graph a choice of $c_{i+1} = -1$. For $a_i + b_i = 0$ it is obvious, that d_{i+1} depends on the choice of c_{i+1} but not on c_i. Furthermore, for $a_i + b_i = 0$,s_i does not depend on the choice of c_{i+1}, but s_{i+1} does depend on the choice of c_{i+1} through d_{i+1}.

We construct a decision graph, see Figure 2, that shows all possible degrees of freedom when constructing a functionally correct SBAC that is constrained by the formal model from Section 2. To enforce the independence constraints , the adder cell has no knowledge of c_i and d_i when calculating c_{i+1}, and no knowledge of d_i when calculating d_{i+1}. Some choices of c_{i+1} and d_{i+1} may therefore be wrong, when worst case values of c_i or d_i occur. In Figure 2, all impossible choices of c_{i+1} and d_{i+1} have already been removed. There are still left some degrees of freedom in choosing c_{i+1} and d_{i+1}, but by fixing a choice on c_{i+1}, we lose all freedom of choice in d_{i+1}. We illustrated the choice of $c_{i+1} = -1$ by dotted arrows and of $c_{i+1} = 0$ by dashed arrows. We see by the dotted and dashed paths, that this choice also fixes the decision of d_{i+1}.

Our SBAC model offers $2^3 = 8$ different signed-binary adder cells at the numerical level. Let t be the *type id* of the design choice, $0 \leq t < 2^3$. All possible design choices are

t	$\gamma(t,0,0)$	$\gamma(t,+1,-1)$	$\gamma(t,-1,+1)$
0	0	0	0
1	0	0	-1
2	0	-1	0
3	0	-1	-1
4	-1	0	0
5	-1	0	-1
6	-1	-1	0
7	-1	-1	-1

Table 1: Meaning of parameter t in description of SBAC.

l_i	z	$P(l_i = z)$
e_i	-2	$P(a_i = -1) \cdot P(b_i = -1)$
	-1	$P(a_i = 0) \cdot P(b_i = -1) + P(a_i = -1) \cdot P(b_i = 0)$
	0	$P(a_i = 0) \cdot P(b_i = 0) + P(a_i = -1) \cdot P(b_i = +1) +$ $P(a_i = +1) \cdot P(b_i = -1)$
	$+1$	$P(a_i = 0) \cdot P(b_i = +1) + P(a_i = +1) \cdot P(b_i = 0)$
	$+2$	$P(a_i = +1) \cdot P(b_i = +1)$
c_{i+1}	-1	$P(e_i = -2) + P(e_i = -1) + P(\gamma(t, a_i, b_i) = -1)$
	0	$P(e_i = +1) + P(e_i = +2) + P(\gamma(t, a_i, b_i) = 0)$
f_i	0	$P(e_i = -2) + P(\gamma(t, a_i, b_i) = 0)$
	$+1$	$P(e_i = -1) + P(e_i = +1)$
	$+2$	$P(e_i = +2) + P(\gamma(t, a_i, b_i) = -1)$
g_i	-1	$P(f_i = 0) \cdot P(c_i = -1)$
	0	$P(f_i = 0) \cdot P(c_i = 0) + P(f_i = +1) \cdot P(c_i = -1)$
	$+1$	$P(f_i = +2) \cdot P(c_i = -1) + P(f_i = +1) \cdot P(c_i = 0)$
	$+2$	$P(f_i = +2) \cdot P(c_i = 0)$
d_{i+1}	0	$P(g_i = -1) + P(g_i = 0)$
	$+1$	$P(g_i = +1) + P(g_i = +2)$
h_i	-1	$P(g_i = -1) + P(g_i = +1)$
	0	$P(g_i = 0) + P(g_i = +2)$
s_i	-1	$P(h_i = -1) \cdot P(d_i = 0)$
	0	$P(h_i = -1) \cdot P(d_i = +1) + P(h_i = 0) \cdot P(d_i = 0)$
	$+1$	$P(h_i = 0) \cdot P(d_i = +1)$

Table 2: Probability level description of $SBAC_t$.

shown in Table 1. Note that the formula for calculating c_{i+1} depends on the input digits (a_i, b_i) and the chosen design parameter t. Let $SBAC_t$ be the design using choice t to calculate c_{i+1}.

Assigning probability information to the symbols in Equations 8 through 12 we are able to calculate the digit probabilities. At probability level $SBAC_t$ is described as shown in Table 2.

$P(\gamma(t, a_i, b_i) = 0)$ and $P(\gamma(t, a_i, b_i) = -1)$ are calculated in accordance to Table 1 as

$$\begin{aligned} P(\gamma(t, a_i, b_i) = 0) \quad = \quad & P(a_i = 0) \cdot P(b_i = 0) \cdot P(t \in \{0, 1, 2, 3\}) + \\ & P(a_i = +1) \cdot P(b_i = -1) \cdot P(t \in \{0, 1, 4, 5\}) + \\ & P(a_i = -1) \cdot P(b_i = +1) \cdot P(t \in \{0, 2, 4, 6\}) \end{aligned}$$

$$
\begin{aligned}
P(\gamma(t,a_i,b_i)=-1) &= P(a_i=\ \ 0)\cdot P(b_i=\ \ 0)\cdot P(t\in\{4,5,6,7\}) + \\
&\quad P(a_i=+1)\cdot P(b_i=-1)\cdot P(t\in\{2,3,6,7\}) + \\
&\quad P(a_i=-1)\cdot P(b_i=+1)\cdot P(t\in\{1,3,5,6\})
\end{aligned}
$$

Note that $P(e_i=0)=P(\gamma(t,a_i,b_i)=0)+P(\gamma(t,a_i,b_i)=-1)$.

4 Digit error scenario

Figure 3 shows two circuits exchanging digits by signal lines 0 through $n-1$. On each line, the signal is sent as $l_i \in \{0,1\}$ and received as $T(l_i) \in \{0,1\}$. The digits received may differ from the digits sent due to imperfect wiring [SOHH07, KPKJ07].

In our simple error model, p_{bf} denotes the probability, that one bit is inverted. The possibility of a bit flip leads to

$$
\begin{aligned}
T(l_i) &= \begin{cases} l_i & \text{when no bit flip occurred,} \\ 1-l_i & \text{else.} \end{cases} \\
P(T(l_i)=l_i) &= 1-p_{bf}
\end{aligned}
$$

When encoding signed-binary digits $\{-1,0,1\}$ with two bits, we can leave one bit combination unused or encode one of the digit values by two different bit combinations. We call the first non-redundant, the second redundant encoding. When using redundant encoding, our SBAC outputs only one code for the double encoded digit. The other code can only occur by faulty transmission, but is interpreted as a correct double encoded digit. Table 3 shows the effects of using a 2-bit redundant encoding of signed-binary digits with such a correction in comparison to a non-redundant encoding with no error correction. $T_{nr}(dw)$ is the result of transporting the 2-bit data word dw with non-redundant encoding and no error correction, $T_r(dw)$ is the result of transporting the data word with redundancy and error correction.

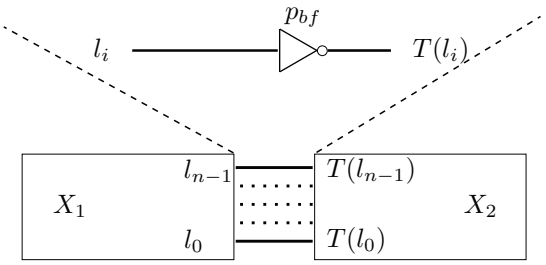


Figure 3: Circuit X_1 communicates with circuit X_2 through signal lines l_0 to l_{n-1}. Our simple error model consists of a possible bit flip with probability p_{bf}.

	no redundancy		redundancy	
example encoding	data word	error prob.	data word	error prob.
00	dw0	$2 \cdot p_{bf} - p_{bf}^2$	dw0	p_{bf}
01	unused	—	defected dw0	—
10	dw1	$2 \cdot p_{bf} - p_{bf}^2$	dw1	$2 \cdot p_{bf} - p_{bf}^2$
11	dw2	$2 \cdot p_{bf} - p_{bf}^2$	dw2	$2 \cdot p_{bf} - p_{bf}^2$

Table 3: Simple error model: Error reduction by using gray code adjoined encoding for data words $dw0$, $dw1$, and $dw2$. All calculations using the redundant encoding consist of a 01 to 00 correction.

The probability of an uncorrected error in $dw0$ in our example is reduced

$$P(T_{nr}(dw0) \neq dw0) = 2 \cdot p_{bf} - p_{bf}^2 \quad > \quad 2 \cdot p_{bf} - p_{bf} = p_{bf} \\ = P_{red}(T_r(dw0) \neq dw0) \tag{15}$$

The error probability for any 2-bit data word $dw \in \{dw0, dw1, dw2\}$ with no error correction can be calculated as

$$\begin{aligned} P(T_{nr}(dw) \neq dw) &= P(dw = dw0) \cdot (2 \cdot p_{bf} - p_{bf}^2) + P(dw = dw1) \cdot \\ &\quad (2 \cdot p_{bf} - p_{bf}^2) + P(dw = dw2) \cdot (2 \cdot p_{bf} - p_{bf}^2) \\ &= (P(dw = dw0) + P(dw = dw1) + P(dw = dw2)) \cdot \\ &\quad (2 \cdot p_{bf} - p_{bf}^2) \\ &= 1 \cdot (2 \cdot p_{bf} - p_{bf}^2) = 2 \cdot p_{bf} - p_{bf}^2 \end{aligned}$$

In comparison to an applied error correction

$$\begin{aligned} P(T_r(dw) \neq dw) &= P(dw = dw0) \cdot p_{bf} + P(dw = dw1) \cdot \\ &\quad (2 \cdot p_{bf} - p_{bf}^2) + P(dw = dw2) \cdot (2 \cdot p_{bf} - p_{bf}^2) \\ &= P(dw = dw0) \cdot p_{bf} + (P(dw = dw1) + P(dw = dw2)) \cdot \\ &\quad (2 \cdot p_{bf} - p_{bf}^2) \end{aligned}$$

With equation 16 we get

$$P(T_{nr}(dw) \neq dw) \quad > \quad P(T_r(dw) \neq dw)$$

The encoding strategy is rather simple: Use the redundant encoding $00, 01$ to encode the digit with the highest probability of occurrence to reduce the error probability. The error ratio of this strategy can be calculated by

$$\begin{aligned} \text{error ratio (of red dw encoding)} &= \frac{\text{error rate of dw red encoding}}{\text{error rate of dw non-red encoding}} \\ e(dw) &= \frac{P(T_r(dw) \neq dw)}{P(T_{nr}(dw) \neq dw)} \end{aligned} \tag{16}$$

5 Results

For any trivial digit probability, where one symbol out of $\{-1, 0, 1\}$ has a probability of 1, and the others have of 0, the probabilities of the output symbols are either 0.0 or 1.0. If any other, non-trivial digit probability is applied to initial a_i,b_i, and the probability distribution of s_i is looped back to the $SBAC_t$ inputs a_i,b_i, the probabilities converge, see for example Figure 4, where $t = 5$ and initially $P(a_i = 0) = P(b_i = 0) = 0.1$ and $P(a_i = 1) = P(b_i = 1) = 0.9$.

Since the calculation of c_{i+1} (and indirectly d_{i+1}) depends on t, the converg also depends on t, see Figure 5.

A sample application for signed-digit arithmetic could be a CORDIC-based algorithm. CORDIC [Vol59] transforms initial data iteratively with predefined coefficients. Let A_0 be the initial N-bit data, and B_i the predefined coefficients:

$$A_{i+1} = f_{cordic}(A_i, B_i) \text{ , with } 0 \leq i \leq N - 1 \tag{17}$$

f_{cordic} is the CORDIC function for processing A_i and B_i by an adder/subtracter and shifter. A_N is the final result, A_0 the input data to be processed. To simulate the impact of subtraction, we assume a probability of 50%, that input for b_i has opposite signs. Figure 6 corresponds to this more realistic use case.

Applying the simple error model to a SBAC type 7 with a probability of 50% of $+/-$ alternation, we investigate the digit error depending on the bit flip error and the encoding, as shown in Figure 7. The error ratio is 75% for small p_{bf} and increases to expected 100% for $p_{bf} = 1$. This means by using error correction and redundant encoding for digit 0 instead of no error correction for any digit, when using SBAC type $t = 7$ in a CORDIC like arithmetic with switching sign possibility of one operand of 50%, error rate drops by 25%.

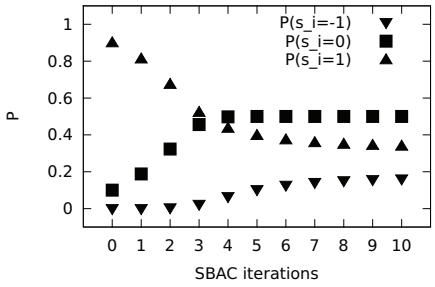


Figure 4: SBAC adder operation data probability for $t = 5$ and initial $P(a_i = 0) = P(b_i = 0) = 0.1$ and $P(a_i = 1) = P(b_i = 1) = 0.9$.

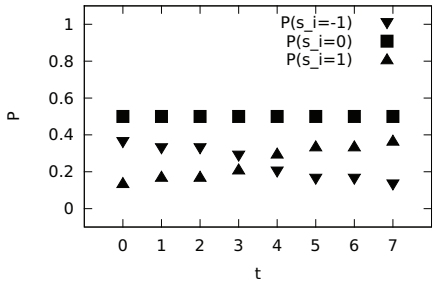


Figure 5: SBAC adder operation data probability depending on t for non-trivial initial data probabilities.

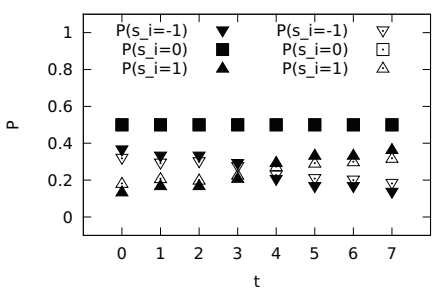


Figure 6: SBAC adder/subtracter data probability depending on type t for non-trivial initial data probabilities. Solid symbols represent Figure 5, hollow symbols represent a probability of 50% for $+/-$ alternation.

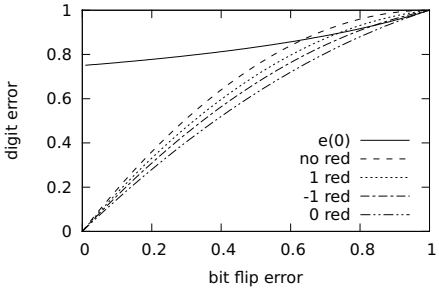


Figure 7: SBAC adder/subtracter digit error probability for $t = 7$ and non-trivial initial digit occurrence. Probability of $+/-$ alternation is 50%. "no red" denotes no redundant encoding, "-1 (0,1) red" denotes redundant encoding and error correction for -1 (0,1), "$e(0)$" denotes the error ratio of redundant encoding and error correction of digit "0", see equation 16.

6 Conclusion and future work

In a scenario, where data paths between memory and a SBAC as well as between several SBACs are fault-prone, the knowledge of digit probabilities offers a chance to use a data encoding scheme that provides some implicit fault tolerance. We have shown a model to gain data flow probability information for signed-binary based arithmetic operations and have proposed a data encoding scheme that provides advanced fault tolerance properties.

In our example, the proposed $SBAC_t$ tends to generate symmetric $P(s_i = -1)$ and $P(s_i = +1)$ probabilities with respect to type t, see again Figure 5. The actual values of $P(s_i = -1)$ and $P(s_i = +1)$ are due to the choice of $c_{i+1} \in \{-1, 0\}$ and subsequently $d_{i+1} \in \{0, +1\}$. Let us call such a design $SBAC_{-t}$.

In contrary, $SBAC_{+t}$ with $c_{i+1} \in \{0, +1\}$ and $d_{i+1} \in \{-1, 0\}$ produces the opposite probability behavior for $P(s_i = -1)$ and $P(s_i = +1)$ with respect to t. Since the probabilities $P(s_i = -1)$ and $P(s_i = +1)$ are symmetric, $SBAC_{-,t_1}$ and $SBAC_{+,t_2}$ with $t_1 + t_2 = 7$ have the same s_i digit probabilities. The digital circuit designer is free to chose the more implementation friendly design.

Still, more detailed research is needed. Changing the design from using one $SBAC_-$ (or a chain of n $SBAC_-$) to the use of alternating $SBAC_-$ and $SBAC_+$ will lead to digits with $P(s_i = -1) = P(s_i = 1)$. The possible advantages for fault tolerance and reduced (false) carry generation c_n and d_n have to be investigated.

The mentioned application, i.e. CORDIC, is not very accurately described, since the coefficients B_i are actually calculated and saved to some memory in advance. An arbitrary encoding can be chosen here to enforce a desired data probability characteristic, making the whole system even more fault-tolerant, especially when allowing more than two bits for encoding one digit.

References

[Avi61] Algirdas A. Avizienis. Signed-digit number representations for fast parallel arithmetic. *IRE Transactions on Electronic Computers*, 10(3):389–400, Sep 1961.

[COP+06] G. C. Cardarilli, M. Ottavi, S. Pontarelli, M. Re, and A. Salsano. Localization of Faults in Radix-n Signed Digit Adders. In *Proceedings of the 12th IEEE International On-Line Testing Symposium*, IOLTS, pages 178–180, Washington, DC, USA, 10–12 Jul 2006. IEEE Computer Society.

[CR78] C. Y. Chow and J. E. Robertson. Logical Design of a Redundant Binary Adder. *Proc. 4th Symposium on Computer Arithmetic*, pages 109–115, 1978.

[ITR11] ITRS. *International Technology Roadmap for Semiconductors. 2011 Edition. Emerging Research Devices*. http://www.itrs.net/links/2011itrs/2011Chapters/2011ERD.doc, 2011.

[KPKJ07] Amit Kumar, Li-Shiuan Peh, Partha Kundu, and Niraj K. Jha. Express virtual channels: towards the ideal interconnection fabric. In *Proceedings of the 34th annual international symposium on Computer architecture*, ISCA '07, pages 150–161, New York, NY, USA, 2007. ACM.

[NSF01] K. Nikolic, A. Sadek, and M. Forshaw. Architectures for Reliable Computing with Unreliable Nanodevices. In *Proc. 1st IEEE Conference on Nanotechnology*, pages 254–259, 2001.

[PMN+09] Rodolfo Pellizzoni, Patrick Meredith, Min-Young Nam, Mu Sun, Marco Caccamo, and Lui Sha. Handling mixed-criticality in SoC-based real-time embedded systems. In *Proceedings of the seventh ACM international conference on Embedded software*, EMSOFT '09, pages 235–244, New York, NY, USA, 2009. ACM.

[RSKW07] Warren Robinett, Gregory S. Snider, Philip J. Kuekes, and R. Stanley Williams. Computing with a trillion crummy components. *Commun. ACM*, 50(9):35–39, Sep 2007.

[SOHH07] M. Stahl-Offergeld, H.-P. Hohe, and M. Hackner. Spinning current offset in vertical Hall sensors caused by imperfect wiring. In *Proc. 13th International Sensor Conference*, volume 2 of *SENSOR 2007*, pages 211–216, 22–24 May 2007.

[Vol59] Jack E Volder. The CORDIC Trigonometric Computing Technique. *Ieee Transactions On Electronic Computers*, EC-8(3):330–334, 1959.

[Zeh92] Eberhard Zehendner. Reguläre parallele Addierer für redundante binäre Zahlsysteme. Technical report, Report 255, Institut für Mathematik der Universität Augsburg, Juni 1992.

Parallel coding for storage systems - An OpenMP and OpenCL capable framework

Peter Sobe
Faculty of Mathematics and Computer Engineering
Dresden University of Applied Sciences
Dresden, Germany
sobe@htw-dresden.de

Abstract: Parallel storage systems distribute data onto several devices. This allows high access bandwidth that is needed for parallel computing systems. It also improves the storage reliability, provided erasure-tolerant coding is applied and the coding is fast enough.
In this paper we assume storage systems that apply data distribution and coding in a combined way. We describe, how coding can be done parallel on multicore and GPU systems in order to keep track with the high storage access bandwidth. A framework is introduced that calculates coding equations from parameters and translates them into OpenMP- and OpenCL-based coding modules. These modules do the encoding for data that is written to the storage system, and do the decoding in case of failures of storage devices. We report on the performance of the coding modules and identify factors that influence the coding performance.

1 Introduction

Parallel and distributed storage systems are susceptible against faults due to their higher number of storage devices that all can fail or can become unaccessible temporarily. Thus, a combination with fault-tolerant coding, particularly erasure-tolerant coding is often applied. Codes are applied to calculate redundant data that is distributed in addition to the original data onto several failure-independent devices. That redundant data serves for the recalculation of 'erased' data that can not be read when devices fail or get disconnected.
There is a number of simple solutions, e.g. duplication of every data unit in a distributed system to another storage node. This introduces a high overhead in terms of storage capacity and a high write access load. Another simple solution is a parity code across all units that are distributed. The parity data is a kind of shared redundancy and can be applied to recalculate any data pice in case of a single failure. Erasure-tolerant codes are a generalization of the shared redundancy principle and are capable to tolerate a higher number of failures. Generally, codes base on a distribution of original data across k devices and a number of redundant data blocks that are placed on m additional devices (see Figure 1). It must be known which devices failed in order to decode the original data successfully.

This assumption is typically fulfilled within storage systems and differentiates the applied codes from general error-correction codes, e.g. codes for channel coding.

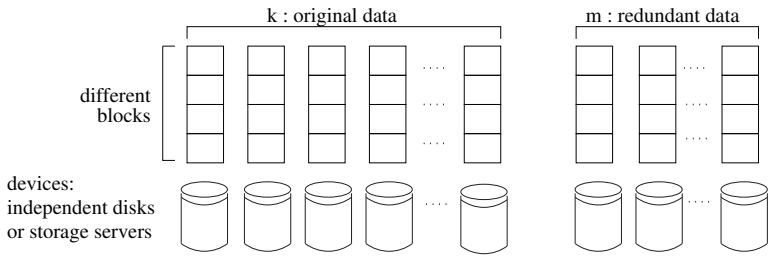


Figure 1: Data block distribution and redundancy used in parallel and distributed storage systems.

Some erasure-tolerant codes are optimal in terms of tolerated failures and storage overhead by allowing to tolerate every combination of up to m failed devices among these $k+m$ devices in total. The coding community investigated much research effort to find codes that show this optimal property for a large range of parameters k and m. Another criterion is the number of operations for encoding and decoding that should be as low as possible.
We already introduced an equation-oriented approach to erasure-tolerant coding in [SP08] that applies the Cauchy Reed/Solomon code arithmetics. Equations that calculate redundant data units by XORing original data units in an appropriate way define the functionality of the storage system. Initially, we provided these equations in data files in order to parameterize the en- and decoder of the storage system. The contribution of this paper is a proof of the concept that equations can be translated into programming language code directly. This code is enriched with expressions that control parallel processing, either in terms of data-parallel OpenCL kernel code, or in terms of OpenMP directives. These expressions are generated automatically.
The paper is organized as follows. Related work is surveyed in Section 2. The principle of equation-oriented en- end decoding is explained in Section 3 and in Section 4 we describe the translation to OpenCL and OpenMP code. A performance evaluation of our implementation can be found in Section 5. We conclude with a summary.

2 Related Work

Parallel storage systems that employ several storage devices and coding for fault tolerance first have been introduced with RAID systems [KGP89] in the context of several host-attached disks. This general idea later got adopted to networked storage. Later a variety of different codes were explored and applied for different

types of systems, e.g. networked storage, distributed memory systems or memories for sensor networks.

The Reed/Solomon code [IR60] (R/S) is a very flexible code that allows to construct coding systems for different distribution factors (k) and different amount of redundant data (m). R/S provides specific coding and decoding rules for any k and m, following a linear equation system approach. Originally, R/S requires Galois Field arithmetics and therefore needs more instructions and processing time on general purpose processors, compared to XOR-based codes that can directly use the processors XOR instruction. An XOR-based variant of R/S was introduced by Blomer et al. [BKK+95] and got later known as the so called Cauchy-Reed/Solomon code (CRS). This code divides each of the $k+m$ storage resources into ω different units (ω is chosen such that $2^\omega > k+m$ holds) that are individually referenced by XOR-based calculations. In our previous work on the NetRAID [Sob03, SP06] system an equation-based description of encoding and decoding was developed and allows a flexible use of different codes.

Equation-based coding strongly relates to the matrix-based coding technique that is supported by the jerasure library for erasure-tolerant codes [Pla07]. A binary code generator matrix selects Bits of the original data word to be XORed to the redundant Bits. Optimizations of the encoding algorithms and the creation of decoding algorithms are a result of matrix operations. The main objective is to find efficient codes with optimal failure-correction capabilities and minimal computation cost. In our tools we apply matrix-based techniques as well, but provide a textual description of coding algorithms that consists of equations over different Bits.

In an environment with parallel processes and parallel storage devices, it is necessary to exploit parallelism as well for storage coding to reach reasonable high coding throughput that keeps track with the desired high speed of the storage system. To use multi core processors is obvious. In addition, R/S and CRS have been offloaded to FPGA [HKS+02],[HSM08], GPU using NVidia CUDA [CSWB08] and other hardware [SPB10]. In [CSWB08] a GPU was evaluated for encoding a k=3, m=3 R/S code. It could be shown that the GPU's encoding rate is higher than the RAID level 0 aggregated write rate to the disks and coding keeps track with the pure disk system performance. The wide availability of multicore processors and OpenMP (Open Multi Processor) motivated further steps to run the coder as a multithreaded system.

Besides data parallelism as a straightforward way, further functional parallelism can be exploited in storage system coding. The functional parallelism is represented by the different equations for different redundant data units. For CRS, a number of $\omega \cdot m$ different redundant units can be calculated independently using individual XOR calculations which allows equation-based functional parallelism. A comparison between equation-oriented coding and data-parallel coding in [Sob10] revealed that equation-parallel coding improves the locality of data access for input and output data. Nevertheless, equation-oriented parallelism does not always produce an evenly balanced workload and requires a special choice of parameters to create evenly distributed encode equations.

3 Coding by Equations

The concept to describe encoding and decoding by XOR equations has been introduced in [SP08]. The equations are provided by a tool that includes all the CRS arithmetics and delivers the equation set for a storage system.

The naming of the units and the placement of the units on the storage resources is defined as follows. We place units $0, 1, \ldots \omega$-1 consecutively on the first original storage device, units ω to $2 \cdot \omega$-1 on the second device and so on. Each unit is denoted by the character 'u' and a number, e.g. $u0$ for the first unit in the system. The code calculations have to reference these units properly in the XOR equations. For the example with $k = 5$ and $m = 2$, the number of equations is 6. There is an individual equation for each of the 6 units. These 6 units are placed on two redundant storage devices (see Listing 1).

```
u15 = XOR(u2,u3,u4,u5,u7,u9,u11,u12)
u16 = XOR(u0,u2,u3,u7,u8,u9,u10,u11,u13)
u17 = XOR(u1,u3,u4,u6,u8,u10,u11,u14)
u18 = XOR(u0,u2,u4,u6,u7,u8,u11,u12,u13)
u19 = XOR(u0,u1,u2,u4,u5,u6,u9,u11,u14)
u20 = XOR(u1,u2,u3,u5,u6,u7,u10,u12)
```

Listing 1: Example for a coding scheme ($k = 5$, $m = 2$, $\omega = 3$).

The equations above allow to calculate every redundant unit independently from the other ones. Such a coding naively supports parallel processing, but contains redundant calculations, e.g. $XOR(u2, u3)$ is calculated 3 times. We call this the *direct coding style*. Another style of coding is called the *iterative coding style* that exploits previously calculated elements when possible. In that way, redundant calculations can be eliminated, e.g. $XOR(u2, u3)$ is stored in a temporary unit $t0$ and then referenced 3 times. Replacing all common subexpressions reduces significantly the number of XOR operations. For the $k = 5$, $m = 2$ system a reduction from 45 to 33 XOR operations occurred. For this example, the equations are given in Listing 2 with temporary units denoted with 't' and their number. The iterative equations can be formed from the equations given in the direct style using an automated preprocessing step.

Our approach is to translate the equations in a further processing step directly to OpenCL kernel code, or alternatively to OpenMP code. Both variants use extension of the C programming language. The generated code can be compiled to storage system components during system runtime. Particularly, at the time when a new failure situation is detected, the framework calculates new decoding equations to recalculate the missing data units from the other ones that are still available. A new decoder code can be generated from the decoding equations, translated to C code with parallel OpenCL or OpenMP extensions and then compiled to runtime components of the system.

```
u15 = XOR(t1,t3,t4)
u16 = XOR(t4,t6,t7)
u17 = XOR(u3,u4,u8,t6,t9)
u18 = XOR(u2,u4,u6,u7,t3,t7)
u19 = XOR(u0,u2,u9,u11,t1,t9)
u20 = XOR(u5,u7,u10,u12,t0,t8)
t0 = XOR(u2,u3)
t1 = XOR(u4,u5)
t2 = XOR(u7,u9)
t3 = XOR(u11,u12)
t4 = XOR(t0,t2)
t5 = XOR(u0,u8)
t6 = XOR(u10,u11)
t7 = XOR(u13,t5)
t8 = XOR(u1,u6)
t9 = XOR(u14,t8)
```

Listing 2: Coding scheme ($k = 5$, $m = 2$, $\omega = 3$)

4 Translation to parallel code

Our tool that generates the equations is capable to generate OpenCL kernel code and OpenMP program code as well. To do that, the user solely has to specify a few optional parameters, e.g. the file for code output and the unit length that is needed for addressing within the data arrays. This can be seen in the following command line of the tool:

```
./cauchyrs -k=5 -m=2 --ocl_encoder --ocl_file=crs5+2.cl
                     --ocl_unit_len=2048 --ocl_encstyle=iterative
```

The OpenCL code, generated from a CRS code with $k = 5$, $m = 2$ is listed in Listing 3. The unit numbers got translated into index values in order to address the data that relate to the unit. For instance $u0$ got translated to $n[0 + i]$, $u4$ to $n[8192 + i]$ and $u15$ to $r[0 + i]$ by comparing the OpenCL code with the equations given in Listing 2. The constant offset in the index is calculated from the unit number and the length of the units, e.g the 5th unit with the index 4 points to 4×2048, when 2048 is the unit length. The redundant units with numbers $n : k \times \omega \leq n < (k+m) \times \omega$ are translated into elements within the r array (r denotes redundant data). The constant part is derived by $((n-k) \times \omega) \times$ unitlength. Every index contains a variable part i that addresses each individual byte in the unit. The XOR (operator symbol ^) causes that corresponding bytes of the different units in the C statements are XORed bitwise. Finally, a XOR operation on corresponding bits within the units takes place. A processing along different i values is done by the GPU that invokes the kernel code when the function

```
clEnqueueNDRangeKernel(...,ckKernel,1,NULL,&GlobalSize,&LocalSize,...)
```

```
// Parameters: k=5, m=2, w=3, OCL_UNIT_LEN=2048
__kernel void crs(__global const char *n, __global char *r)
{
  unsigned int i = get_global_id(0);
  char t21 = n[4096+i]^ n[6144+i];
  char t22 = n[8192+i]^ n[10240+i];
  char t23 = n[14336+i]^ n[18432+i];
  char t24 = n[22528+i]^ n[24576+i];
  char t25 = t21^ t23;
  char t26 = n[0+i]^ n[16384+i];
  char t27 = n[20480+i]^ n[22528+i];
  char t28 = n[26624+i]^ t26;
  char t29 = n[2048+i]^ n[12288+i];
  char t30 = n[28672+i]^ t29;

  r[0+i] = t22^ t24^ t25;
  r[2048+i] = t25^ t27^ t28;
  r[4096+i] = n[6144+i]^ n[8192+i]^ n[16384+i]^ t27^ t30;
  r[6144+i] = n[4096+i]^ n[8192+i]^ n[12288+i]^ n[14336+i]^ t24^ t28;
  r[8192+i] = n[0+i]^ n[4096+i]^ n[18432+i]^ n[22528+i]^ t22^ t30;
  r[10240+i] = n[10240+i]^ n[14336+i]^ n[20480+i]^ n[24576+i]^ t21^ t29;
}

// another kernel that works with a word len of 16 bytes
// and reaches slightly better performance
__kernel void crs16(__global const char16 *n, __global char16 *r)
```

Listing 3: Automatically generated OpenCL kernel.

is called by the host program. A number of *LocalSize* threads are started on the GPU multiprocessors and are supported by the SIMD-like data parallel execution technique. The GPUs used, allowed to start 512 threads (NVidia Quadro FX880M) and 1024 threads (NVidia Quadro 600). The parameter *GlobalSize* can express a higher thread number, that are run in a batch mode in groups of *LocalSize* threads.

In the same style like the OpenCL code generation we support OpenMP (Open Multi Processor language) programs for coding. OpenMP allows to create multithreaded processes from a sequential code by adding directives to the program. Typically, the workload of for-loops is distributed to several threads. OpenMP threads run on a shared memory and do not need to transfer any data before and after the multithreaded execution.
In the example, a C-program is written to the file omp5+2.c.

```
./cauchyrs -k=5 -m=2 --omp_encoder --omp_file=omp5+2.c
                     --omp_unit_len=2048 --omp_encstyle=iterative
```

The C program (see Listing 4) contains array index values instead of unit numbers. For OpenCL we generated macro code for the index. This allows to read the code like equations and find the unit numbers. The C preprocessor replaces the macro symbols with the macro expressions. At compile time, the constant part

of each index is calculated. The variable part i of an index is controlled by the for-loop during the runtime of the encoder. Where a common C program would run all the iterations from i=0 to i=unitlength-1, OpenMP delegates the loop to several threads that cooperatively run different index ranges. The OpenMP directive (#pragma omp ...) is generated automatically in the same way as the program code.
Variables that are in local use for each iteration have to be declared as private. For our application this applies to the character variables for the temporary units.

```
// Parameters: k=5, m=2, w=3, OCL_UNIT_LEN=2048
#define UNIT_LEN 2048
#define RUNIT(a) a*UNIT_LEN+i
#define OUNIT(a) a*UNIT_LEN+i

void calc(const char *n, char *r)
{
    int i;
    char t21, t22, t23, t24, t25, t26, t27, t28, t29, t30;

    #pragma omp parallel for private(t21, t22, t23, t24, t25, t26, t27,
        t28, t29, t30)
    for (i = 0; i < UNIT_LEN; i++)
    {
        t21 = n[OUNIT(2)]^ n[OUNIT(3)];
        t22 = n[OUNIT(4)]^ n[OUNIT(5)];
        t23 = n[OUNIT(7)]^ n[OUNIT(9)];
        t24 = n[OUNIT(11)]^ n[OUNIT(12)];
        t25 = t21^ t23;
        t26 = n[OUNIT(0)]^ n[OUNIT(8)];
        t27 = n[OUNIT(10)]^ n[OUNIT(11)];
        t28 = n[OUNIT(13)]^ t26;
        t29 = n[OUNIT(1)]^ n[OUNIT(6)];
        t30 = n[OUNIT(14)]^ t29;

        r[RUNIT(0)] = t22^ t24^ t25;
        r[RUNIT(1)] = t25^ t27^ t28;
        r[RUNIT(2)] = n[OUNIT(3)]^ n[OUNIT(4)]^ n[OUNIT(8)]^ t27^ t30;
        r[RUNIT(3)] = n[OUNIT(2)]^ n[OUNIT(4)]^ n[OUNIT(6)]^ n[OUNIT(7)]^
            t24^ t28;
        r[RUNIT(4)] = n[OUNIT(0)]^ n[OUNIT(2)]^ n[OUNIT(9)]^ n[OUNIT(11)]^
             t22^ t30;
        r[RUNIT(5)] = n[OUNIT(5)]^ n[OUNIT(7)]^ n[OUNIT(10)]^
                        n[OUNIT(12)]^ t21^ t29;
    }
}
```

Listing 4: Automatically generated OpenMP program.

5 Performance Evaluation

OpenCL always uses 'just in time' compilation of the GPU code. This means that during the operation of the storage system processes a new kernel code can be compiled and executed. Typically, when the storage system processes are started, the encoding algorithm is compiled into the run time components once. Later on, for encoding the data is transferred to the GPU, the kernel is invoked and the data is moved back to the host memory. An OpenCL process runs through the following phases:
(1) platform exploration and connecting to the GPU, (2) buffer management i.e. allocating GPU memory, (3) kernel compilation, (4) input data transfer, (6) kernel invocation and (7) result data transfer. We measured the time of these phases by a host program (see Listing 5).

```
OpenCL  K=3 M=2 w=3 UNIT_LEN=7168 wordlen=16
0.088277 seconds for platform exploration
0.000014 seconds for buffer management
0.189414 seconds for kernel compilation
0.000048 seconds for data transfer
0.000573 seconds for kernel execution and result return
Data rate incl. transfer: 99.058733 MiB/s
Data rate w/o. transfer: 107.341098 MiB/s
```

Listing 5: Example for an OpenCL kernel execution and the measured times.

The times measured for OpenCL phases of different runs are depicted in Figure 2 for a small system (code parameters $k = 1$, $m = 1$, $\omega = 1$, unitlen= 16 byte) that moves 1×16 Byte to the GPU, copies the 16 Bytes to the array of redundant bytes and moves $1\times$ 16 Byte back to he host memory. The comparison for a system with a higher distribution factor, more redundancy and a larger block size is given in Figure 3 (code parameters $k = 4$, $m = 2$, ω=3, unitlen= 32kByte). This coding scenario transfers $4 \times 3 \times 32kiB = 384kiB$ to the GPU memory and $2 \times 3 \times 32kiB = 192kiB$ The kernel executes 33 XOR operations for every 12 Bytes input and 8 Bytes output data. These are 1081344 Byte XOR operations in total. Both measurements were taken on a NVidia Quadro FX880M. The individual times for specific phases show that a bigger equation system causes a longer kernel compilation time. The other time values are dependent from the size of processed data.

A direct comparison of sequential processing, OpenMP processing with 4 threads and OpenCL processing on two different GPUs (GPU-1: NVidia Quadro FX 880M, GPU-2: NVidia Quadro 600) is given by the data throughput rates in Table 1. The execution times for sequential and OpenMP processing were measured on two different processors (CPU-1: Intel Core i5 M520, 2.4 GHz, CPU-2: AMD Phenom II X4, 840, 3.2 GHz). We calculated the redundancy for a $k = 4$, $m = 2$, $\omega = 3$ code with a unit length of 32 kiB and 64 kiB. The values are average times taken from 10 runs. The measurements for the direct encoding style were done with equations

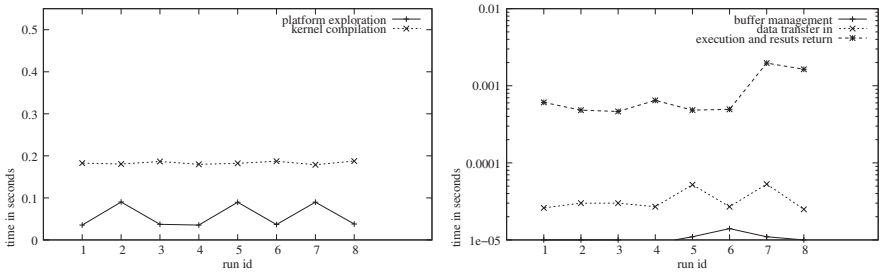


Figure 2: Minimal Code: Time consumption of phases for OpenCL processing on a GPU.

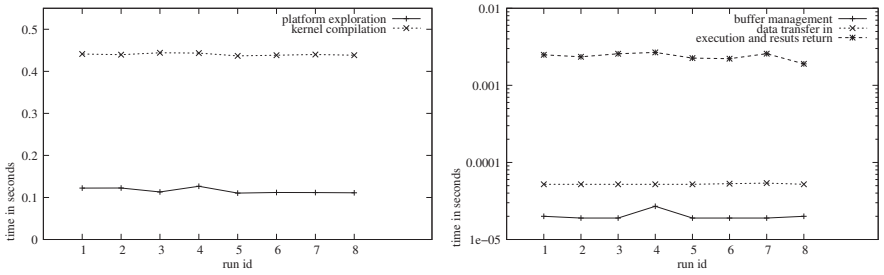


Figure 3: Bigger Code: Time consumption of phases for OpenCL processing on a GPU.

that still contained redundant calculations and are clearly disadvantageous for sequential processing.

OpenMP shows a moderate performance improvement. CPU-1 is a dual core processor that supports 4 threads. The measured speedup on that dual core system is 1.6 and 1.8. CPU-2 is a real 4-core system and the speedup factor is approximately 4 on that system. Besides of the different speedup, both CPUs reach nearly the same absolute performance when OpenMP processing is applied. This can be explained with the bigger cache of CPU-1 that improves the performance of each thread for this data-intense coding application.

The GPU performance is significantly better than sequential processing on the CPU, and better than multithreaded execution on the CPUs as well. However, it does not reach the theoretical performance of the GPU by far. This is caused by the data transfer between the host memory and the GPU memory.

When doing coding with a higher distribution factor and more redundant devices, the computational cost increases. Accordingly, the ratio between transferred data and computations is moved in direction of a bigger computational part. For sequential processing the data rates sink due to the higher computation cost. For GPU computing the disadvantageous cost of data transfer is assumed to diminish with increasing distribution and redundancy factors, due to the higher computational load that can be coped with by the highly multithreaded architecture.

unit length	encoding style	CPU-1		CPU-2		GPU-1	GPU-2
		sequential	OpenMP	sequential	OpenMP	OpenCL	
32 kiB	direct	124.3	189.7 (× 1.5)	44.9	179.3 (× 4)	224.4	299.9
	iterative	168.9	269.8 (× 1.6)	72.0	256.1 (× 3.5)	222.2	407.1
64 kiB	direct	111.3	181.2 (× 1.6)	54.6	181.4 (× 3.3)	273.2	318.6
	iterative	252.6	464.1 (× 1.8)	70.7	257.3 (× 3.6)	263.6	410.3

Table 1: Data throughput of encoding on different platforms in MiB/s.

6 Summary

OpenCL and OpenMP are appropriate platforms to implement software-based erasure-tolerant coding for storage systems. Because the erasure-tolerant codes strictly follow mathematical principles, particularly linear equation systems in case of the Cauchy Reed/Solomon code, the kernels of the coding programs can be generated in an automated way. We showed that an equation-oriented description of the codes can be easily translated to OpenCL and to OpenMP code. Fortunately, all the expressions to control parallel execution could be generated automatically as well.

OpenCL supports just in time compilation of GPU code which can be applied for a storage system to exchange coding modules during runtime. This is needed to insert new decoding algorithms in case of failures. The decoding algorithm adapts to the specific failure situation without requiring to run through control flow instructions. Because OpenCL is capable to run code on several platforms, especially on a multicore CPU device as well, it is a preferable platform compared to OpenMP. The performance evaluation revealed a moderate speedup for GPU processing using OpenCL and for multicore processing using OpenMP. We expect that GPU computing using OpenCL can reach to I/O bound (transfer bandwidth to and from GPU via the system interface) when all optimizations are applied.

References

[BKK+95] J. Blomer, M. Kalfane, M. Karpinski, R. Karp, M. Luby, and D. Zuckerman. An XOR-based Erasure–resilient Coding Scheme. Technical Report TR-95-048, International Computer Science Institute, August 1995.

[CSWB08] M. L. Curry, A. Skejellum, H. L. Ward, and R. Brightwell. Acellerating Reed-Solomon Coding in RAID Systems with GPUs. In *Proceedings of the 22nd IEEE Int. Parallel and Distributed Processing Symposium*. IEEE Computer Society, 2008.

[HKS+02] A. Haase, C. Kretzschmar, R. Siegmund, D. Müller, J. Schneider, M. Boden, and M. Langer. Design of a Reed Solomon Decoder Using Partial Dynamic Reconfiguration of Xilinx Virtex FPGSs - A Case Study. 2002.

[HSM08] V. Hampel, P. Sobe, and E. Maehle. Experiences with a FPGA-based Reed/Solomon-encoding coprocessor. *Microprocessors and Microsystems*, 32(5-6):313–320, August 2008.

[IR60] G. Solomon I. Reed. Polynomial Codes over Certain Finite Fields. *Journal of the Society for Industrial and Applied Mathematics [SIAM J.]*, 8:300–304, 1960.

[KGP89] R. Katz, G. Gibson, and D. Patterson. Disk System Architectures for High Performance Computing. In *Proceedings of the IEEE*, pages 1842–1858. IEEE Computer Society, December 1989.

[Pla07] J. S. Plank. Jerasure: A Library in C/C++ Fasciliating Erasure Coding to Storage Applications. Technical Report CS-07-603, University of Tennessee, September 2007.

[Sob03] P. Sobe. Data Consistent Up- and Downstreaming in a Distributed Storage System. In *Proc. of Int. Workshop on Storage Network Architecture and Parallel I/Os*, pages 19–26. IEEE Computer Society, 2003.

[Sob10] P. Sobe. Parallel Reed/Solomon Coding on Multicore Processors. In *6th IEEE International Workshop on Storage Network Architecture and Parallel I/Os*. IEEE Computer Society, 2010.

[SP06] P. Sobe and K. Peter. Comparison of Redundancy Schemes for Distributed Storage Systems. In *5th IEEE International Symposium on Network Computing and Applications*, pages 196–203. IEEE Computer Society, 2006.

[SP08] P. Sobe and K. Peter. Flexible Parameterization of XOR based Codes for Distributed Storage. In *7th IEEE International Symposium on Network Computing and Applications*. IEEE Computer Society, 2008.

[SPB10] T. Steinke, K. Peter, and S. Borchert. Efficiency Considerations of Cauchy Reed-Solomon Implementations on Accelerator and Multi-Core Platforms. In *Symposium on Application Accelerators in High Performance Computing 2010*, Knoxville, TN, July 2010.

Parallelization Strategies to Speed-Up Computations for Terrain Analysis on Multi-Core Processors

Steffen Schiele[1], Holger Blaar[1], Detlef Thürkow[2], Markus Möller[2], Matthias Müller-Hanneman[1]

[1]University of Halle-Wittenberg
Institute of Computer Science
Von-Seckendorff-Platz 1
06120 Halle (Saale)

[2]University of Halle-Wittenberg
Institute of Geosciences
Von-Seckendorff-Platz 4
06120 Halle (Saale)

steffen.schiele@informatik.uni-halle.de

Abstract: Efficient computation of regional land-surface parameters for large-scale digital elevation models becomes more and more important, in particular for web-based applications. This paper studies the possibilities of decreasing computing time for such tasks by parallel processing using multi-threads on multi-core processors. As an example of calculations of regional land-surface parameters we investigate the computation of flow directions and propose a modified D8 algorithm using an extended neighborhood. In this paper, we discuss two parallelization strategies, one based on a spatial decomposition, the other based on a two-phase approach. Three datasets of high resolution digital elevation models with different geomorphological types of landscapes are used in our evaluation. While local surface parameters allow for an almost ideal speed-up, the situation is different for the calculation of non-local parameters due to data dependencies. Nevertheless, still a significant decrease of computation time has been achieved. A task pool-based strategy turns out to be more efficient for calculations on datasets with many data dependencies.

1 Introduction

A large variety of methods for space-oriented analyses of the earth's surface have been developed in the past couple of years. Methods based on photogrammetry and laserscanning are able to produce digital elevation models (DEMs) with a geometric resolution within centimeters and a high quality, as well as a high quantity of data. Algorithms have to be developed enabling computational efficient processing of large datasets in reasonable time [Woo09]. Therefore, new strategies for efficient implementation and parallelization of such computations are needed.

The attribute 'flow direction' is the basis for the calculation of the most popular hydrological parameters like 'specific catchment area' or 'topographic wetness index' [ZKLY07, GP09]. The typical workflow for the computation for these parameters is sketched in Figure 1. First, the raw data of the DEM is preprocessed to remove artefacts, systematic errors, and to reduce noise. It is also important to eliminate spurious sinks by filling [RHGS09]. Afterwards, the flow directions are determined. The computation of catchment area and

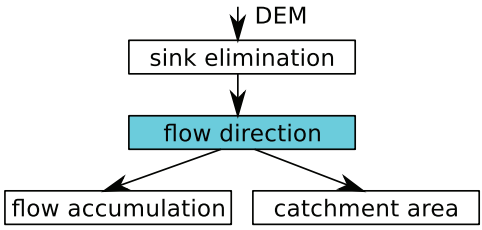


Figure 1: Sketch of the typical workflow for hydrological parameter calculations.

flow accumulation base on the flow direction. Computationally, these steps are less intensive.

Although the sequential running time for test sites with 10^8 grid cells is in the range of a few minutes on standard desktop machines, further reductions of processing times are still of crucial importance for the acceptance of web-based hydrological applications [ASMB03, GTDS10]. Our intention is to investigate how parallel processing using multi-threads can help to decrease computing time of flow direction's calculation. For this purpose, we have developed two parallelization strategies and tested them on a modified single-flow algorithm (Section 2.1). In this paper, we restrict our discussion of parallelization strategies to this single-flow algorithm, although we have implemented parallel algorithms for the remaining steps of the workflow, too [Sch10].

A classical and most basic algorithm to determine flow directions is the so-called D8-algorithm [OM84]. From each grid cell, all flow is passed to the neighbor in direction of the steepest descent. Except for cells at the boundary of the DEM, a cell can be viewed as the center cell of a 3×3 subgrid. The 8 non-central cells of such a subgrid are considered as neighbors, hence the name D8. The crucial point in this method is how ambiguous flow directions are resolved, when the same minimum down-slope gradient is found for several neighbor cells. We would like to emphasize that such ambiguities are not an academic consideration, they occur quite often in practice. To resolve such ambiguities, we have developed an extended neighborhood approach. Extended neighborhoods pose a particular challenge for parallelization due to non-local memory access patterns.

Related work Based on the model of SIMD (single instruction stream, multiple data stream) computers, Mower [Mow94] discusses data-parallel procedures for drainage basin analysis. More recent work on multi-core machines uses the OpenMP library or MPI. For example, Neal et al. [NFT09] describe and report experience with parallelization of procedures for flood inundation models using the OpenMP interface. Building on the message passing interface MPI, Tesfa et al. [TTW+11] developed parallel approaches for the extraction of hydrological proximity measures. In contrast to the single flow direction method studied in this paper, they use the multiple flow direction model D-infinity. Instead of using multi-core processors, another interesting approach for parallelization is the usage of GPUs. Ortega and Rueda [OR10] have studied the applicability of this approach for parallel computation of drainage networks using the CUDA framework.

To cope with large-scale high-resolution datasets, Mølhave et al. [MAAR10] developed I/O-efficient external memory algorithms. The focus of our paper, however, is on algorithms which can be handled within internal memory.

Overview In Section 2, we first sketch our extended D8 algorithm for flow computations and then explain our two parallelization strategies. We also describe the sites used in our experimental study. Computational results for both parallelization strategies are given in Section 3. Finally, we summarize and discuss our observations in Section 4.

2 Methods

2.1 Extended D8 algorithm for flow computations

To avoid ambiguous flow directions, we introduce a modified algorithm D8*e* which recursively extends the neighborhood in such cases until a unique single flow direction is found. For a given cell c and its neighborhood $N(c)$ let $S(c) \subseteq N(c)$ be the subset of neighbors which realize the steepest descent. Thus, $S(c)$ forms the candidate set for the flow direction of c. If $|S(c)| = 1$, the flow direction is unique and we are done. Otherwise, we determine the extended neighborhood $EN(S(c))$ as the set of all cells which are connected to some cell $\bar{c} \in S(c)$ by a path of cells with the same altitude as $\bar{c}$. Then we compute recursively the flow direction of all cells within $EN(S(c))$. Among all considered cells, we take again the steepest descent. If this value is unique, we can now assign the flow direction of cell c as the one which leads along a path of assigned flow directions to the cell of steepest descent. Otherwise, the procedure has to be continued in the same manner until the ambiguities are resolved or no further neighborhood extension is possible. In the latter case, an arbitrary decision for cell c is made.

The neighborhood extension is the most time-consuming part of the computation of flow directions. The average size of the extended neighborhood varies widely depending on the terrain. An example of the effect of ambiguous flow directions is given in Figure 2. The figure shows two catchment areas, one computed with the D8 and one with the D8*e* algorithm. The first one misses a significant part of the catchment area.

2.2 Parallelization strategies

We investigate two main strategies for parallelization. The first approach divides the DEM into squares and the second one divides the computation of the flow directions into two phases. All threads access the same DEM stored in shared memory. Therefore, there is no need to transfer data but the data access has to be synchronized among the threads.

Dividing the DEM into squares If the grid domain of the DEM is partitioned into a number of disjoint squares and the flow directions are computed concurrently by different threads complications often arise. Namely, the extended neighborhood computation can include cells of other squares. This means that the same flow direction of a single cell might be calculated several times and the number of such cells could be prohibitively high.

To avoid such problems a pre-computation is executed. At first, the algorithm chooses squares which are slightly overlapping at common boundary cells. Then the flow direction of all boundary cells and their neighbors is computed in a sequential step. In doing so, no thread will cross its square boundary in the upcoming parallelized computations. Figure 3 exemplarily illustrates two cases where the algorithm extends the neighborhood of the cell with altitude 7 and precalculates the flow directions of all cells of the extended neighborhood (cells with altitude 5). Finally, the results of the sequential pre-computation are accessible by each thread. Afterwards, each thread computes the flow direction of the remaining cells.

Dividing the computation into two phases The main idea of this approach is to compute the flow direction using the original D8 algorithm during a first phase. Instead of extending the neighborhood of cells with ambiguous flow directions (see Section 2.1) such cells are only marked. During the second phase, our algorithm extends the neighborhood of each marked cell and computes their flow directions.

The first phase can be parallelized by dividing the DEM into squares. Only using the 3×3-neighborhood of a cell for computing the flow direction, no pre-computation is needed. In the second phase, the marked cells are assigned to different groups in such a way that redundant calculations are avoided. At first, a marked cell is assigned to an empty group. Afterwards, a cell will be assigned to this group, if

- the marked cell is adjacent to one cell of the group, or
- there is at most one unmarked cell between the marked cell and a cell of the group.

The group assignment is done with breadth-first search.

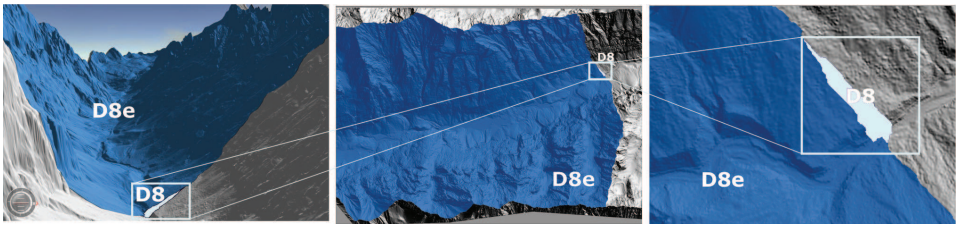
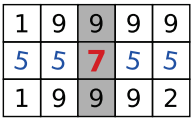
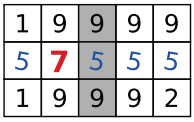


Figure 2: Comparison of catchment areas based on the flow directions computed using the D8 and D8*e* algorithm. This example shows the dramatic difference between the traditional D8 algorithm (which leads to a way too small catchment area) and our new D8*e* algorithm resulting in a catchment area confirmed by experts.



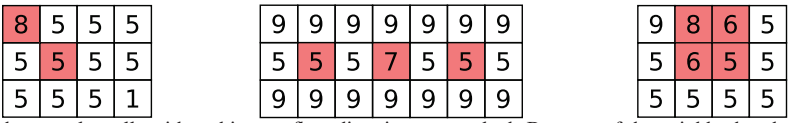
(a)



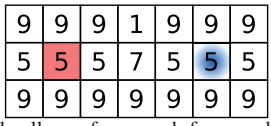
(b)

Figure 3: Excerpts of a DEM where two squares overlap in the gray-shaded boundary cells. Numbers in each cell of the DEMs correspond to the elevation. In both examples, the D8*e* algorithm extends the neighborhood of the cell with altitude 7 and precalculates the flow directions of all cells with altitude 5.

Figure 4(a) shows marked cells for which the computation requires a neighborhood extension. In each of the three examples the algorithm puts the marked cells into the same group. The differently marked cells in the example of Figure 4(b) belong to two different groups. The extended neighborhoods of both cells are disjoint. One thread is responsible for assigning cells to groups and manages these groups by a task pool [RR10]. Pseudocode is given in Algorithm 1. Each of the other threads takes a group out of the task pool and computes the flow direction of the marked cells of this group. In doing so, this approach ensures that two threads will never compute the flow direction of the same cell and that the computed extended neighborhoods will never overlap.



(a) In each example, cells with ambiguous flow direction are marked. Because of the neighborhood extension, the marked cells will be computed by the same thread. Therefore, our algorithm puts them into the same group.



(b) In this example, differently marked cells are far enough from each other so that their extended neighborhoods are disjoint. Thus, our algorithm puts them into different groups.

Figure 4: Parallelization of the D8*e* algorithm by dividing the computation into two phases. Numbers in each cell correspond to the elevation in the DEM.

2.3 Efficiency measurement

For measuring the runtime of a parallel implementation, the *real time* (wall clock time) can be used, but it can be influenced by other applications running on the system. *User* and *system CPU time* of a parallel application is the *accumulated* user and system CPU time on all processors. Because of the disruptive effect of other processes running on the system the number of cores used by one job cannot easily be determined and can also vary during program execution. We used an almost unloaded system for real-time measurements.

Algorithm 1 The procedure determines groups of cells with ambiguous flow direction (see Figure 4(a)) and puts each group of such cells into the task pool. If all cells with ambiguous flow direction are grouped a signal is send to all threads.

```
1:  procedure DETERMINE GROUPS OF CELLS
2:      for each cell ∈ DEM ∧ cell.flowdirection == 0 do
3:          create newGroup
4:          newGroup ← {cell}
5:          for each nCell in 4 × 4 neighborhood of a cell ∈ newGroup do
6:              if nCell.flowdirection == 0 then
7:                  newGroup ← newGroup ∪ {nCell}
8:              end if
9:          end for
10:         put newGroup into the taskpool
11:     end for
12:     send signal termination to all threads
13: end procedure
```

2.4 Study sites and datasets

The computational experiments are executed on three different DEMs with high spatial resolutions (Table 1). All DEMs are based on airborne laser-scanning which are cleaned from vegetation and artificial objects like buildings. The datasets represent three geomorphological types of landscapes in Central Europe: high mountains (the Alps - Reintal; see Figure 2), low mountain ranges (the Ore Mountains - Saidenbachtal) and floodplains of the lowlands (Floodplain of the River Mulde). In all DEMs sinks were removed by filling within a preprocessing step (see [RHGS09]). Table 1 shows the meta data of the used DEMs.

Table 1: Meta data parameters of the DEM datasets

DEM dataset	**Cell number**	**Columns and rows**	**Spatial resolution** $[m^2]$	**Filesize**[a] [Mb]
Reintal	37,734,557	10717×3521	1×1	290
Saidenbachtal	35,000,000	7000×5000	2×2	240
Mulde	116,674,076	6661×17516	1×1	880

[a]ascii-grid format (*.asc)

3 Results

The following runtime measurements were done on a symmetric multiprocessing computer (two Intel(R) Xeon(R) CPUs with four cores and 2.93 GHz each, and with 47 GB main memory). Additional runtime measurements were executed on different hardware, like a Linux Server with four AMD Opteron™ processors 852 (1000 MHz) and 16 GB main memory, or a symmetric multiprocessing cluster (18 computation nodes with 16 CPU cores each and with at least 32 GB main memory each). In all cases the computations require up to 2 GB main memory. Computations on the symmetric multiprocessing cluster were executed only on one node. On the available architectures speed-up and efficiency did not show significant differences. The algorithms are implemented in C++ and the Pthread library is used for parallelization, using compiler `g++` in version 4.4.3.

3.1 Dividing the DEM into squares

The sequential runtimes for the three DEMs are 3.4s for DEM 1 (Reintal), 6.7s for DEM 2 (Saidenbachtal) and 122s for DEM 3 (Mulde). The obtained speed-ups for up to four threads are shown in Table 2. The number of threads is displayed in the form of „$a \cdot b$“ where a is the number of rowwise partitions, and b the number of columnwise partitions. In Figure 5 the runtime (real time) is compared to the ideal runtime. The stacked bars show

Table 2: Speed-ups of the parallel algorithm which divides the DEM into squares.

number of threads:	$\mathbf{1 \cdot 1}$	$\mathbf{1 \cdot 2}$	$\mathbf{2 \cdot 1}$	$\mathbf{1 \cdot 3}$	$\mathbf{3 \cdot 1}$	$\mathbf{2 \cdot 2}$
speed-up (DEM 1 “Reintal”):	0.99	1.80	1.78	2.33	2.23	3.23
speed-up (DEM 2 “Saidenbachtal”):	0.97	1.59	1.63	2.05	2.02	2.40
speed-up (DEM 3 “Mulde”):	0.98	1.53	1.55	1.46	1.81	1.82

the sequential computing part and the cumulative runtime of threads. Added together they are a measure for the cost of computation. The cost of the computation is also directly obtained by measuring the CPU time used. Both clarify that rising the number of threads tends to result in an increase of used CPU time. But the effects differ between different runs with the same number of threads. The more threads we use, the more likely it is that the pre-computation steps are computationally more intensive. Note that by chance the boundary cells of the squares may have (almost) with unambiguous flow directions. If so, the pre-computation step is computationally less intensive as we can see in the case 2×2 in Figure 5. The load balance depends in particular on the topology and on the partition of the DEM. In most but not all cases we have a suboptimal load balance.

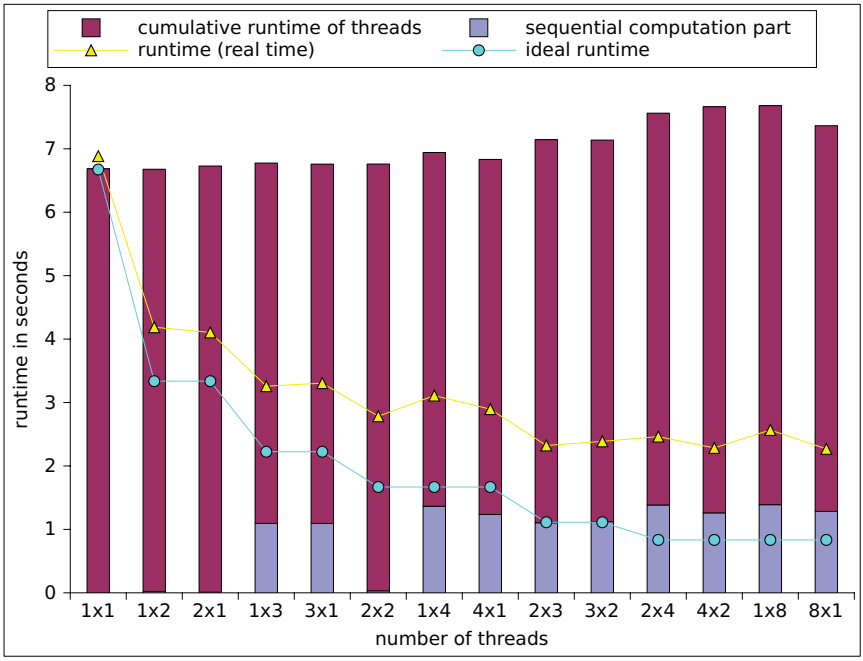


Figure 5: Runtime diagrams of the parallel algorithm which divides the DEM into squares. The number of threads on the x–axis is displayed in the form of „$a \cdot b$“ where a is the number of rowwise partitions, and b the number of columnwise partitions.

3.2 Dividing the computation into two phases

In Table 3, the speed-ups for up to four threads are presented. The runtime (real time) compared to the ideal runtime is shown in Figure 6. The stacked bars show the cumulative calculation time of threads of both phases and the runtime which is needed to group the cells. The calculation time is the period in which the threads perform computations without synchronization and communication. The maximum waiting time becomes significant when using more than two threads (“Reintal” and “Mulde”) or more than three threads (“Saidenbachtal”). For instance, using four threads for computing the flow directions of “Saidenbachtal”, one of the four threads had to wait up to 63% of the whole parallel runtime. In case of less than four threads, a thread has to wait for a task up to one percent of the whole runtime. Regarding the dataset “Mulde” a thread has to wait up to 23% (four threads) or 28% (eight threads), respectively, of the whole parallel runtime. The average waiting time of a thread is 14% (using four threads) or 27% (using eight threads), respectively, of its calculation time.

Working on the data set “Saidenbachtal” and “Reintal”, further investigations show that all threads have nearly equal computation time. The effort of calculation is well balanced, but in some periods there were no tasks for the threads. Thus, one or more threads had to wait. The maximum difference between computation time and the cumulated time of all

threads were determined, too. The more threads were used the more threads had to wait to get new tasks, but the threads had all nearly the same computation time.

Using the data set “Mulde”, we get a maximum difference between the calculation time of each thread of 10% (two threads are used) up to 18% (eight threads are used).

Table 3: Speed-ups of the parallel algorithm which divides the computation into two phases.

number of threads:	$\mathbf{1 \cdot 1}$	$\mathbf{1 \cdot 2}$	$\mathbf{2 \cdot 1}$	$\mathbf{1 \cdot 3}$	$\mathbf{3 \cdot 1}$	$\mathbf{2 \cdot 2}$
speed-up (DEM 1 “Reintal”):	0.90	1.67	1.73	2.23	2.26	2.47
speed-up (DEM 2 “Saidenbachtal”):	0.94	1.82	1.84	2.38	2.40	2.58
speed-up (DEM 3 “Mulde”):	0.85	1.76	1.77	2.31	2.29	2.08

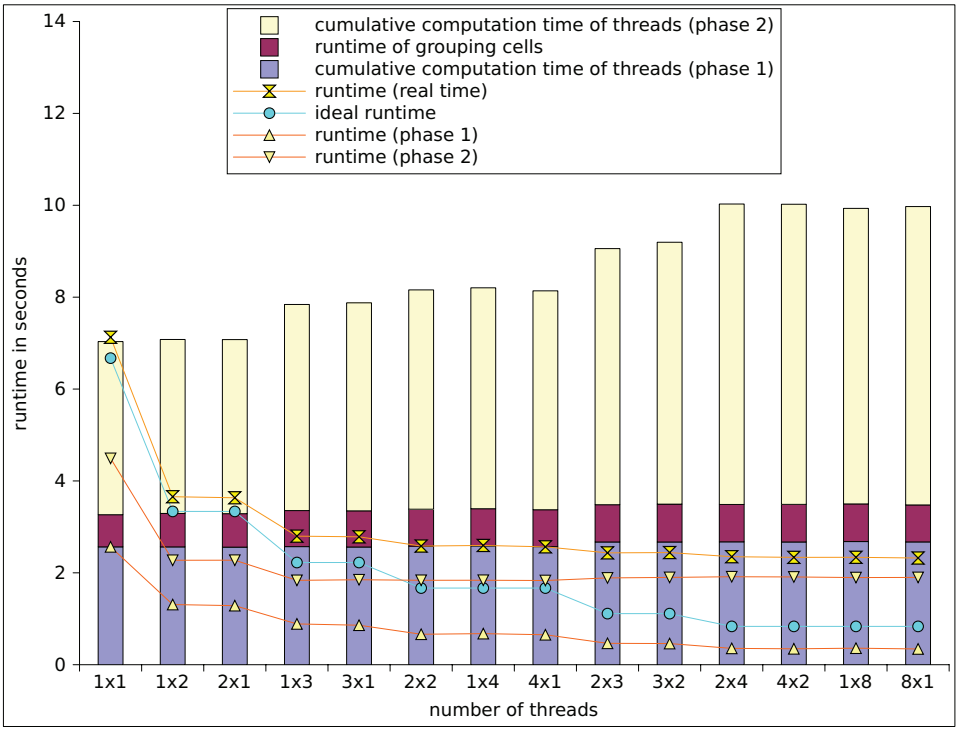


Figure 6: Runtime diagrams of the parallel algorithm which divides the computation into two phases. The number of threads on the x–axis is displayed in the form of „$a \cdot b$“ where a is the number of rowwise partitions, and b the number of columnwise partitions. Theses partitions only apply to the first phase.

We have implemented several modifications of the algorithm to improve the runtime. One modification eliminates recursion. Another one implements a heuristic which tries to change the processing order in such a way that large groups of cells are handled with priority. To avoid an overwhelming effort for the data access, a modification – which

merges small groups of cells together and puts those into the task pool – has also been implemented. However, all these modifications did not show significant runtime differences.

As shown in Figure 6, for all test sites the runtime decreases but the accumulated computation time increases with the number of threads. There is no explicit synchronization during the calculations because of the disjoint groups of cells. An increased number of threads causes an increase of the overhead of thread administration and an increase of random access which can induce for instance false sharing (see [HS08]). The cumulated waiting time also increases.

4 Discussion and conclusion

In this study, we investigated the efficiency of parallelization techniques on the example of the non-local "extended neighborhood" of raster cells. An extended neighborhood is used within the D8*e* algorithm to make the flow direction unique when ambiguous flow directions occur. The ordinary D8 neighborhood has ambiguous flow directions. Two parallelization approaches were tested regarding their efficiency (Section 2.2):

1. The advantage of parallelization by dividing the DEM into squares is the low cost of synchronization. A disadvantage is the inefficient load balancing. On the one hand, if we have a well-adjusted load balance the speed-ups would be improved. On the other hand, speed-ups near to the best possible speed-up in all cases could not be achieved because of the sequential part (pre-computation step). Synchronization costs are insignificant in our implementations.

2. The advantage of parallelization by dividing the computation into two phases is the well-adjusted load balance. Disadvantages are the increasing cost of synchronization and data access. The sequentially grouping of cells causes increased computation expenditure and is independent from the number of threads. This parallelization alternative is more independent of the composition of the DEM because of the dynamic distribution of the cost-intensive calculations to the threads. Possible reasons for the non-ideal speed-ups have been examined. Unbalanced distribution of calculations, synchronization time and conflicts between threads because of a shared data structure can be excluded as being mainly responsible for these observations. Possible explanations are data access and the effect of false sharing caused by the high number of write and read accesses.

Neither parallelization strategy enabled speed-ups that are equal to the number of threads. This is in contrast to the parallelized calculation of local surface parameters like slope and aspect where speed-ups near to the number of threads have been achieved for all three study areas. The speed-up comparisons also revealed landscape-related dependencies. While the speed-ups for the high mountain dataset computation are higher by running the first strategy (DEM "Reintal"), the speed-ups of the second strategy have proved to be more efficient for the calculation on the low mountain and the floodplain datasets (DEM 2

"Saidenbachtal" and DEM 3 "Mulde"). The second case shows the effect of flat areas (e.g. dams, filled sinks and floodplains) on the parallelization efficiency where a fixed DEM division into threads was carried out. This implies that the distribution of the extended neighborhoods is fixed, too. This could lead to a sub-optimal load balancing. The second parallelization strategy is more appropriate to such DEMs because of the dynamic consideration of the extended neighborhood. As mentioned in the abstract, this task pool-based strategy is more efficient for calculations on datasets with many data dependencies. However, the second parallelization strategy is more computationally intensive. Thus, in the case of almost optimal load balances (e.g. dataset "Reintal") the speed-ups of the second parallelization strategy are lower than the speed-ups of the first one.

In all datasets there are many small and few huge extended neighborhoods. As a consequence, one thread could work on just one huge extended neighborhood while the other threads have already finished.

Further runtime measurements based on different spatial resolutions (area of DEM 1 in $2 \times 2\,\mathrm{m}^2$ and $5 \times 5\,\mathrm{m}^2$ resolution, area of DEM 2 in $5 \times 5\,\mathrm{m}^2$ resolution, and area of DEM 3 in 2×2 resolution) were executed, too. Because of the decreased runtime of the calculations caused by the lower number of cells, the overhead of administration of the threads becomes more significant. The speed-ups were a bit lower than the speed-ups presented above. Runtime measurements based on the original DEMs (the datasets previous to the sink filling) were also executed. Because of significantly fewer flat areas, the speed-ups were significantly higher than the presented speed-ups.

In future work we will try to improve the computation by parallelizing the extended neighborhood computation. But this seems to be challenging because of data dependencies. We will also work on much larger datasets (about 10^9 cells). Because of runtimes greater than some minutes we will focus on applications besides web-based implementations, too. In anticipation of a growing number of cores per processor, it will be worth studying other parallelization strategies for shared memory machines.

References

[ASMB03] W. Al-Sabhan, M. Mulligan, and G.A. Blackburn. A real-time hydrological model for flood prediction using GIS and the WWW. *Computers, Environment and Urban Systems*, 27(1):9–32, 2003.

[GP09] S. Gruber and S. Peckham. Land-surface parameters and objects in hydrology. In T. Hengl and H.I. Reuter, editors, *Geomorphometry - Concepts, Software, Applications*, volume 33 of *Developments in Soil Science*, pages 171–194. Elsevier, Amsterdam, The Netherlands, 2009.

[GTDS10] C. Gläßer, D. Thürkow, Ch. Dette, and S. Scheuer. The development of an integrated technical-methodical approach to visualise hydrological processes in an exemplary post-mining area in Central Germany. *ISPRS Journal of Photogrammetry and Remote Sensing*, 65(3):275–281, 2010. Theme issue "Visualization and exploration of geospatial data".

[HS08] M. Herlihy and N. Shavit. *The art of multiprocessor programming*. Elsevier, Amsterdam, The Netherlands, 2008.

[MAAR10] T. Mølhave, P. K. Agarwal, L. Arge, and M. Revsbæk. Scalable algorithms for large high-resolution terrain data. In *Proceedings of the 1st International Conference and Exhibition on Computing for Geospatial Research & Application*, COM.Geo '10, pages 20:1–20:7. ACM, 2010.

[Mow94] J. E. Mower. Data-parallel procedures for drainage basin analysis. *Computers & Geosciences*, 20:1365–1378, November 1994.

[NFT09] J.C. Neal, T. Fewtrell, and M. Trigg. Parallelisation of storage cell flood models using OpenMP. *Environmental Modelling & Software*, 24:872–877, 2009.

[OM84] J.F. O'Callaghan and D.M. Mark. The extraction of drainage networks from digital elevation data. *Computer Vision, Graphics, and Image Processing*, 28(3):323–344, 1984.

[OR10] L. Ortega and A. Rueda. Parallel drainage network computation on CUDA. *Computer & Geosciences*, 36:171–178, 2010.

[RHGS09] H.I. Reuter, P. Hengl, P. Gessler, and P. Soille. Preparation of DEMs for Geomorphometric Analysis. In T. Hengl and H. I. Reuter, editors, *Geomorphometry - Concepts, Software, Applications*, volume 33 of *Developments in Soil Science*, pages 87–120. Elsevier, Amsterdam, The Netherlands, 2009.

[RR10] T. Rauber and G. Rünger. *Parallel Programming: For Multicore and Cluster Systems*. Springer, Berlin, Heidelberg, 2010.

[Sch10] S. Schiele. Effiziente parallele Simulation von Niederschlagsabflüssen in digitalen Geländemodellen. Master's thesis, Martin-Luther-Universität Halle-Wittenberg, 2010.

[TTW+11] T.K. Tesfa, D.G. Tarboton, D.W. Watson, K.A.T. Schreuders, M.E. Baker, and R.M. Wallace. Extraction of hydrological proximity measures from DEMs using parallel processing. *Environmental Modelling & Software*, 26(12):1696–1709, 2011.

[Woo09] J. Wood. Overview of Software Packages Used in Geomorphometry. In T. Hengl and H. I. Reuter, editors, *Geomorphometry - Concepts, Software, Applications*, volume 33 of *Developments in Soil Science*, pages 257–267. Elsevier, Amsterdam, The Netherlands, 2009.

[ZKLY07] L. Zhang, Z. Kang, J. Li, and L. Yang. Comparison of the performance of flow-routing algorithms used in GIS-based hydrologic analysis. *Hydrological processes*, 21:1026–1044, 2007.

A Speed-Up Study for a Parallelized White Light Interferometry Preprocessing Algorithm on a Virtual Embedded Multiprocessor System

Dominik Schoenwetter, Max Schneider and Dietmar Fey

Chair of Computer Science 3 (Computer Architecture)
Friedrich-Alexander-University Erlangen-Nuremberg
Martensstr. 3, 91058 Erlangen
{dominik.schoenwetter, max.schneider, dietmar.fey}@informatik.uni-erlangen.de

Abstract: Parallel computing has been a niche for scientific research in academia for decades. However, as common industrial applications become more and more performance demanding and raising the clock frequency of conventional single-core systems is hardly an option due to reaching technological limitations, efficient use of (embedded) multi-core CPUs and many-core platforms has become imperative. 3D surface analysis of objects using the white light interferometry presents one of such challenging applications. The goal in this article is to get an impression which speed-up for an established and parallelized white light interferometry preprocessing algorithm, called *Contrast Method*, is possible on an embedded system that works without any operating system. Therefore, we decided to use a virtual environment that is able to simulate embedded multi-core as well as many-core systems and that enables running real application code on the designed system. The results show, that a significant speed-up is possible when using a many-core platform, instead of a design that only implements one single core, if the algorithm is parallelized for getting full advantage of the many-core design. Furthermore, an acceptable absolute run time is achievable.

1 Introduction

The white light interferometry scanning is a versatile technology which provides a reliable non-contact, 3D optical measurement of surface roughness in the nanometer range [KM07]. In the scanning device, which is usually a Michelson interferometer equipped with a broadband light source, the emitted white light beam is split into two separate beams. One of the beams is projected onto the object to be measured, while the other beam follows a well defined and constant path to a reference mirror. Both beams are reflected and superimposed, resulting in an interference pattern of light and dark fringes. This fringe pattern is captured on a CCD camera chip and processed in software. By moving the object closer to the scanning device in discrete time steps, the path difference between the two beams and, thus, the fringe pattern changes. During a common measurement process, the whole interference range (the region where the path length difference of reflected beams is less than the coherence length of the wight light) is covered [Lar00].

Thereby, individual time series of interference intensities are recorded by the pixels of a CCD sensor. Figure 1 shows such a two dimensional series, called *correlogram* or *interferogram*, for one pixel. At each pixel, where the optical path length difference of both beams is zero, the occurred constructive interferences have reached their maximum value. In the case that the object is not a flat plane, the maximum interference of each pixel point is obtained in different time steps for each pixel. A 3D map can be derived from the positions of the translation arm, where maximum intensities are observed, and the distance to the start position [His05]. Thus, the aim of the white light interferometry analysis process is to find the corresponding maximum interference value for each pixel of the CCD sensor.

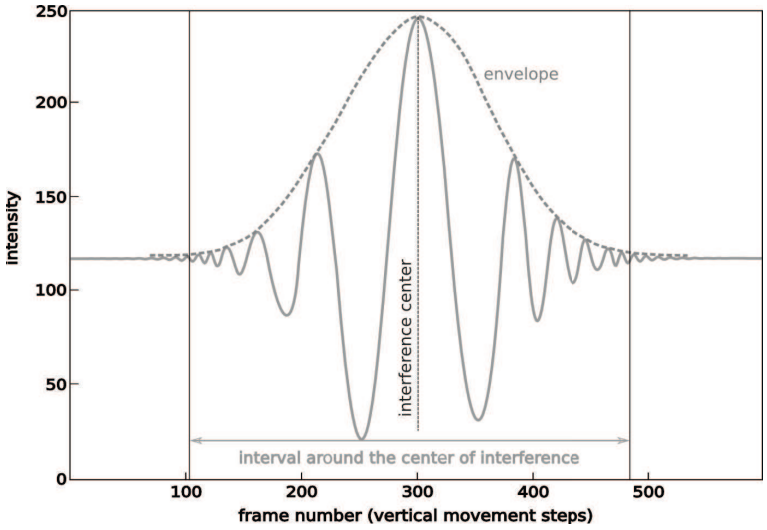


Figure 1: A synthetic interferogram or correlogram signal of a pixel

The measurement of the whole test object is done using the so-called stitching process. This means, that the scanning area is subdivided in 1×1 mm^2 regions that are scanned subsequently. For example a 25×25 mm^2 measuring area must be subdivided in 625 partitions. Usually more than 200 frames are necessary for a complete scan of each subarea. The scan of the whole area takes approximately 70 minutes with the camera that is used at the moment. To increase performance, a high speed camera is used to accelerate the scanning procedure. The necessary computations on the data, which are captured for such a subarea, are performed while the scan of the next subarea is already in execution. Because the scanning procedure is accelerated, the elaboration procedure has to be accelerated, too. This can be achieved by the parallelization of the elaboration procedure.

The *interference range* (*IR*) span at most few hundred intensity values and all other intensities captured by the camera are not relevant for the height map calculation. Thus, a preprocessing step in the white light interferometry analysis is used, to reduce the required image data to significant regions for the height map calculation. In the postprocessing

stage, the maximum interference for each pixel in the corresponding extracted region is calculated [His05].

Because each pixel's correlogram is elaborated independently from those of the other pixels, the analysis process is well suited for parallel processing.

As can be seen in Section 2 there are studies about performance and speed up using white light interferometry with GPUs as well as with conventional multi-core systems from Intel and AMD. According to our present knowledge there are no studies which performance and which speed up could be achieved on single or multi-core and many-core, respectively, embedded processors. As a consequence this is an unexplored niche which is, according to our opinion, a very interesting region because the usage of embedded multi-core systems is growing more and more and embedded computing gives a substantial added value to many products in fields such as automotive, industrial automation, telecommunications, consumer electronics or entertainment [COMS11]. Using an embedded hardware layout for the white light interferometry has the advantage of more power efficiency (less power consumption). When using a NVIDIA Tesla C2050 GPU, there is a maximum power consumption of 238 W TDP (Thermal Design Power) [Cor10]. For the Xeon X5650 Hexacore (2.66 GHz), a conventional multi-core processor developed by Intel, the power consumption amounts to 95 W TDP [Int11]. In comparison to that, a single ARM CortexA9 processor, that we used for our investigations, has a power consumption of 0.4 W (soft macro implementation) [ARM11]. As a consequence, the power consumption of an embedded multi-core system, consisting of single CortexA9 processors and up to ten cores, is only a fraction compared to a NVIDIA Tesla C2050 GPU or a Xeon X5650 Hexacore. In addition to that, a more compact construction of the measurement system is possible when using an embedded system instead of a GPU or a conventional multi-core system.

Our final goal is to integrate such a compact embedded system in a smart thrilling head, which includes micro optical devices and fast electronic evaluation systems, too. A GPU board with cooling equipment, for example, could not be integrated in the even mentioned thrilling head, because there is plain and simple not enough space available.

The development of a real embedded hardware layout can be very expensive because, as it is often the case, the hardware layout has to be changed during the development phase. To avoid this it will be of great advantage to use an environment that emulates the embedded hardware and could be modified whenever necessary. Such a virtual environment for embedded hardware and software development is *Open Virtual Platforms*^{TM} (*OVP*^{TM}) provided by *Imperas*^{TM}. With the aid of OVP it is possible to build single-core as well as multi-core and many-core hardware platforms, add desired peripherals and simulate real application code [OVP11].

Because of the ability to establish multi-core and many-core platforms running real application code, it is possible to develop parallel applications that can be simulated, verified and evaluated. That is a very important feature for our work and the reason why we chose OVP as the virtual environment for our study. The chosen preprocessing algorithm is the so-called Contrast Method (see Section 4). This method uses only one arithmetic instruction to calculate the central fringe in each interferogram and achieves a high performance as a consequence [SFKM11].

2 Related Work

The usage of multi-core and many-core technology in industrial white light interferometry is gaining more and more attention. The first attempt to use a graphical processing unit in the surface metrology has been approached by Purde et al. in 2004 [PMS+04]. They implemented analysis algorithms of the so called electronic speckle pattern interferometry on GPUs, allowing the measurement of surface contours using the High Level Shading Language (HLSL).

Gao presented in September 2010 a solution for imaging processing using GPUs for the wavelength scanning interferometry [GJMM10]. Thereby a GeForce GTX 280 was used for the parallelization of the computational intensive data analysis procedure and a approximately 30x speed-up was achieved.

Also in September 2010 Sylwestrzak et al. presented the application of massively parallel processing of Spectral Optical Coherence Tomography (SOCT) data using GPUs [SSST10]. By utilizing NVIDIA's GeForce GTX 285 with 2 GB device memory Sylwestrzak achieved overall imaging speed of over 100 fps for 2D tomograms consisting of 1024 A-scans.

In 2011 Schneider et al. published an article about three designated preprocessing white light interferometry algorithms on emerging multi- and many-core architectures [SFKM11]. He figured out that the best performing algorithm is the Contrast Method and that conventional multi-core systems are the best suited architectures for this algorithm.

3 The simulator

The simulator included in OVP is an instruction accurate simulator because the provided processor models are instruction accurate, too. This means, that the functionality of a processor's instruction execution is represented without regard to artifacts like pipelines.

OVP processors in multiprocessor platforms are not working simultaneously. For efficiency, each processor advances a certain number of instructions in turn. So in multiprocessor simulations a processor cannot respond until the processor has signaled, that he has finished its quantum. The quantum is defined as the time period in which each processor in turn simulates a certain number of instructions [Lim11a]. Simulated time is moved forward only at the end of a quantum. This can create simulation artifacts, where a processor spends time in a wait loop, while waiting for the quantum to finish. To avoid this the quantum has to be set very low (perhaps even to one, which will have a significant impact on simulation performance) so that the measurements will not be affected by this simulation artifacts. This can be adjusted in the simulator settings [Lim11b]. The simulation can only figure out how many instructions were executed. Assuming a perfect pipeline, where one instruction is executed per cycle, the instruction count divided by the mips rate (millions of instructions per second) would give the amount of time the program runs. Instruction accurate simulation cannot make a clear statement about time spent during pipeline stalls, due to cache misses and other things that are not modeled, so any conversion to time will have limited accuracy compared to actual hardware. But it is still

useful for comparing the relative performance of different algorithms, assuming that they have similar pipeline stall effects. Furthermore, the OVP-simulator provides the possibility for measuring instruction counts within a program. As a consequence, the instruction counts for specific code snippets can be recorded and if the quantum is set to one, like in this study, the registered instruction count divided by the mips rate is the amount of time the processor requires for executing the code snippet. In single-processor platforms there is no need to set the quantum to one because the multiprocessor scheduling algorithm does not affect and intervene, respectively, the simulation.

4 The preprocessing algorithm

In the preprocessing analysis step, each pixel's correlogram is demodulated, separating the carrier wave from the envelope. The demodulation process can be done by a simple approximation of envelope values. For this purpose, depending on the surface characteristics and the signal-to-noise ratio of the generated signals, different approaches have to be considered [Rob93]. Envelope demodulation serves also as a data reduction step, because only the interval around the center of the interference is relevant for the postprocessing [KM07], as can be seen in Figure 1. The center of interference signal itself can not be seen as the envelope's peak. Due to noise and the discrete measurement along the z-axis, the actual maximum could be between two captured intensities or it could be shifted in some direction away from the measured interference center. Thus, in the demodulation process the center of the interference is determined and a predefined number of intensities left and right from this point is extracted. This is done parallel to the scanning procedure, so that not all interference images must be stored, but only the relevant regions in corresponding correlograms. These regions are used in the postprocessing stage to get an approximation of the actual maximum, as accurate as possible. The approximation can be achieved by fitting a model envelope function to the detected envelope [Lar00].

For our investigations a algorithm called Contrast Method was chosen. For each pixel p it uses the maximum absolute difference of successive sampling points I from the input signal as an estimator for the envelope, see (1). Variable i represents the number of the current translation step. This filter becomes maximum where the interferogram oscillations have a maximum gradient, which is approximately around the maximum of the envelope [His05].

$$\hat{z}_0(p) = \arg\max_i |I_{i-1}(p) - I_i(p)| \tag{1}$$

5 Implementation details

5.1 Implementation of the virtual test environment

For generating the virtual hardware design with OVP, an application, that defines the necessary components at runtime, was written. The application accepts several parameters, e.g. if intermediate results shall be displayed or not, and of which type the used virtual processor is, e.g. a *CortexA9* made by *ARM* that we used for our investigations. One important parameter defines the number of CortexA9 processors that should be implemented for the design. Depending on that parameter, the corresponding application to run by the respective core is automatically loaded into the corresponding processor memory. There are two different programs that have to be loaded into the memories, if more than one core shall be used. These core-applications are called *MasterApp* and *SlaveApp* (see section 5.2). For each processor two local memories are generated. The first one contains the heap, the stack and the memory area for the application program to run. The second local memories (one belonging to each processor) contain the data, the Contrast Method works with, as well as necessary synchronization variables. These second local memories are implemented as a distributed shared memory system. As a consequence, each core has the possibility to access the variables that are required for the synchronization of every processor. After all local memories are instantiated, they are linked to the associated cores. The resulting hardware design, depending on the number of cores to implement, is graphically illustrated in Figure 2.

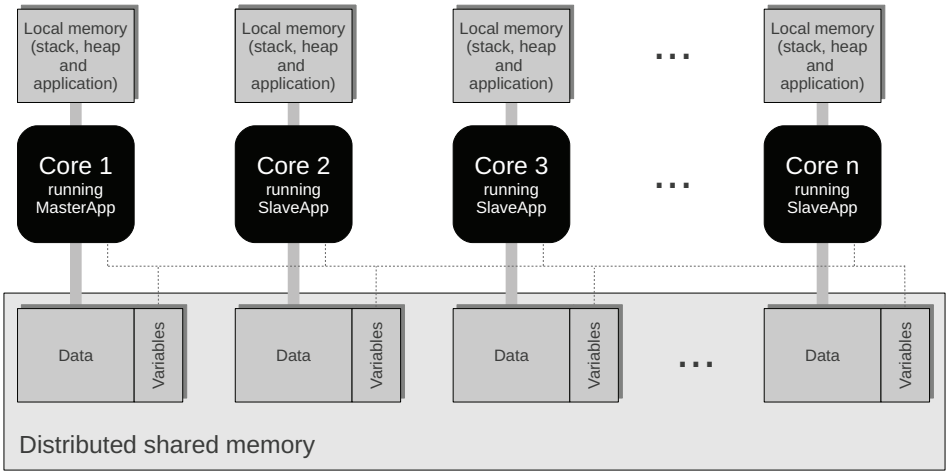


Figure 2: Generated design depending on the number of cores to use

5.2 Parallel implementation of the algorithm

Employed cameras in industrial applications use different color depths (8 bit / 16 bit). However, the data analysis has to be accomplished in single precision or in double precision mode. Therefore, we decided to employ template classes, so that all the functionality needed to execute the described algorithm with corresponding parameter combinations, can be accessed through the same class. The user only has to provide the template parameters during instantiation. The methods of this class are compiled and linked into an application named *MasterApp*, that is executed by only one core, independent of the number of cores used in the design, called *master* in this paper. This processor defines all necessary shared (synchronization) variables and their contents in the initial phase. All other cores (the *slaves*) run another application that waits continuously for instructions of the master with the exception of the initial phase where all address assignments of required shared variables are performed. The application running on the slaves is called *SlaveApp* (refer to Figure 2). The interaction of both applications is illustrated in the following. As mentioned in Section 1, each pixel's correlogram is elaborated independently from those of the other pixels. So the first stage of the preprocessing algorithm (computing the maximum of each pixel's interferogram) and the second stage (calculating the relevant intensities around the maximums) may also be regarded as independent. Hence, the parallelization of the algorithm takes places here. The number of pixels of an image can be seen equal to the runs of several for-loops that have to be executed during the algorithm and whose instructions are the same for every pixel's correlogram. The number of pixels each core has to process shall be evenly distributed, i.e. n_P/n_C. Thereby, n_P is the number of pixels per frame and n_C is the number of cores used in the design. No core may process more than one pixel than any other core to achieve an almost equal load balancing. If the reminder rem of the division n_P/n_C is unequal to zero, then one pixel more is assigned to the first rem cores.

For a clearer understanding, Listing 1 illustrates this circumstance in pseudo code.

```
// Total number of cores used in the design
unsigned int number_of_cores;
// Total number of pixels per frame
unsigned int number_of_pixels;
// Array that holds the runs each core has to perform
// runs_per_core[0] = number of loop passes for core 1
// runs_per_core[1] = number of loop passes for core 2
// ...
unsigned int runs_per_core[number_of_cores];
// The number of cores that have to perform one loop pass more
// than the other cores
unsigned int rem = number_of_pixels % number_of_cores;
// Assign loop passes
for(unsigned int index = 0; index < number_of_cores; index++)
{
    // Integer division
    runs_per_core[index] = number_of_pixels / number_of_cores;
    if(i < rem)
        runs_per_core[index] += 1;
}
```

Listing 1: Pseudo code for equal load balancing

Because all relevant data is located in the distributed shared memory, and each core is working on a different address space within the memory area, there are no simultaneous memory accesses that could trigger additional delays. Everytime such a parallelized region is accessed by the application running on the master, he instructs all slaves to perform the necessary (and the same as the master) computations on their assigned data regions in the distributed shared memory. After all cores (including the master) have finished their work, the application on the master is continued and the slaves wait for processing (sleep mode) until the next parallelized region is entered. The even mentioned procedure is realized using the spin lock principle and without any operating system. The synchronization variables are located in the shared memory area which is accessible for each processor.

The whole number of input frames to analyze by the preprocessing algorithm, as well as the number of intensities to store around each pixel's interferogram for postprocessing, are depending on the user input. For test purposes the input frames are generated in software.

6 Results

A detailed and precise overview about the execution times, the Contrast Method requires for the chosen test cases on the respective virtual hardware, is shown in Table 1.

	256 x 256 pixels			**512 x 512 pixels**			**1024 x 1024 pixels**		
	p/sec	m/sec	c/sec	p/sec	m/sec	c/sec	p/sec	m/sec	c/sec
1 core	10.93	9.55	1.28	43.70	38.20	5.14	174.71	152.80	20.55
2 cores	5.59	4.78	0.72	22.32	19.10	2.86	89.19	76.40	11.43
3 cores	3.76	3.18	0.47	14.00	12.73	1.91	59.91	50.93	7.62
4 cores	2.85	2.39	0.36	11.34	9.55	1.43	45.27	38.20	5.71
5 cores	2.30	1.91	0.27	9.14	7.64	1.14	36.49	30.56	4.57
6 cores	1.93	1.59	0.24	7.68	6.37	0.95	30.64	25.47	3.81
7 cores	1.67	1.37	0.20	6.63	5.46	0.82	26.45	21.83	3.27
8 cores	1.47	1.19	0.18	5.85	4.78	0.71	23.32	19.10	2.86
9 cores	1.32	1.06	0.16	5.24	4.25	0.64	20.88	16.98	2.54
10 cores	1.20	0.96	0.14	4.75	3.82	0.57	18.93	15.28	2.29

Table 1: Detailed overview of obtaining times depending on the number of cores and the resolution

Thereby, variable p stands for the required time of the whole preprocessing algorithm, variable m for the time that is required for calculating the maximum of each pixel's interferogram and variable c is the representative for the time required for computing the intensities around the maximums. Each measurement was executed with a setup where a color depth of 16 bits and the double precision processing mode were used. Furthermore, the number of frames for the scanning procedure was set to 100 and the number of frames

containing relevant data (intensity values) around the maximum of each pixel's interferogram was placed to 30. The resolution by contrast changed during the measurements from 256×256 to 512×512 and finally to 1024×1024 pixels.

As expected, the amount of the required time for calculating the maximums and for computing the intensities is not equal to the time the whole preprocessing algorithm requires (serial proportion). For a resolution of 256×256 pixels this proportion amounts to 100 ms, for 512×512 pixels to 359 ms and for 1024×1024 pixels to 1.357 seconds of the whole algorithm. As a consequence, the percentage of this serial proportion rises (with reference to the time the whole algorithm requires), the more cores are used.

Based on the results shown in Table 1, the speed-up, depending on the number of cores used and in comparison to one core, for each resolution can be calculated. Figure 3 illustrates the resulting speed-ups of the whole preprocessing algorithm for different resolutions graphically.

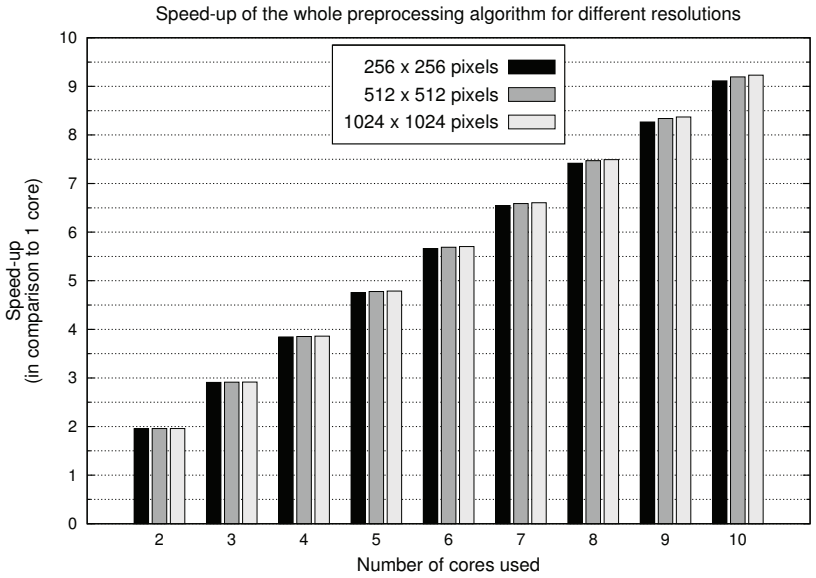


Figure 3: Speed-up of the whole preprocessing algorithm for different resolutions

The speed-up for calculating the maximum of each pixel's correlogram, in comparison to one core, is numerically equivalent to the number of cores used and as a consequence constant. As opposed to this, the speed-up for the computation of the intensity interval around the maximum of each pixel's interferogram is numerically not equal to the number of cores used, but constant, too. For that reason, the speed-up of the whole algorithm and the Contrast Method, respectively, is not numerically equivalent to the number of cores used. Despite this, the speed-up of the whole algorithm is linear. This results from the distributed shared memory architecture. Each core operates on the data in its own local memory and thus independent of the other cores. As a consequence, there are no

simultaneous memory accesses, that could slow down the application, even in practice. Only simultaneous access to the shared synchronization variables could slow down the application. This is the case when the master polls for the synchronization variable of a slave (read), while the slave wants to signal that he has finished its computations (write).

In multiprocessor simulations, and as mentioned in Section 3, each core advances a certain number of instructions in turn. This number of instructions (the quantum) can be set very low to increase simulation accuracy and to avoid simulation artifacts. As a consequence, setting the quantum very low has a significant impact on simulation performance. We investigated this circumstance for a resolution of 256×256 pixels and for quanta of 1, 100 and 100000. The results have shown, that the *simulated time* of the whole system, i.e. the time the simulator determines for the execution in the corresponding real system, is not influenced by the quantum and amounts to 55.67 seconds for a system consisting of two cores. As opposed to this, the *wall-clock time* changes significantly. For the simulation of two cores and a quantum of 1, the wall-clock time amounts to 4 hours and 27 minutes. Setting the quantum to 100 evokes a wall-clock time of 33 minutes and 36 seconds. The fastest simulation is achievable when setting the quantum to 100000. The wall-clock time than amounts to 42 seconds only. When investigating the instruction counts for the whole preprocessing algorithm, an amendment depending on the quantum is ascertainable, too. For a quantum of 1 the instruction count of the whole preprocessing algorithm results in 558,969,518 instructions, for a quantum of 100 in 558,969,742 instructions and for a quantum of 100000 in 559,090,510 instructions. When dividing that instruction counts by the mips rate, which was set to 100, preprocessing times of 5.58970 seconds for a quantum of 1, 5.58970 seconds for a quantum of 100 and 5.59091 seconds for a quantum of 100000 are emerging. Thus, the simulation accuracy changes at the second decimal place and at the tenth millisecond place, respectively. This circumstance was noticed independent of the number of cores used. Because the results of our measurements for the whole preprocessing algorithm are in a range of seconds (see Table 1), a quantum of 100000 is rather adequate to provide enough accuracy for our simulations. As a consequence, the wall-clock time of the simulation is reduced heavily.

7 Conclusion and future work

The speed-up for the parallelized preprocessing algorithm, called Contrast Method, rises sharply the more cores are used. Furthermore, the speed-up is approximately independent of the chosen resolution for a constant number of frames to evaluate. When more than ten cores are used, we expect the speed-up to remain linear until the saturation is reached, also independent of the resolution. In this context, saturation is reached, when no more notable speed-up can be achieved by raising the number of cores. This depends on the number of frames to evaluate and on the chosen resolution. In comparison to the times that are achievable on a GPU (range of milliseconds for a resolution of 1024×1024 pixels), the times we achieved (see Table 1) are slower, but sufficient enough for the timings required for an industrial inspection process. Concerning to power consumption we can clearly make the statement that an embedded solution will reduce power consumption in compar-

ison to a GPU or a conventional multi-core architecture (see Section 1). As a result, the full advantages, regarding to compactness and less power consumption can be occupied. The advantage of a virtual environment is the possibility to simulate any desired number of cores, even if there is no corresponding real hardware. Our final goal to integrate such a compact embedded system in a smart thrilling head is therefore achievable.

In future work we will extend our approach by parallelization of the remaining steps from the white light interferometry data analysis process, which includes the analysis of each interferograms phase information [His05] and the postprocessing step, where a Gauss-Newton fitting is applied to each correlogram of a pixel. Furthermore, we will investigate the difference between the measured times and instruction counts, respectively, when using other available virtual cores than the CortexA9. It would be very interesting to see what total power consumption appears on the simulated hardware. OVP provides statistics of power consumption for the simulated hardware not directly. However, with some software effort it is possible to get these statistics. This is another point we will treat in future. Last but not least, one important thing is the comparison of the times figured out with OVP and the times appearing on real hardware. Therefore, we want to recreate the virtual hardware design, introduced in this paper, with one to four cores in real. Afterwards, we will port our software to the real hardware and compare the times we determined with OVP to the times arising on the real hardware.

References

[ARM11] http://www.arm.com/products/processors/cortex-a/cortex-a9.php?tab=Performance, 2011. Last visit on 14.12.2011.

[COMS11] M. Conti, S. Orcioni, N. Martinez Madrid, and R. E.D. Seepold. *Solutions on Embedded Systems*. Springer Dordrecht Heidelberg London New York, 2011. doi: 10.1007/978-94-007-0638-5.

[Cor10] NVIDIA Corporation. *Datasheet TESLA C2050/C2070 GPU computing processor*, July 2010.

[GJMM10] F. Gao, X. Jiang, H. Muhamedsalih, and H. Martin. Wavelength scanning interferometry for thin film analysis of fusion target. In *3rd European Target Fabrication Workshop, Science & Technology Facilities Council*, 2010.

[His05] M. Hissmann. *Bayesian estimation for white light interferometry*. PhD thesis, Combined Faculties for the Natural Sciences and for Mathematics of the Ruperto-Carola University of Heidelberg, 2005.

[Int11] http://ark.intel.com/products/47922/Intel-Xeon-Processor-X5650-(12M-Cache-2_66-GHz-6_40-GTs-Intel-QPI), 2011. Last visit on 14.12.2011.

[KM07] D. Kapusi and T. Machleidt. White Light Interferometry in Combination with a Nanopositioning- and Nanomeasuring Machine (NPMM). In *Proceedings of the International Society for Optical Engineering*, 2007.

[Lar00] K. G. Larkin. *Topics in multi-dimensional signal demodulation*. PhD thesis, The Faculty of Science in the University of Sydney, 2000.

[Lim11a] Imperas Software Limited. *OVP Guide to Using Processor Models*. Imperas Buildings, North Weston, Thame, Oxfordshire, OX9 2HA, UK, July 2011. Version 0.4, docs@imperas.com.

[Lim11b] Imperas Software Limited. *OVPsim and Imperas CpuManager User Guide*. Imperas Buildings, North Weston, Thame, Oxfordshire, OX9 2HA, UK, September 2011. Version 2.0.40, docs@imperas.com.

[OVP11] www.ovpworld.org, 2011. Last visit on 10.11.2011.

[PMS+04] A. Purde, A. Meixner, H. Schweizer, T. Zeh, and A. Koch. Pixel shader based real-time image processing for surface metrology. In *Instrumentation and Measurement Technology Conference, 2004. IMTC 04. Proceedings of the 21st IEEE*, volume 2, pages 1116 – 1119 Vol.2, 2004. doi: 10.1109/IMTC.2004.1351259.

[Rob93] D. W. Robinson. *Interferogram Analysis: Digital Fringe Pattern Measurement Techniques*. Institute of Physics, Bristol, Philadelphia, 1993.

[SFKM11] Max Schneider, Dietmar Fey, Daniel Kapusi, and Torsten Machleidt. Performance comparison of designated preprocessing white light interferometry algorithms on emerging multi- and many-core architectures. In *Procedia CS*, volume 4, pages 2037–2046, 2011.

[SSST10] M. Sylwestrzak, M. Szkulmowski, D. Szlag, and P. Targowski. Real-time imaging for spectral optical coherence tomography with massively parallel data processing. In *Photonics Letters of Poland*, volume 2, 2010.

Achieving scalability for job centric monitoring in a distributed infrastructure

Marcus Hilbrich, Ralph Müller-Pfefferkorn

Center for Information Services and High Performance Computing (ZIH)
Technische Universität Dresden
01062 Dresden
Germany
marcus.hilbrich@tu-dresden.de
ralph.mueller-pfefferkorn@tu-dresden.de

Abstract: Job centric monitoring allows to observe jobs on remote computing resources. It may offer visualisation of recorded monitoring data and helps to find faulty or misbehaving jobs. If installations like grids or clouds are observed monitoring data of many thousands of jobs have to be handled.

The challenge of job centric monitoring infrastructures is to store, search and access data collected in huge installations like grids or clouds. We take this challenge with a distributed layer based architecture which provides a uniform view to all monitoring data. The concept of this infrastructure called SLAte and an analysis of the scalability is provided in this paper.

1 Introduction

Direct observability of computing tasks is more and more lost by using external, distributed resources for getting calculations done. Job centric monitoring is a service which takes the challenge to fill the gap between using external resources and direct observability of jobs. Therefore monitoring data of each job are recorded for future analysis and visualisation. Job centric monitoring gives detailed information about used resources of currently running jobs where grid middlewares or batch systems give just the fact that the jobs are running. Using detailed information enables us to analyse the behaviour of already finished jobs instead of just showing the exit state. So the reason of an aborted job can be analysed and found. This can be an unexpected behaviour of the own job or that ensured computing resources are not available due to hardware problems or misbehaving users. A comparison of jobs is also offered to detect unusual jobs even if they are not aborted.

Job centric monitoring is most benefiting for environments with large numbers of jobs and resources which are shared by various users. In these systems, users can not easily observe jobs like on local resources by using desktop monitoring systems. Resource monitoring does not solve this problem because it does not provide job observability. Observed are the characteristics of resource usage, not the impact of single users or specific jobs.

The architectures we have in focus for job centric monitoring are grids (offering huge amounts of heterogeneous resources), clouds (offering resources on demand) and HPC or cluster systems operated by different computing centres. Currently the D-Grid[1] infrastructure is used for research.

One of the challenges in this field of research is to develop visualisation systems which offer methods to handle thousands of jobs running on different hardware with changing side effects e.g., the influence of jobs of other users utilising the same computing systems. But before we can analyse the data we have to handle them. To get an idea which amount of monitoring data we have to handle we consider an example: A single measurement consists of $15\,kB$ of data[2] (e.g., user name, grid VO, timestamp, CPU time, load average, used and free memory, used and free swap space, IRQ load, job ID, host name). Assuming that a single computing system or resource provider offers capacity for 5000 jobs and a measurement is done every minute ($60\,s$), this results in a network load of $1.25\,MB/s$. Considering only ten of these systems brings us to $12.5\,MB/s$ or 833 packages of $15\,kB$ each per second only for storing these data! This huge number of small packages is easier to handle in a distributed infrastructure instead of a central storage system.

Our answer to the challenge of storing, accessing and searching huge amounts of job centric monitoring data is the infrastructure SLAte [HMP10]. It is build on a concept of distributed servers organised in three layers. These layers allow to distinguish between different network capabilities (within a site[3] and between different sites) and to build a monitoring infrastructure adapted to the network. The performance of each of the three layers can be increased by installing additional servers. Thus the performance of the monitoring infrastructure can be adapted to user needs.

This paper describes the SLAte architecture, major implementation details and related work. Based on the architecture it is demonstrated how the performance of the job centric monitoring infrastructure can be increased to handle the increasing amount of data caused by additional computing resources and users. Afterwards scalability issues and potential bottlenecks are analysed.

2 Related Work

A lot of scientific work has been done on monitoring in general and similar fields of research. These topics are:

Monitoring on local computing resources: For local monitoring command line tools like ps, top or free[4] or graphical ones like Gnome System Monitor[5] can be used. On clusters Ganglia[6] gives information about the utilisation of resources. The im-

[1] `http://www.dgrid.de/`
[2] About $15\,kB$ per measurement is what we measured on our current installations.
[3] A site is a set of resources of a resource provider at a single location.
[4] `http://procps.sourceforge.net/index.html`
[5] `http://library.gnome.org/users/gnome-system-monitor/stable/index.html`
[6] `http://ganglia.info/`

pact of a specific user or job can not be identified directly by these tools.

Tracing and profiling: Profiling e.g., done with GNU gprof[7] gives information which functions of a program are used and how long they are used. This information is often presented as statistical evaluation. Even more detailed information are given by tracing tools like Vampir [BHJR10]. These analyses often tend to significantly slowdown the application. Thus they are not valuable for monitoring many jobs at once. Moreover the infrastructure to handle profiling or tracing data is designed for one job and in most cases the transfer of data over a wide area network is not needed. Thus it can not be easily adapted for job centric monitoring.

Accounting: Accounting is used for billing the use of resources and discovering the utilisation of computing systems. Examples are SGAS [EGM+] and DGAS [PRAGA09]. Based on the fact that only basic information of a job have to be recorded the amount of data to be handled is low. Thus there is no demand on a scalable infrastructure which is suitable for job centric monitoring if a central database is able to handle all accounting data. An other aspect of accounting is a high demand on reliability and methods to verify information while job centric monitoring looks more on scalability and performance.

Logging: Logging means to record events relevant for the reconstruction of the life cycle of a job. These events are e.g., the start, exit and abort of a job like recorded with accounting systems. Another source of relevant events is the program itself. It can report which subtask is executed or if problems occur. Such problems are e.g., missing rights to read or write files, wrong configurations and missing input data. The information is strongly bound to the context of the program and logging has to be intended by the program developers. Thus, not each job can deliver logging data. The logging information is often written to so called log-files or recorded by syslog daemons implementing rfc 5424[8]. An centralised implementation of logging grid jobs is shown by [RSK+08].

Resource monitoring: The task of grid resource monitoring is to record information about grid computers. Collected are information about hardware, used middlewares, offered services, known outages, planed maintenances and utilisation or free resources. This allows to create statistics of reliability and utilisation of resources and services or to assign jobs to handy resources. Examples of this kind of projects are D-Mon [BBK+09], CMS Dashboard [ACC+10] and Ganga [VBC+10]. A lot of the information collected by these tools are static and have no need for a high frequency of updates. Thus all resource monitoring data generated on a huge system like grid are handled by a central database.

The specific needs for handling job centric monitoring data are not properly considered by any of these tools or fields of research. Thus, we developed new concepts for job centric monitoring which are explained in the following sections.

[7] http://sourceware.org/binutils/docs/gprof/
[8] http://tools.ietf.org/html/rfc5424

3 Scalable, Layered Architecture – SLAte

3.1 Architecture

The architecture of SLAte is designed in a way that installing additional servers increases the performance of handling monitoring data. This is needed if more users want to access monitoring data or new computing resources should be used. As previously introduced the network capacity may be a limiting factor when observing many jobs in SLAte the overall bandwidth can be adjusted. For easy access to the distributed monitoring data we provide an unified and global view.

To get the idea of a scalable monitoring infrastructure to reality, a layer based concept is realised. It consists of three layers schematically shown in Figure 1. The Short Time Storage (STS) layer gets the data from the compute nodes and stores them temporarily. The Long Time Storage (LTS) layer accumulates the data from the STS servers and stores them persistently and distributed over multiple servers. The outer layer is the Meta Data Service (MDS) which provides the global view of the data. In the following these layers are described in more detail.

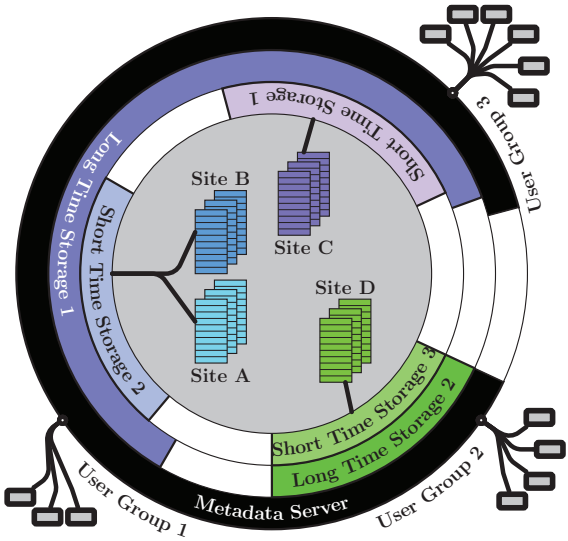


Figure 1: The layer based SLAte infrastructure for job centric monitoring data. Each layer can consist of multiple locally distributed servers. The servers of the MDS layer give a global view to all data which are stored distributed on the LTS servers.

The STS servers should be installed local at a site (e.g., at the frontend of a grid computing resource) to be close to the computing node on which the monitoring data are produced. To avoid that the monitoring data use too much resources on the computing node e.g., if stored in memory and to avoid that the monitoring data are copied after the job is done,

which prevents the next job to start, the monitoring information have to be moved to an STS server as early as possible. This results in a transfer for each single measurement and tends to many small packages which are handled by the local network (within a site or a computing system).

The monitoring data cached on the STS server have to be moved to the LTS server probably over a wide and slow network connection which is not able to handle huge numbers of small packages. To improve the usage of such a connection the monitoring data is transferred in batches. In most cases[9] the monitoring data of one job are moved at once. By repacking the monitoring data to huge packages it is avoided to slow down the network by transferring many small packages thus the effective network bandwidth is increased.

Like the STS layer, the LTS layer can consist of multiple servers to stores the monitoring data in a distributed way. In contradiction to the STS servers the LTS servers stores permanently and the monitoring data of individual STS servers can be merged (in Figure 1 symbolised by the sites A, B and C).

Furthermore an LTS server makes locally stored data accessible to users and visualisation systems. Therefore it can offer services to search and access job monitoring data.

To provide an unified view to the monitoring data distributed over LTS servers the MDS layer is used (see Figure 1). Like the other layers it can consist of multiple servers. Unlike the LTS and STS servers an MDS server has a global view to all data stored in the inner layers. In this way a user or tool easily accesses monitoring data without knowing any of the LTS servers. To realise this we distinguish between monitoring data and their metadata.

The monitoring data are the measurements (timestamp, CPU time, load average, used and free memory and so on) while the metadata hold information to search for jobs. Such an information is for instance the job ID. Usually, this ID is a unique number or a unique identifier like the Globus job ID. But mostly the job ID is not known by the user and it is preferred to search for the users jobs which run during a given time frame. Therefore the start and end time of a job are metadata as well as the name of the owner (more precisely the distinguished name of the certificate which was used to submit the job). Additional metadata are the exit state of the job and the storage location of the data.

To look up or search for jobs only the metadata are considered. This search can be performed by an LTS or an MDS server with the distinction that an LTS server holds only the metadata of the locally stored data while an MDS server has the metadata of all LTS servers. Thus an MDS server can find all monitoring data. The monitoring data is not stored on the MDS server because it is not needed for searching jobs. This dramatically reduces the amount of data transferred to these servers.

[9] To realise online monitoring it is needed to get monitoring data of running jobs. In this case the data on the LTS server are accessed and the data recorded afterwards have to be transferred in an additional package.

3.2 Implementation Details

The servers creating the three layers were implemented using the Grid middleware Globus (GT4) [Fos05] with its WSRF framework. The components are implemented in Java as GT4 resources and services which communicate with each other and the client applications via their SOAP based web services. So the components can be deployed to already running GT4 servers (like the frontend of computing resources) or dedicated GT4 containers can be used.

The security, authentication and authorisation services of GT4 are used to handle the monitoring data of grid users in a secure way. STS, LTS and MDS use grid server certificates to authenticate and authorise the communication. The authorisation of the user is based on users grid certificates.

Parts of the infrastructure we use for job centric monitoring are based on AMon [MPNB+06]. It offers visualisation components which are adapted to SLAte. Recording of data is done by lcg-mon-wn[10].

3.3 Exemplification of Scalability

Like already mentioned, one of the major concepts of SLAte is to be scalable with the demands of users and computing resources. Therefore additional servers can be installed in the three layers. In Figure 2 an example is shown how additional servers are added to increase the performance of SLAte.

Expansion stage 1 shows a minimal installation with one STS, one LTS and one MDS server for one computing resource. These servers are able to handle even more than one computing resource on the same site and additional users (expansion stage 2).

Additional computing resources on other sites require an additional STS server to buffer the monitoring data sent in small packages to the STS server. This configuration is shown as expansion stage 3 (Figure 2).

An additional LTS server adds storage capacity and a network link[11] which can be used to handle more STS servers with additional computing resources. This is shown by expansion stage 4.

Expansion stage 5 adds a MDS server to handle the search requests of an increased user group. In Figure 2 two distinct user groups are shown. If it is not possible to establish distinct user groups load balancing can be implemented with tools like pound[12].

[10] http://tu-dresden.de/die_tu_dresden/zentrale_einrichtungen/zih/forschung/grid_computing/amon/index_html

[11] A network link is added if the new LTS server is deployed on a new location.

[12] http://www.apsis.ch/pound/

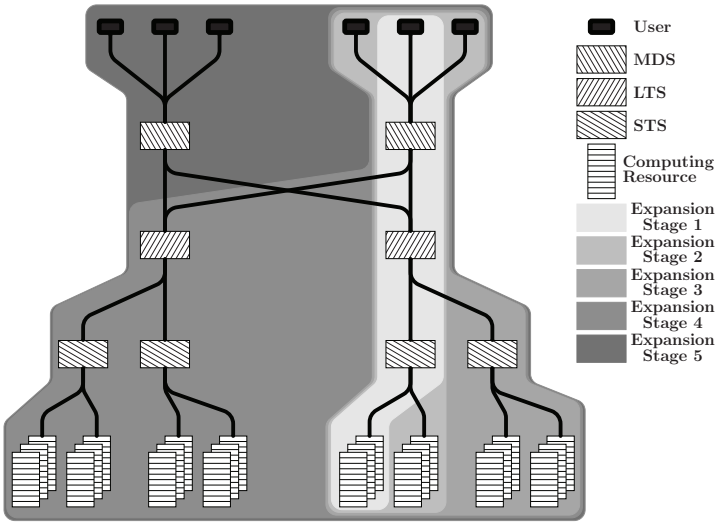


Figure 2: Example of different expansion states of a SLAte installation to handle different amount of users and resources to monitor.

4 Analysis of Scalability

In the following we look into aspects like required storage capacity and network performance for the servers in the three layers of SLAte. We show which criteria influence these aspects in particular the installation of additional servers.

For the analysis we consider a completely symmetrical installation. Every server in a layer has the same performance and each STS server has to handle the same amount of monitoring data. Thus, an STS server represents a computing resource and the load is caused by its job centric monitoring data. As a consequence the number of STS servers defines the overall amount of monitoring data which has to be handled by the SLAte infrastructure. The jobs are also considered as uniform and constant in computing time, measurement interval for monitoring and are equally distributed over the computing resources.

These assumptions are simplifications which only partially reflect the reality. But they ease the scalability analysis of the concepts used by the SLAte infrastructure. In future work we have to test and to verify them in a real installation.

4.1 Storage Capacity

The monitoring data are recorded on computing resources. They consist of the monitoring data and metadata. The amount of metadata k_{S_meta} is about constant. The monitoring data per job depends on the runtime of a job (t_{job}), the amount of monitoring data collected

on an single measurement (S_{single}) and the interval (t_{single}) between two measurements[13]. Due to the uniform jobs, these parameters are fixed and the amount of monitoring data is constant where t_{job}/t_{single} is the number of measurements per job.

$$k_{S\_mon} = S_{single} \cdot \frac{t_{job}}{t_{single}} \tag{1}$$

On an STS server the monitoring data and the according metadata are stored (S_{STS}) as long as the job is running. Afterwards the data is moved to an LTS server. Assuming the worst case scenario that all jobs start and stop at the same time, the storage capability depends on the amount of monitoring data and metadata and on the number of jobs (n_{jobs}) the STS server has to handle:

$$S_{STS} = n_{jobs} \cdot (k_{S\_mon} + k_{S\_meta}) \tag{2}$$

Each STS server sends its data to an LTS server. Thus the data stored on an LTS server depend on the number of STS servers copying data to it (n_{STS}), the data stored on one STS server (equation 2), the time interval the data is stored on an STS server (which is the runtime of a job) and the time the data should be hold on the LTS servers[14] (t_{hold}) which is defined by the configuration of the LTS server:

$$S_{LTS} = n_{STS} \cdot n_{jobs} \cdot (k_{S\_mon} + k_{S\_meta}) \cdot \frac{t_{hold}}{t_{job}} \tag{3}$$

The amount of metadata, stored on a MDS server can be calculated similar to the metadata stored on an LTS server. Instead of monitoring data and metadata only metadata have to be considered but the information of all STS servers (n_{all_STS}) is stored.

$$S_{MDS} = n_{all\_STS} \cdot n_{jobs} \cdot (k_{S\_meta}) \cdot \frac{t_{hold}}{t_{job}} \tag{4}$$

Before we start to interpret equation 2, 3 and 4 the number of STS servers per MDS server is replaced with $n_{STS} = n_{all\_STS}/n_{all\_LTS}$ where n_{all_LTS} is the number of all LTS servers. Also the amount of metadata is replaced with its fraction to the monitoring data k_{mon_meta} ($k_{S\_meta} = k_{S\_mon} \cdot k_{mon\_meta}$).

$$S_{STS} = n_{jobs} \cdot (k_{S\_mon}\,(1 + k_{mon\_meta})) \tag{5}$$

$$S_{LTS} = \frac{n_{all\_STS}}{n_{all\_LTS}} \cdot n_{jobs} \cdot (k_{S\_mon} + (1 + k_{mon\_meta})) \cdot \frac{t_{hold}}{t_{job}} \tag{6}$$

$$S_{MDS} = n_{all\_STS} \cdot n_{jobs} \cdot (k_{S\_mon} \cdot k_{mon\_meta}) \cdot \frac{t_{hold}}{t_{job}} \tag{7}$$

[13] We assume that the measurements are done in constant time intervals. This simplifies the considerations. For more complex interval distribution it is possible to use the minimum or mean time between measurements.

[14] After some time (weeks or month) the monitoring information is considered as outdated and is removed automatically from the LTS servers

By considering that the amount of monitoring data is much larger then the metadata ($k_{S\_mon} \gg k_{mon\_meta}$) and by combining constants ($t_{hold}/t_{job} = k_1$) we get to the following approximations:

$$S_{STS} \approx n_{jobs} \cdot k_{S\_mon} \tag{8}$$

$$S_{LTS} \approx \frac{n_{all\_STS}}{n_{all\_LTS}} \cdot S_{STS} \cdot k_1 \tag{9}$$

$$S_{MDS} \approx n_{all\_STS} \cdot S_{STS} \cdot k_{mon\_meta} \cdot k_1 \tag{10}$$

These formulas show how storage capacity relates to different aspects. In (8) we see that the needed capacity is proportional to the number of jobs which send monitoring data to an STS server.

The amount of data on an LTS server is proportional to the needed storage on an STS server[15] and can be lowered by increasing the number of LTS servers.

In opposite to the servers in the other layers, the needed capacity of the MDS server can not be decreased by installing additional servers (according to (10)). An important impact has k_{mon_meta}. It gives the fraction of metadata to monitoring data and is quite small (about 0.01 to 0.001). This gives a clear limitation because an MDS server has to be able to store all metadata. On other hand we expect that we can handle this limit and do not need to come up with an architecture overcoming this issue.

4.2 Network Capacity for Storing Monitoring Data

In this section we analyse the network capacities needed. Depending on how the network is used we have to consider factors which reflect this. If packages of reasonable size are transferred we expect nearly the maximum network speed. If small packages are used we have to consider a slowdown of k_{N_slow}. This is the case for the input network capacity of the STS servers (N_{STS_in}) where k_2 is a combination of other constant values:

$$N_{STS\_in} = k_{N\_slow} \cdot \frac{n_{jobs}}{t_{job}} \cdot (k_{S\_mon} + k_{S\_meta}) = k_{N\_slow} \cdot k_2 \tag{11}$$

For the output of an STS server the data is repacked to avoid a slow down of the network. The amount of data to be transferred is the same as for the input.

$$N_{STS\_out} = \frac{n_{jobs}}{t_{job}} \cdot (k_{S\_mon} + k_{S\_meta}) = k_2 \tag{12}$$

An LTS server receives data of multiple STS servers (where n_{all_STS}/n_{all_LTS} is the number of STS servers sending data to one LTS server)

$$N_{LTS\_in} = \frac{n_{all\_STS}}{n_{all\_LTS}} \cdot N_{STS\_out} = \frac{n_{all\_STS}}{n_{all\_LTS}} \cdot k_2 \tag{13}$$

[15]The factor $k1$ depends on how long the information is stored and can be expected in the range from 10 to 1000.

and sends the metadata to each MDS server.

$$N_{LTS\_out} = n_{all\_MDS} \cdot \frac{n_{all\_STS}}{n_{all\_LTS}} \cdot k_2 \cdot \frac{k_{mon\_meta}}{1 - k_{mon\_meta}} \tag{14}$$

Based on the fact that $k_{mon\_meta} \ll 1$ and $k_{mon\_meta} > 0$ (metadata is small in comparison to monitoring data) we get to a approximation:

$$N_{LTS\_out} \approx n_{all\_MDS} \cdot \frac{n_{all\_STS}}{n_{all\_LTS}} \cdot k_2 \cdot k_{mon\_meta} \tag{15}$$

A MDS server (N_{MDS_in}) gets input from all LTS servers. For easier interpretation we used the same relations like for (15).

$$N_{MDS\_in} = n_{all\_STS} \cdot k_2 \cdot \frac{k_{mon\_meta}}{1 - k_{mon\_meta}} \approx n_{all\_STS} \cdot k_2 \cdot k_{mon\_meta} \tag{16}$$

Equations (11) and (12) show us that the network load on the STS server is bound by the resource to monitor (k_2) and that input is the more limiting factor. This aspect is represented by k_{N_slow}.

Additional LTS server lower the network demands per server of this layer. This is shown by (13) and (15).

For the MDS server we see similar limitations like for the required storage capacity. We have the factor k_{mon_meta} which reduces the transfer rate, because we only transfer metadata but the network utilisation raises with the overall amount of data delivered through STS servers. Thus, this aspect does not scale.

4.3 Network Capacity for Accessing Monitoring Data

Before the monitoring data can be read they have to be found. Therefore the MDS server is requested by the user to find relevant job monitoring data. Afterwards the data are read from the LTS servers.

The search requests are partitioned over all MDS servers and occur every t_{search}. The requests consist of k_{S_search} data which is about the amount of metadata. The answer is a list of n_{found} jobs found represented by their metadata k_{S_meta}. With these parameters the resulting network load on the MDS server N_{MDS_access} can be calculated:

$$N_{MDS\_access} = \frac{k_{S\_search} + n_{found} \cdot k_{S\_meta}}{n_{all\_MDS} \cdot t_{search}} \tag{17}$$

$$N_{MDS\_access} \approx \frac{k_{mon\_meta} \cdot k_{S\_mon} \cdot (n_{found} + 1)}{n_{all\_MDS} \cdot t_{search}} = \frac{k_{S\_mon\_meta}}{n_{all\_MDS}} \cdot k_3 \tag{18}$$

Where k_3 is defined as $k_3 = \frac{k_{S\_mon} \cdot (n_{found}+1)}{t_{search}}$.

On the LTS server the monitoring data k_{S_mon} and the metadata k_{S_meta} are requested. This happens every t_{search} for n_{found} jobs. Assuming that these data are equally distributed over all LTS servers, the network load on all LTS servers (N_{LTS_access}) is equal.

$$N_{LTS\_access} = \frac{(k_{S\_mon} + k_{S\_meta}) \cdot n_{found}}{n_{all\_LTS} \cdot t_{search}} \tag{19}$$

Considering that $k_{S\_mon} \gg k_{mon\_meta}$ and that each search delivers a huge amount of found jobs ($n_{found} \gg 1$) we can approximate the network load to:

$$N_{LTS\_access} \approx \frac{1}{n_{all\_LTS}} \cdot k_3 \tag{20}$$

Additional MDS or LTS servers reduce the load per server (according to (18) and (20)). The factor $k_{S\_mon\_meta}$ in (18) which is not present in (20) shows that the number of needed MDS servers is lower than the amount of LTS servers.

5 Conclusions and Outlook

We have shown a concept which handles job centric monitoring data in a distributed infrastructure with an uniform access layer. It distinguishes between monitoring data and their metadata. The concept and the idea to take different network infrastructures into account (within a site and between sites) by organising the servers in different layers is implemented by SLAte.

Handling monitoring data in a distributed way allows scalability in terms of storing capacity and network bandwidth by increasing the number of STS and LTS servers.

Implementing an uniform view to all monitoring data by using MDS servers which hold all metadata tends to bottlenecks. This is due to the fact that metadata of all monitored jobs are stored on each MDS server. But the amount of meta data per job is significantly lower in comparison to the monitoring data. This allows us to drive even huge installations.

A possible strategy to overcome the limitations of the MDS is a distributed search strategy. Therefore a component is needed which delegates a search request to all LTS servers and combines the results of the search. Such a component could replace the MDS servers without the need to change LTS and STS. The disadvantage of this strategy is an increased response time caused by the additional steps in the request handling and by the time to wait for the slowest LTS server.

A further topic of our scientific research is the visualisation and analysis of job centric monitoring data. Currently, we use AMon to e.g., display single jobs or to compare jobs by colour coded bars. To support the users in the monitoring our ongoing work is on the detection of abnormal job behaviour by an automatic analysis and comparison of multiple jobs.

References

[ACC+10] J. Andreeva, M. Calloni, D. Colling, F. Fanzago, J. D'Hondt, J. Klem, G. Maier, J. Letts, J. Maes, S. Padhi, S. Sarkar, D.Spiga, P. Van Mulders, and I. Villella. CMS Analysis Operations. *Journal of Physics: Conference Series*, 219(7):072007, 2010.

[BBK+09] Timo Baur, Rebecca Breu, Tibor Klmn, Tobias Lindinger, Anne Milbert, Gevorg Poghosyan, Helmut Reiser, and Mathilde Romberg. An Interoperable Grid Information System for Integrated Resource Monitoring Based on Virtual Organizations. *Journal of Grid Computing*, 7:319–333, 2009. 10.1007/s10723-009-9134-3.

[BHJR10] Holger Brunst, Daniel Hackenberg, Guido Juckeland, and Heide Rohling. Comprehensive Performance Tracking with Vampir 7. In Matthias S. Mller, Michael M. Resch, Alexander Schulz, and Wolfgang E. Nagel, editors, *Tools for High Performance Computing 2009*, pages 17–29. Springer Berlin Heidelberg, 2010. 10.1007/978-3-642-11261-4_2.

[EGM+] Erik Elmroth, Peter Gardfjll, Olle Mulmo, ke Sandgren, and Thomas Sandholm. A Coordinated Accounting Solution for SweGrid Version: Draft 0.1.3 Date: October 7, 2003.

[Fos05] Ian Foster. Globus Toolkit Version 4: Software for Service-Oriented Systems. In Hai Jin, Daniel Reed, and Wenbin Jiang, editors, *Network and Parallel Computing*, volume 3779 of *Lecture Notes in Computer Science*, pages 2–13. Springer Berlin / Heidelberg, 2005. 10.1007/11577188_2.

[HMP10] Marcus Hilbrich and Ralph Müller-Pfefferkorn. A Scalable Infrastructure for Job-Centric Monitoring Data from Distributed Systems. In Marian Bubak, Michal Turala, and Kazimierz Wiatr, editors, *Proceedings Cracow Grid Workshop '09*, pages 120–125, ul. Nawojki 11, 30-950 Krakow 61, P.O. Box 386, Poland, February 2010. ACC CYFRONET AGH.

[MPNB+06] Ralph Müller-Pfefferkorn, Reinhard Neumann, Stefan Borovac, Ahmad Hammad, Torsten Harenberg, Matthias Hüsken, Peter Mättig, Markus Mechtel, David Meder-Marouelli, and Peer Ueberholz. Monitoring of Jobs and their Execution for the LHC Computing Grid. In Marian Bubak, Michal Turala, and Kazimierz Wiatr, editors, *Proceedings Cracow Grid Workshop '06*, pages 224–231, ul. Nawojki 11, 30-950 Krakow 61, P.O. Box 386, Poland, October 2006. ACC CYFRONET AGH.

[PRAGA09] M. Piro Rosario, Guarise Andrea, Patania Giuseppe, and Werbrouck Albert. Using historical accounting information to predict the resource usage of grid jobs. *Future Generation Computer Systems*, 25(5):499–510, 2009.

[RSK+08] Miroslav Ruda, Jiří Sitera, Aleš Křenek, Luděk Matyska, Zděnek Šustr, and Michal Voců. Job Centric Monitoring on the Grid – 7 years of experience with L&B and JP services. *CESNET Conference*, 2008.

[VBC+10] D C Vanderster, F Brochu, G Cowan, U Egede, J Elmsheuser, B Gaidoz, K Harrison, H C Lee, D Liko, A Maier, J T Mocicki, A Muraru, K Pajchel, W Reece, B Samset, M Slater, A Soroko, C L Tan, and M Williams. Ganga: User-friendly Grid job submission and management tool for LHC and beyond. *Journal of Physics: Conference Series*, 219(7):072022, 2010.

I/O-efficient approximation of graph diameters by parallel cluster growing – a first experimental study. *

Deepak Ajwani† Andreas Beckmann‡ Ulrich Meyer‡ David Veith‡

†Centre for Unified Computing
University College Cork
Cork, Ireland

‡Institut für Informatik
Goethe-Universität Frankfurt
Frankfurt am Main, Germany

deepak.ajwani@gmail.com, {beckmann,umeyer,dveith}@cs.uni-frankfurt.de

Abstract: A fundamental step in the analysis of a massive graph is to compute its diameter. In the RAM model, the diameter of a connected undirected unweighted graph can be efficiently 2-approximated using a Breadth-First Search (BFS) traversal from an arbitrary node. However, if the graph is stored on disk, even an external memory BFS traversal is prohibitive, owing to the large number of I/Os it incurs. Meyer [Mey08] proposed a parametrized algorithm to compute an approximation of graph diameter with fewer I/Os than that required for exact BFS traversal of the graph. The approach is based on growing clusters around randomly chosen vertices 'in parallel' until their fringes meet. We present an implementation of this algorithm and compare it with some simple heuristics and external-memory BFS in order to determine the trade-off between the approximation ratio and running-time achievable in practice. Our experiments show that with carefully chosen parameters, the new approach is indeed capable to produce surprisingly good diameter approximations in shorter time. We also confirm experimentally, that there are graph-classes where the parametrized approach runs into bad approximation ratios just as the theoretical analysis in [Mey08] suggests.

1 Introduction

Massive graphs arise naturally in many applications. Social network graphs or the WWW graph have implicitly become part of our daily life. A whole branch of computer science deals with network analysis [BE05]. A fundamental step in the analysis of a massive graph is to compute its diameter: In this paper, we consider connected undirected unweighted large sparse graphs $G(V, E)$ where $n = |V|$ and $m = |E|$. The distance $d(u, v)$ between two nodes $u, v \in V$ is the number of edges in the shortest path connecting u and v. The eccentricity of a node v is defined as $\mathrm{ecc}(v) = \max_u d(v, u)$. Finally, $D = \max_{u,v} d(u, v) = \max_v \mathrm{ecc}(v)$ is called the diameter of G. We are particularly interested in the case when G is sparse ($m = O(n)$) but nevertheless too big to fit into the main memory of a single computing device. In that setting the typical strategies are: (i) to distribute the data over many computers and apply parallel algorithms (e. g., see [BM08, MEJ+09]) and/or (ii) store the data on secondary memory like hard disks or flash memory. In this paper we will concentrate on practically feasible diameter approximation algorithms for the latter (external memory) approach.

*Partially supported by the DFG grant ME 3250/1-3, and by MADALGO – Center for Massive Data Algorithmics, a Center of the Danish National Research Foundation.

External memory model: The huge difference in time between accessing an element from the main memory and fetching an element from the disk is nicely captured by the external memory (EM) model (sometimes also called the I/O model), which was introduced by Aggarwal and Vitter [AV88]. It assumes a two level memory hierarchy. The internal memory is fast, but has a limited size of M elements (nodes/edges). In addition, there is an external memory which can only be accessed using I/Os that move B contiguous elements between internal and external memory. At any particular time, the computation can only use the data already present in the internal memory. The measure of performance of an algorithm is the number of I/Os it performs – the less I/Os the algorithm requires, the better it is. The number of I/Os required to scan n contiguously stored elements in the external memory is $\mathrm{scan}(n) = O(n/B)$ and the number of I/Os required for sorting n elements is $\mathrm{sort}(n) = O(\frac{n}{B} \log_{M/B} n/B)$. For realistic values of B, M and n, $\mathrm{scan}(n) < \mathrm{sort}(n) \ll n$ and the goal of designing external memory algorithms is often to reduce the I/O complexity from $O(n)$ to $O(\mathrm{sort}(n))$. Further discussions on realistic models for computing on large data can be found in a recent survey article by Ajwani and Meyerhenke [AM10].

Outline: Section 2 presents an overview of various algorithms and heuristics designed in recent years to compute the exact or approximate diameter including the approaches implemented in this project. Section 3 provides some implementation details. The experimental results are detailed in Section 4 and we conclude in Section 5.

2 Related Work

The problem of computing the diameter of large graphs, particularly for complex networks, has received considerable attention lately, both from an algorithmic and empirical point of view. For the exact computation of diameters in unweighted undirected graphs, breadth-first search (BFS) can be executed from all nodes of the graph and the longest height of a BFS tree is reported as the diameter. Doing so naively requires $O(n^2+mn)$ operations. However, the method can be improved by logarithmic factors (e. g., [Cha06]) in the standard RAM model with logarithmic word size. There are also algebraic approaches for computing all pairs breadth-first search (AP-BFS) based on matrix-multiplication based algorithm (e. g., [Sei95]). But the exact diameter computation based on either combinatorial or algebraic approaches remains computationally expensive and impractical for massive sparse graphs. Even the recent result by Peres et al. [PSSZ10] who gave an $O(n^2)$ algorithm for computing all-pair shortest path with high probability is infeasible for such graphs.

Internal-Memory Small-Factor Approximations

As the exact computation of diameters for massive sparse graphs seems impractical, approximation algorithms and heuristics have received attention lately. In the following we review some strategies that rely on the fact that a rather small number of BFS computations can easily be afforded as long as the input graph fits into internal memory.

The trivial bounds: It is folklore that already a single BFS run rooted at an arbitrary source r yields trivial lower and upper bounds on the diameter: $\mathrm{ecc}(r) \leq D \leq 2 \cdot \mathrm{ecc}(r)$. Obviously, the choice of r determines which of these bounds is tight (if any). The approximation can be improved by performing k BFS explorations from k carefully chosen starting points [BFLO06]: in that case the additive error drops to $O(n/k)$ at the cost of increased running time (for k BFS runs).

The double sweep heuristic: In practice the trivial lower bound based on a BFS run with some random source r can frequently be improved significantly by just one additional BFS run from some source v' satisfying $d(r, v') = \mathrm{ecc}(r)$ and reporting the bound $d(v', v'') = \mathrm{ecc}(v')$ for some node v'' with maximal depth in the BFS tree for source v'. This technique is also called the *double sweep method* (e. g., see [CDHP01, MLH09]). On certain graph classes including trees the double sweep method even guarantees to yield a tight lower bound on the diameter [CDHP01]. On general graphs, the double sweep lower bound will at least not be worse than the trivial one. In addition, the double sweep method can be iterated for different sources but then previously applied sources for the respective second BFS runs should not be reused. The double sweep lower bound can also be used to yield improved *upper* bounds on the diameter of these BFS trees since it is able to derive the exact diameter of a tree (and hence also of a BFS tree) as mentioned above. Again iterating over several carefully chosen BFS trees may strengthen the upper bound even further. Observe, however, that there are graph classes (like rings) where any BFS tree of those graphs has a larger diameter than the respective original graph (up to a factor of two).

The fringe heuristic: As long as the input graph fits into main memory, the name of the game is to find matching upper and lower bounds for many graph classes while investing as few BFS traversals as possible. To the best of our knowledge, the most efficient approach of this kind is the *fringe* heuristic by Crescenzi et al. [CGI$^+$10]. For some vertex u, the fringe of u, denoted $F(u)$, is set of all vertices $v \in V$ such that $d(u, v) = \mathrm{ecc}(u)$. The fringe heuristic uses the double sweep method to find a lower bound on the diameter and computes an upper bound on the diameter as follows:

1. Let r, v', and v'' be the vertices identified by double sweep method.
2. Find the vertex u that is halfway along the path connecting v' and v'' inside the BFS-tree rooted at v'.
3. Compute the BFS tree for source u and its eccentricity $\mathrm{ecc}(u)$.
4. If $|F(u)| > 1$, find the BFS trees for all sources $z \in F(u)$, and compute $B(u) = \max_{z \in F(u)} \mathrm{ecc}(z)$:
   – If $B(u) = 2 \cdot \mathrm{ecc}(u) - 1$, return $2 \cdot \mathrm{ecc}(u) - 1$.
   – If $B(u) < 2 \cdot \mathrm{ecc}(u) - 1$, return $2 \cdot \mathrm{ecc}(u) - 2$.
5. Return the diameter of the BFS tree rooted at u.

It is shown in [CGI$^+$10] that the fringe algorithm correctly computes an upper bound on the diameter using at most $|F(u)| + 3$ BFS traversals. While $|F(u)| = \Omega(|V|)$ in the worst case, Crescenzi et al. demonstrate that $|F(u)|$ is often rather small (less than 20) for real world graphs. Additionally, for nearly all tested cases in [CGI$^+$10] the fringe heuristic produced matching lower and upper bounds.

External-Memory Approaches

There has been a significant number of publications on external-memory graph algorithms; see [MSSE03, Vit06] for recent overviews. Exact computation of the diameter on unweighted undirected graphs (via All-Pairs Shortest-Paths, APSP) has been addressed in [AMT04, CR05]: both approaches require $\Theta(n \cdot \mathrm{sort}(n))$ I/Os for sparse graphs. Taking into account that current machines easily feature several gigabytes of RAM, in the external-memory setting where $n > M \gg B$, an algorithm spending $\Theta(n \cdot n/B) = \Omega(n^2/B)$ I/Os is practically useless.

Small-factor approximations: Chowdhury and Ramachandran [CR05] also gave an algorithm for computing approximate all-pairs shortest-paths with additive error. However, their approach only takes less I/O than exact EM APSP when $m \geq n \log n$, which is not the sparse graph case we are interested in, and even then the I/O volume is huge.
Of course, the simple BFS-based RAM approximation approaches discussed above can be implemented in external-memory using EM BFS. However, even for the trivial 2-approximation this takes $\Omega(n/\sqrt{B})$ I/Os in the worst-case [MM02]. What this means in practice will be discussed in Section 4.

Parametrized approximations: In the following, we review a recent parametrized approach by Meyer [Mey08] to trade approximation quality with sub-BFS I/O time, which we will also experimentally evaluate in Section 4. The problem of computing an approximate diameter of the input graph G (with n nodes and m edges) is reduced to that of computing exact shortest paths on a weighted graph G' with $O(n/k)$ nodes and $O(m)$ edges. Graph G' is computed using a static external memory BFS [MM02] like preprocessing as follows: We first choose each node to be a master node with a probability $1/k$. Additionally, we select every k-th node in the Euler-tour traversal around an arbitrary spanning tree of G, to also be a master node. Thereafter, we grow the clusters "in parallel". In each round, each master node tries to capture all unvisited neighbors of the current cluster. This is done by first sorting the nodes at the fringes of the clusters and then scanning the adjacency-lists of the nodes in the yet unexplored graph. Ties are broken arbitrarily.
Let $C(u)$ be the cluster containing u. An edge $\{u, v\} \in G$ results in an edge $\{C(u), C(v)\} \in G'$ if $C(u) \neq C(v)$. The weight of the created edge $\{C(u), C(v)\}$ is $d_c(u) + 1 + d_c(v)$, where $d_c(u)$ is the distance of u from its cluster center. We remove the parallel edges by keeping only the lightest edge between $C(u)$ and $C(v)$. We run single source shortest path (SSSP) from an arbitrary node s in G' and output the maximum distance from s to any other node in G'. Note that this is a constant-factor approximation to the weighted diameter of G'. Meyer [Mey08] showed that the expected weighted diameter of G' satisfies $D_{G'} = O(\sqrt{k} \cdot D_G)$.
Since each k-th node on the Euler tour is a master node, each node $u \in G$ is at most distance k away from a master node and the clusters are grown for at most k rounds. As each cluster growing round requires $O(\mathrm{scan}(m))$ I/Os to scan the adjacency lists of unexplored graph and each node appears only once as a fringe node of some cluster leading to a total of $O(\mathrm{sort}(n))$ I/Os, the total complexity of computing G' is $O(k \cdot \mathrm{scan}(n+m) + \mathrm{sort}(n+m) + ST(n,m))$ I/Os, where $ST(n,m)$ is the I/O complexity of computing a spanning tree of an n node and m edge undirected graph: $O(\mathrm{sort}(n+m))$ I/Os randomized [ABW02]

and $O(\mathrm{sort}(n) \log\log \frac{n \cdot B}{m})$ I/Os with a deterministic spanning tree algorithm [ABT04]. Computing single source shortest path on a graph with $O(n/k)$ nodes and $O(m)$ edges with the ratio between maximum and minimum edge weight being k requires $O(\sqrt{\frac{n \cdot m}{k \cdot B} \log_2 k} + \mathrm{sort}(n+m) + ST(n,m))$ I/Os [MZ03]. The total I/O complexity for this algorithm is thus $O(\sqrt{(\frac{n \cdot m}{k \cdot B} \log_2 k} + k \cdot \mathrm{scan}(n+m) + \mathrm{sort}(n+m) + ST(n,m))$ I/Os.

Spanning tree heuristics: While the parametrized approximation discussed above still offers some (expected) approximation guarantees one might also go to the extreme: omit any kind of guarantee and just rely on an I/O-efficient heuristic. While (at least in theory) BFS computations on sparse graphs tend to spend much more I/O than spanning tree computations it is natural to ask if one could use a spanning tree rather than a BFS traversal for approximating the diameter. Unfortunately, the diameter of a spanning tree can be very far from the diameter of the graph. For instance, consider a cycle graph $u_1 \ldots u_{n-1}$ of $n-1$ nodes and edges and a special node s with edges to all nodes of the cycle. The diameter of a spanning tree $\{s, u_1\}, \{u_1, u_2\}, \ldots, \{u_{n-1}, u_n\}$ is $n-1$ while the diameter of the original graph is 2. Unfortunately, even the diameter of a random spanning tree can be very far away from the diameter of the graph. For instance, Rényi and Szekeres [RS67] showed that the expected diameter of a random spanning tree in the complete graph K_n is $O(\sqrt{n})$.
A simpler way to use randomization in this context is to consider the minimum spanning tree in the original graph with edge weights assigned independently and uniformly from the range $(0, 1]$. However, even the diameter of such a spanning tree can be quite large compared to the diameter of the graph. Nevertheless, our heuristic based on initial work by Brudaru [Bru07] takes this spanning tree as the base case and iteratively refines it to approximate a BFS tree rooted at some arbitrary node s. Let T_i be the spanning tree after the i-th iteration (minimum spanning tree with random weights being T_0) and $h_i(u)$ be the distance of node u from s in T_i. Each iteration consists of carefully selecting the edges for T_{i+1} such that for each node u, $h_{i+1}(u) \leq h_i(u)$ and for at least one node v, $h_{i+1}(v) < h_i(v)$, thus, eventually the sequence $T_0, T_1, \ldots, T_i$ converges to a BFS tree with root s.

An iteration consists of scanning the list of edges of the original graph and for each node u, selecting the edge $\{v, u\}$ such that $h_i(v)$ is minimum. This is done independently for all nodes. Note that although some neighbors of u may have found a shorter path to s in the course of this iteration, this is completely ignored as using this information naively may require random I/Os.

3 Implementation details

For our experimental study we implemented or modified three approaches: (i) we modified the external memory BFS [ADM06] to use double sweep lower bound, (ii) we reimplemented the spanning tree heuristics from Section 2, and (iii) we engineered a simplified version of the parametrized approximation algorithm from Section 2. Our C++ code uses the external memory library STXXL [DKS08] ver. 1.3.1 for algorithms like sorting

and data structures like priority queues. An additional benefit of using STXXL is the streaming interface of various algorithms that allows us to make extensive use of pipelining to save a factor of 2–3 in the total I/O volume.

Implementation of the parametrized approximation: The data structures and graph generators used in our code are similar to those of BFS implementation of Ajwani et al. [ADM06]. This was done to ensure better comparison with the external memory BFS implementations. Also, we use their pipelined randomized clustering implementation in our approach to compute the clustering of the input graph.
Notwithstanding the theoretical description of the parametrized approximation approach in [Mey08] and Section 2 we omitted to choose extra masters deterministically and only rely on the randomly chosen master vertices. We then create a condensed graph based on this clustering using a constant number of sorting and scanning rounds. If the condensed graph fits internally, we use an internal memory SSSP sub-routine that we implemented using STL. Otherwise, we use our adaptation of the *semi*-external memory SSSP sub-routine by Meyer and Osipov [MO09], SE_SSSP for short, to compute the diameter of the weighted condensed graph. Note that this code requires at least one bit per vertex of the condensed graph in internal memory and also drops strict performance guarantees for non-random edge weights, which occur in our application. We refer to our implementation of the parameterized approximation algorithm as PAR_APPROX.
Both SSSP-subroutines (internal or semi-external) apply the double sweep lower bound ideas [MLH09] to find a source which guarantees reasonable values for the resulting diameter.

Implementation of the spanning tree heuristics: In her original work [Bru07], Brudaru implemented these heuristics in internal memory using the LEDA library and also created an external memory prototype. We re-implemented the heuristics to maximally utilize the pipelining and other features offered by STXXL.

4 Experimental results

The goal of our experiments is (i) to determine the trade-off between approximation quality and running time (dominated by I/Os) and (ii) to ascertain if the diameter for massive sparse graphs can be computed in a reasonable time (e. g. overnight running on a standard desktop PC). Our hope is that the insights learned during these experiments will assist a practitioner to determine the right technique for computing the diameter of a given graph.

Graph classes: We chose three different graph classes: one real-world graph with logarithmic diameter and two synthetic graph classes with diameter $\Theta(\sqrt{n})$ and $\Theta(n)$. The real-world graph sk-2005 has around 50 million vertices, about 1.8 billion edges and is based on a web-crawl. It was selected for better comparison with DSLB_UP_BOUND of Crescenzi et al. [CGI$^+$10] and because it has a known diameter of 40.
The synthetic x-level graphs are similar to the B-level random graphs in [ADM06]. The graph consists of x levels, each having $\frac{n}{x}$ vertices (except for level 0 which contains only one vertex). The edges are randomly distributed between consecutive levels, such that

these x levels approximate the BFS levels if BFS were performed from the source vertex in level 0. We selected $x = \sqrt{n}$ and $x = \Theta(n)$ to generate $\sqrt{n}$ and $\Theta(n)$-level graphs with 2^{28} vertices and around 1 billion edges for our experiments. The generated graphs have 1,127,310,556 edges ($\sqrt{n}$-level graph) and 903,876,452 ($\Theta(n)$-level graph).
To elicit the worst-case approximation ratio from the PAR_APPROX approach, we also generated another graph from a class with three parameters: k_1, k_2 and k_3 satisfying $n = k_3 \cdot (k_1 + k_2)$. It consists of a list of length k_3. There are k_3 node-disjoint lists of length k_1 incident on each vertex of the original list (of size k_3). Each of the k_3 lists have a fan-out of k_2 at the other end. The main idea behind this graph class is as follows: for appropriately chosen k_j values, there are most masters of PAR_APPROX in the fans and only few masters in the lists so that many clusters meet with large edge weights at the original list of length k_3, thus blowing up the weighted diameter of the condensed graph significantly.
We randomize the layout of the synthetic graphs on the disk to ensure that the disk layout does not reveal any additional information that is exploitable. However, we use the ordering provided with sk-2005 graph for fair comparison with results reported in the literature.

Configuration: We performed our experiments on two different architectures.
(i) To determine the behavior of different techniques in an external memory setting, we used a machine with an Intel dual core E6750 processor @ 2.66 GHz, 4 GB internal memory (around 3.5 GB free), 4 hard-disks with 500 GB each as external memory for STXXL, and a separate disk for the operating system, application and storing data, logfiles etc. The operation system was Debian GNU/Linux amd64 'wheezy' (testing) with kernel 3.0. The programs were compiled with GCC 4.4 in C++0x mode using optimization level 3.
(ii) For running the heuristics of Crescenzi et al. [CGI$^+$10] in internal memory, we used a machine (part of the HPC cluster at Goethe University) with 4 quad-core AMD Opteron™ processor 8384 @ 2.7 GHz (only one core was used) and 64 GB internal memory. Note that the purpose of these experiments is to determine the quality of approximation with their approach and to ascertain if it can be matched in an external memory setting. The running time of an approach on this machine is *no* indication of the running time in an external memory setting.

Selecting the correct number of master vertices for PAR_APPROX: Theoretically, as we increase the number of master vertices, the maximum distance d between them should decrease. This should help to reduce the running time of the clustering phase and improve the approximation quality (as the condensed graph better captures the structure of the original graph and the factor $2 \cdot d$ added to the calculated diameter is low). The penalty paid for this is the increased I/O time for external memory SSSP. Varying the number of master vertices gives a trade-off between approximation ratio and the running time.
However, most worst-case efficient external memory SSSP approaches are impractically sophisticated. As such, we have to rely on internal and semi-external memory SSSP (SE_SSSP). This imposes additional constraints on the maximum number of master vertices as for using the internal memory SSSP, the condensed graph should fit internally, while for using SE_SSSP the number of master vertices should be less than the main memory size in bits.
When using the internal memory SSSP, we would like to choose the largest number of

master nodes such that the condensed graph fits internally. However, identifying this number is a non-trivial task because the graph density of the condensed graph depends on the structure of the input graph. The condensed graph of a random graph is significantly more dense than the one for a list graph. Therefore, we would like to select the number of master vertices based on the structure of the graph – fewer master vertices for random graphs than for $\Theta(n)$-level graphs. However, selecting the number of master vertices on the basis of graph type requires a priori information about the graph class, which runs contrary to our objective of analyzing a given graph by determining its diameter.
Thus, we have two alternatives: We can either start with a small (e. g., $O(\sqrt{M})$) number of master vertices such that the condensed graph is guaranteed to fit internally. Thereafter, we can adaptively compute the correct number of master vertices by increasing the number if the running time for the clustering is too high (and aborting and redoing the run with the new number) or by decreasing the number if the resultant condensed graph does not fit internally.
The other alternative is to choose a large number of master vertices and use SE_SSSP on the condensed graph. This is almost always faster than EM_BFS_DSLB. We can then reduce the number of master vertices to get a trade-off between approximation quality and runtime. We have used both of these alternatives in our experimental study.

Results: First we present the results of three different approaches: External memory BFS with double sweep lower bound (EM_BFS_DSLB), the spanning tree heuristic (SPAN) and the fringe approach from Crescenzi et al. with dslb method [CGI+10] (DSLB_UP_BOUND). The external BFS and the internal DSLB_UP_BOUND showed similar results. For sk-2005 we got a lower bound of 39 instead of 40. With a second experiment with a different carefully chosen source we have found the lower bound of 40, too.
For DSLB_UP_BOUND, we used ten iterations in one experiment. The other applications we executed with only one iteration.
The results of the SPAN heuristic were not that close to the real diameter and have a taller spread. The numbers in Table 1 are the resulting heights of the trees multiplied with factor of two as an upper bound. We do not report the detailed running times of

	EM_BFS_DSLB	SPAN	DSLB_UP_BOUND
sk-2005	39	60	**40**
$\sqrt{n}$-level graph	**16,385**	46,262	**16,385**
$\Theta(n)$-level graph	**67,108,864**	86,488,096	**67,108,864**
worst case for PAR_APPROX	**2,440,341**	3982472	**2,440,341**

Table 1: Diameters approximated by various approaches. Exact diameters are marked in boldface.

DSLB_UP_BOUND, since it was only executed on the big 64 GB internal memory machine and was merely used to get hold of the exact diameters. As can be seen in Table 2, the SPAN heuristic was often slower than EM_BFS_DSLB. Only for the $\sqrt{n}$-level graph we obtained a better running time behavior. In correspondence with the results in [ADM06], graphs sk-2005 (with a low diameter) and $\Theta(n)$-level (which is similar to a single chain) are easier for external BFS than the $\sqrt{n}$-level graph (which shares some characteristics with grid graphs).

Results for the PAR_APPROX implementation: To learn more about the behavior of our new approach we ran a couple of different test scenarios. In Table 3 we report on the

	EM_BFS_DSLB	SPAN
sk-2005	5.27	7.65
$\sqrt{n}$-level graph	10.64	7.74
$\Theta(n)$-level graph	4.75	4.81
worst case for PAR_APPROX	1.66	3.34

Table 2: Running times (in hours) for EM_BFS_DSLB and SPAN.

results when the condensed graph G' fits into internal memory. The running times are dominated by the clustering. The internal-memory SSSP for G' usually took only a few seconds.

As for sk-2005, PAR_APPROX was up to 10 times faster than EM_BFS_DSLB and SPAN. Interestingly, the best approximation guarantee (42 vs. 40 exact) was obtained for small numbers of master nodes. Running time and approximation deteriorate with more masters. We verified this phenomenon on the machine with 64 GB internal memory and more samples. When between ten and twenty percent of the original graph nodes are chosen as master vertices for sk-2005, there is a point where the approximation bound improves again. We saw this behavior more or less pronounced for other real world graphs tested in [CGI$^+$10], too.

The results for the $\Theta(\sqrt{n})$-level graph showed just the opposite trend: best running times (up to 12 times faster than EM_BFS_DSLB) and approximation bounds (only 0.14 % away from the exact diameter but nearly three times better than SPAN) were obtained for the largest possible number of masters.

While for graphs with smaller diameter like sk-2005 a small number of master vertices like 2^{14} produces good running times, for the $\Theta(n)$-level graph applying too few master nodes would be dangerous: with a diameter of over 900 millions and only 2^{14} master nodes, more than 50,000 phases of the parallel cluster growing would be needed. The estimated time for this clustering is around a month. Fortunately, the condensed graph G' from $\Theta(n)$-level graph is rather small even for a high number of master vertices: G' fits into internal memory on a 4 GB machine until 2^{25} master vertices. While the approximation guarantee is still within 1 %, the running time gain for 2^{24} masters is only a factor of four.

Expectedly bad approximations bounds could be identified for our worst-case graph, especially with parameters $k_1 = 10$ and $k_2 = 100$ for $n = 2^{28}$, resulting in a diameter of about 2.44 million for $n = 2^{28}$. With 2^{20} master nodes, the PAR_APPROX overestimated the real diameter by more than a factor of five.

As can be seen in Table 3 for high diameter graph classes we need – and can afford – a lot of master vertices, but for $\Theta(\sqrt{n})$-level inputs or $\Theta(n)$-level inputs the resulting condensed graphs are too dense to be used in internal memory. Hence, we tested the behavior of PAR_APPROX for three different numbers of master vertices with SE_SSSP. The results in Table 4 show that also with SE_SSSP as a subroutine PAR_APPROX is faster than EM_BFS_DSLB but not very much. Nevertheless, what is also important, the resulting diameters are still very close to the real diameters.

PAR_APPROX vs. SPAN vs. EM_BFS_DSLB: If the quality of the diameter approximation is most important then EM_BFS_DSLB could be the first choice. EM_BFS_DSLB produced reasonable results for all three graph classes sk-2005, $\Theta(\sqrt{n})$-level, and $\Theta(n)$

masters	$\sim 2^8$	$\sim 2^{10}$	$\sim 2^{12}$	$\sim 2^{14}$	$\sim 2^{16}$	$\sim 2^{18}$	$\sim 2^{20}$	$\sim 2^{22}$	$\sim 2^{24}$
sk-2005									
computed approx. diameter	42	51	68	79	106	113	98	119	
approximation ratio	1.05	1.28	1.70	1.98	2.65	2.83	2.45	2.98	
time [h]	0.46	0.51	0.62	0.68	0.73	0.76	0.78	0.77	
d	12	13	17	17	17	22	18	16	
$\sqrt{n}$-level									
computed approx. diameter		16,836	16,519	16,413	16,409	16,408			
approximation ratio		1.0275	1.0082	1.0017	1.0015	1.0014			
time [h]		4.29	1.30	0.87	0.87	0.85			
d		156	35	15	13	12			
$\Theta(n)$-level									
diameter						67,118,479	67,128,342	67,233,297	67,717,702
approx. ratio						1.00014	1.00029	1.00185	1.00907
time [h]						41.60	12.33	3.68	1.26
d						3444	814	233	60
worst case									
computed approx. diameter					3,643,615	6,729,783	13,461,919	11,265,297	4,399,657
approximation ratio					1.49	2.76	5.52	4.62	1.80
time [h]					5.12	1.87	0.92	0.73	0.58
d					449	144	50	28	22

Table 3: Results of PAR_APPROX for different numbers of master vertices, when the condensed graph fits into internal memory. d is the maximum distance between two master vertices.

masters	$\sim 2^{22}$	$\sim 2^{24}$	$\sim 2^{26}$
sk-2005			
computed approx. diameter	119	90	
approximation ratio	2.98	2.25	
time [h]	1.09 (0.32)	2.78 (1.73)	
d	16	13	
vertices in G'	4,193,085	16,774,091	
edges in G'	58,008,681	298,868,555	
$\sqrt{n}$-level graph			
computed approx. diameter	16,407	16,403	16,401
approximation ratio	1.0013	1.0011	1.0010
time [h]	6.96 (5.51)	8.38 (6.80)	9.41 (7.72)
d	10	8	7
vertices in G'	4,195,701	16,774,408	67,105,247
edges in G'	702,067,655	858,741,691	924,712,355
$\Theta(n)$-level graph			
computed approx. diameter	67,233,297	67,717,702	67,515,826
approximation ratio	1.00185	1.00907	1.00606
time [h]	3.94 (0.06)	1.6 (0.29)	2.47 (1.68)
d	233	60	16
vertices in G'	4,195,701	16,774,408	67,105,247
edges in G'	4,305,371	21,392,325	155,525,706

Table 4: Results of PAR_APPROX with semi-external SSSP. The running times for SE_SSSP (times for clustering with BFS Phase I, sending weights to edges, 2 x SSSP) are reported in brackets.

level. It would have done so for the (k_1, k_2, k_3) worst-case graph too, since that graph is just a tree. But if the running time is also important then rather PAR_APPROX should be chosen. It is faster than EM_BFS_DSLB in each tested case (sometimes more than a factor of ten) and its diameter approximation for each graph class was typically rather close – except for the carefully constructed worst-case graph. The spanning tree heuristic, however, could not convince in this test.

5 Conclusion

Our experiments have shown that the parametrized diameter approximation method is in fact faster than plain external-memory BFS and typically produces much better approximation bounds than the theory predicts. Nevertheless, it turns out that it is currently not suited as a section guide between different BFS approaches: as soon as the condensed graph does not fit into main memory, the overhead to run the semi-external memory SSSP is not worth the subsequent savings of a carefully chosen BFS approach. In fact, this is mostly a problem of the current interface in our SE_SSSP implementation, which causes some extra sorting steps for data conversion. For the future we plan to improve the SE_SSSP code in order to avoid these losses.
We are also interested in more sophisticated methods to condense the input graph. Currently, more master vertices speed-up the reduction time but result in a condensed graph that typically does not fit into main-memory, thus causing more I/O for the subsequent SSSP. Hierarchical clustering seems to be the natural choice but as the condensed graphs become weighted already after the first round, the parallel cluster growing of the next rounds needs to appropriately handle these weights, too. Currently, this step relies on the fact that the edges are unweighted.

References

[ABT04] L. Arge, G. Brodal, and L. Toma. On external-memory MST, SSSP and multi-way planar graph separation. *J. Algorithms*, 53(2):186–206, 2004.

[ABW02] J. Abello, A. Buchsbaum, and J. Westbrook. A functional approach to external graph algorithms. *Algorithmica 32(3)*, pages 437–458, 2002.

[ADM06] D. Ajwani, R. Dementiev, and U. Meyer. A computational study of external memory BFS algorithms. In *Proceedings of the Seventeenth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 601–610, 2006.

[AM10] D. Ajwani and H. Meyerhenke. Realistic Computer Models. In *Algorithm Engineering*, volume 5971 of *LNCS*, pages 194–236. Springer, 2010.

[AMT04] L. Arge, U. Meyer, and L. Toma. External Memory Algorithms for Diameter and All-Pairs Shortest-Paths on Sparse Graphs. In *Proc. 31st ICALP*, volume 3142 of *LNCS*, pages 146–157. Springer, 2004.

[AV88] A. Aggarwal and J. S. Vitter. The input/output complexity of sorting and related problems. *Communications of the ACM, 31(9)*, pages 1116–1127, 1988.

[BE05] U. Brandes and T. Erlebach, editors. *Network Analysis: Methodological Foundations*, volume 3418 of *LNCS*. Springer, 2005.

[BFLO06] K. Boitmanis, K. Freivalds, P. Ledins, and R. Opmanis. Fast and Simple Approximation of the Diameter and Radius of a Graph. In *Proc. 5th WEA*, volume 4007 of *LNCS*, pages 98–108. Springer, 2006.

[BM08] D. A. Bader and K. Madduri. SNAP, Small-world Network Analysis and Partitioning: An open-source parallel graph framework for the exploration of large-scale networks. In *Proc. 22nd IEEE International Symposium on Parallel and Distributed Processing (IPDPS)*, pages 1–12. IEEE, 2008.

[Bru07] I. I. Brudaru. Heuristics for Average Diameter Approximation with External Memory Algorithms. Master's thesis, Universität des Saarlandes, October 2007.

[CDHP01] D. G. Corneil, F. F. Dragan, M. Habib, and C. Paul. Diameter determination on restricted graph families. *Discrete Applied Mathematics*, 113(2-3):143–166, 2001.

[CGI+10] P. Crescenzi, R. Grossi, C. Imbrenda, L. Lanzi, and A. Marino. Finding the Diameter in Real-World Graphs – Experimentally Turning a Lower Bound into an Upper Bound. In *Proceedings of the 18th annual European Symposium on Algorithms (ESA)*, volume 6346 of *LNCS*, pages 302–313. Springer, 2010.

[Cha06] T. M. Chan. All-pairs shortest paths for unweighted undirected graphs in *O(mn)* time. In *Proceedings of the Seventeenth Annual ACM-SIAM Symposium on Discrete Algorithms SODA*, pages 514–523, 2006.

[CR05] R. Chowdury and V. Ramachandran. External-Memory Exact and Approximate All-Pairs Shortest-Paths in Undirected Graphs. In *Proc. 16th Ann. Symposium on Discrete Algorithms (SODA)*, pages 735–744. ACM-SIAM, 2005.

[DKS08] R. Dementiev, L. Kettner, and P. Sanders. STXXL: Standard Template library for XXL data sets. *Software: Practice and Experience*, 38(6):589–637, 2008.

[MEJ+09] K. Madduri, D. Ediger, K. Jiang, D. A. Bader, and D. G. Chavarría-Miranda. A faster parallel algorithm and efficient multithreaded implementations for evaluating betweenness centrality on massive datasets. In *Proc. 23rd IEEE International Symposium on Parallel and Distributed Processing (IPDPS)*, pages 1–8. IEEE, 2009.

[Mey08] U. Meyer. On Trade-Offs in External-Memory Diameter-Approximation. In *Proc. of the 11th Scandinavian Workshop on Algorithm Theory (SWAT)*, pages 426–436, 2008.

[MLH09] C. Magnien, M. Latapy, and M. Habib. Fast computation of empirically tight bounds for the diameter of massive graphs. *Journal of Experimental Algorithmics*, 13:1.10–1.9, 2009.

[MM02] K. Mehlhorn and U. Meyer. External-Memory Breadth-First Search with Sublinear I/O. In *Proceedings of the 10th annual European Symposium on Algorithms (ESA)*, volume 2461 of *LNCS*, pages 723–735. Springer, 2002.

[MO09] U. Meyer and V. Osipov. Design and Implementation of a Practical I/O-efficient Shortest Paths Algorithm. In *Proceedings of the annual conference on Algorithm Engineering and Experiments (ALENEX)*, pages 85–96. SIAM, 2009.

[MSSE03] U. Meyer, P. Sanders, and J. Sibeyn (Eds.). *Algorithms for Memory Hierarchies*, volume 2625 of *LNCS*. Springer, 2003.

[MZ03] U. Meyer and N. Zeh. I/O-Efficient Undirected Shortest Paths. In *Proceedings of the 11th Annual European Symposium on Algorithms (ESA'03)*, volume 2832 of *LNCS*, pages 434–445. Springer, 2003.

[PSSZ10] Y. Peres, D. Sotnikov, B. Sudakov, and U. Zwick. All-Pairs Shortest Paths in $O(n^2)$ Time with High Probability. In *FOCS*, pages 663–672, 2010.

[RS67] A. Rényi and G. Szekeres. On the height of trees. *Journal of the Australian Mathematical Society*, 7:497–507, 1967.

[Sei95] R. Seidel. On the All-Pairs-Shortest-Path Problem in Unweighted Undirected Graphs. *Journal of Computer and System Sciences*, 51(3):400–403, 1995.

[Vit06] J. S. Vitter. Algorithms and Data Structures for External Memory. *Foundations and Trends in Theoretical Computer Science*, 2(4):305–474, 2006.

The Spectral Relation between the Cube-Connected Cycles and the Shuffle-Exchange Network

Christian Riess, Volker Strehl, Rolf Wanka
Department of Computer Science, University of Erlangen-Nuremberg, Germany
{christian.riess, strehl, rwanka}@informatik.uni-erlangen.de

Abstract: We investigate the relation between the spectral sets (i. e., the sets of eigenvalues, disregarding multiplicities) of two d-dimensional networks popular in parallel computing: the Cube-Connected Cycles network $\mathrm{CCC}(d)$ and the Shuffle-Exchange network $\mathrm{SE}(d)$. We completely characterize their spectral sets. Additionally, it turns out that for any odd d, the $\mathrm{SE}(d)$-eigenvalues set is precisely the same as the $\mathrm{CCC}(d)$-eigenvalues set. For any even d, however, the $\mathrm{SE}(d)$-eigenvalues form a proper subset of the set of $\mathrm{CCC}(d)$-eigenvalues.

1 Introduction

Background. Popular *Hypercubic* networks used as parallel machines are the Butterfly network, the Cube-Connected Cycles network, the Shuffle-Exchange network and the DeBruijn network. For a collection of their properties and many algorithms for them, see, e. g., [Lei92]. In particular, these constant-degree networks are able to execute so-called *normal* hypercube algorithms with only constant slowdown if compared to the execution time on the hypercube which has non-constant degree ([Lei92]).

Among the characteristic parameters of networks, the eigenvalues of their adjacency matrices are very important (e. g., see [CDS95, Chu97, BK05] for comprehensive studies). They reflect many structural properties of the network. For instance, from the eigenvalues it can immediately be decided whether the network is bipartite (see Proposition 4 below). Expansion properties, bisection problems, the mixing time of Markov chains and the computation of the isoperimetric number [DT98, Bül97] are fields of application of eigenvalues in algorithmic graph theory.

In the area of parallel computing, there is a direct connection between the eigenvalues and the routing number [ACG94]. Further applications can be found in the analysis of parallel load-balancing algorithms [RSW98] and in the design of interconnection networks [EKM03].

The set of eigenvalues is called the *spectral set*. In the *spectrum* of a graph, additionally the multiplicities of the eigenvalues are considered. For formal definitions, see Subsec. 2.2. Previously, only the full spectral sets of the DeBruijn network [DT98] and the two variants of the Butterfly network [EKM03, Sch01] have been known.

New results. In this paper, we exactly characterize the spectral sets of the Cube-Connected Cycles $\mathrm{CCC}(d)$ and the Shuffle-Exchange network $\mathrm{SE}(d)$ in terms of the spectra of cycles with self-loops that have weights from $\{-1, +1\}$ (see Theorems 1 and 2).

It turns out (see Theorem 3) that for any odd d the *set* of the $\mathrm{SE}(d)$-eigenvalues is precisely the same as the *set* of the $\mathrm{CCC}(d)$-eigenvalues. For any even d, however, the $\mathrm{SE}(d)$-eigenvalues form a *proper* subset of the set of $\mathrm{CCC}(d)$-eigenvalues. The odd case is particularly remarkable because the networks differ in the number of vertices by a factor of d, and hence, the eigenvalues have different multiplicities. Also, there is no obvious way of identifying the eigenvalues bijectively. In fact, corresponding eigenvalues can only be found on scattered cycles of the networks, and a new argument on the involved eigenspaces is necessary in order to find all $\mathrm{CCC}(d)$-eigenvalues in the eigenvalue set of $\mathrm{SE}(d)$, if d is odd. For an instructive (counter-)example for $d = 6$, see Sec. 6.

If d is even, -3 that is an eigenvalue of $\mathrm{CCC}(d)$ is not an eigenvalue of $\mathrm{SE}(d)$. In fact, when d becomes larger, the size of the difference set increases. Let $\Delta_d = |\mathrm{SpS}(\mathrm{CCC}(d)) \setminus \mathrm{SpS}(\mathrm{SE}(d))|$ denote the number of eigenvalues of $\mathrm{CCC}(d)$ that are not eigenvalues of $\mathrm{SE}(d)$. In terms of Δ_d, the result of this paper can be stated as: if d is odd, $\Delta_d = 0$, and $\Delta_d \geq 1$, if d is even. Explicit computation shows for even d, $d \leq 20$:

d	4	6	8	10	12	14	16	18	20
Δ_d	1	3	1	9	7	42	21	179	160

Known results. We briefly mention some known spectral sets, denoted by $\mathrm{SpS}(.)$:

Let $L(n_1, \ldots, n_d)$ denote the d-dimensional $n_1 \times \cdots \times n_d$-array. Then

$$\mathrm{SpS}(L(n_1, \ldots, n_d)) = \left\{ 2 \sum_{i=1}^{d} \cos\left(\frac{\pi j_i}{n_i + 1}\right) \;\middle|\; 1 \leq j_i \leq n_i \text{ for } i \in \{1, \ldots, d\} \right\} .$$

In the following, $L_n := L(n)$ denotes the linear array of length n.

Let $\Theta(n_1, \ldots, n_d)$ denote the d-dimensional $n_1 \times \cdots \times n_d$-torus. Then

$$\mathrm{SpS}(\Theta(n_1, \ldots, n_d)) = \left\{ 2 \sum_{i=1}^{d} \cos\left(\frac{2\pi j_i}{n_i}\right) \;\middle|\; 0 \leq j_i \leq n_i - 1 \text{ for } i \in \{1, \ldots, d\} \right\} .$$

As the adjacency matrices of tori are block-circulant there is a comparatively simple way to compute their spectra (see Proposition 1 in Subsec. 2.2). In the following, $C_n := \Theta(n)$ denotes the cycle of length n.

Interestingly, the spectral sets of other popular networks can be expressed in terms of linear arrays L_n and cycles C_n (in the following, $a \cdot M_k$ denotes the product of the adjacency matrix M_k and the scalar a). For the d-dimensional Butterfly network $\mathrm{BF}(d)$ (for proofs, see [Sch01, EKM03]),

$$\mathrm{SpS}(\mathrm{BF}(d)) = \bigcup_{k=0}^{d+1} \mathrm{SpS}(2L_k) .$$

Similarly, for the Butterfly network with wrap-around edges [Sch01, EKM03, CFGM03], $\mathrm{SpS}(\text{W-BF}(d)) = \mathrm{SpS}(2C_d) \cup \bigcup_{k=0}^{d} \mathrm{SpS}(2L_k)$. Let $\mathrm{DB}(D,d)$ denote the D-ary d-dimensional DeBruijn graph. Then (for a proof, see [DT98]),

$$\mathrm{SpS}(\mathrm{DB}(D,d)) = \mathrm{SpS}(D \cdot C_1) \cup \bigcup_{\tau=1}^{d} \mathrm{SpS}(D \cdot L_\tau) \ .$$

Organization of paper. The paper is organized as follows: In the next section, we define the networks to be investigated, give the necessary definitions regarding graph spectra, and state important properties. In Sections 3 and 4, we exactly characterize the spectra of the Cube-Connected Cycles network and the Shuffle-Exchange network. In Sec. 5, we prove that, if d is odd, the sets of eigenvalues are identical, whereas, if d is even, the set of eigenvalues of $\mathrm{SE}(d)$ is a proper subset of the set of eigenvalues of $\mathrm{CCC}(d)$. That there is no simple correspondence between the eigenvalues of $\mathrm{CCC}(d)$ and $\mathrm{SE}(d$ is exemplified in Sec. 6.

2 Preliminaries

In this section, we introduce the Cube-Connected Cycles network and the Shuffle-Exchange network. We present some of their properties, present tools for computing their eigenvalues, and introduce some necessary notations.

2.1 CCC(*d*), SE(*d*), and Their Properties

The d-dimensional *Cube-Connected Cycles* network $\mathrm{CCC}(d)$ has been introduced by Preparata and Vuillemin in [PV81]. It is the undirected graph with vertex set $V = \{(j,\mathbf{a}) \mid 1 \le j \le d,\ \mathbf{a} \in \{0,1\}^d\}$ and edge set $E = \{\{(j,\mathbf{a}),((j \bmod d+1,\mathbf{a})\} \mid 1 \le j \le d, \mathbf{a} \in \{0,1\}^d\} \cup \{\{(j,\mathbf{a}),(j,\mathbf{a}[j])\} \mid 1 \le j \le d, \mathbf{a} = (a_d,\ldots,a_j,\ldots,a_1) \in \{0,1\}^d, \mathbf{a}[j] = (a_d,\ldots,1-a_j,\ldots,a_1)\}$. $\mathrm{CCC}(d)$ has $d \cdot 2^d$ vertices and is 3-regular.

The d-dimensional *Shuffle-Exchange* network $\mathrm{SE}(d)$ has been introduced by Stone [Sto71]. It is the undirected graph with vertex set $V = \{0,1\}^d$ and edge set $E = \{\{\mathbf{a},\mathbf{a}[1]\} \mid \mathbf{a} \in \{0,1\}^d\} \cup \{\{\mathbf{a},\mathrm{cyc}(\mathbf{a})\} \mid \mathbf{a} \in \{0,1\}^d, \mathrm{cyc}(a_d,\ldots,a_2,a_1) = (a_1,a_d,\ldots,a_2)\}$. The edges of the first subset are called *exchange* edges, the edges of the second subset are called *shuffle* edges. Here, we also consider *multiple* shuffle edges such that $\mathrm{SE}(d)$ is also 3-regular.

$\mathrm{CCC}(3)$ and $\mathrm{SE}(3)$ are shown in Fig. 1 and 2, resp. Note the self-loops at vertices 000 and 111 of $\mathrm{SE}(3)$ that ensure $\mathrm{SE}(d)$ being 3-regular.

Note that for d being even, $\mathrm{CCC}(d)$ is bipartite [LPS+98]. The cycles of $\mathrm{CCC}(d)$ are characterized directly by the corresponding sequence $\mathbf{a}$.

Cycles in $\mathrm{SE}(d)$ are more complex to describe. Let S be a set of integers. Let $\mathbf{a} =$

$(a_k, \ldots, a_1) \in S^k$. **a** is an aperiodic S-sequence if there is no $t > 1$ and **b** with $\mathbf{a} = \mathbf{b}^t$. **a** is a Lyndon S-sequence [CFL58] if it is an aperiodic S-sequence and the lexicographically smallest under all sequences obtained by cyclically shifting **a**.

The shuffle edges of $\mathrm{SE}(d)$ form disjoint *shuffle cycles*. Every cycle is uniquely characterized by a Lyndon $\{0,1\}$-sequence. In this paper, using the correspondence $0 \mapsto +1$ and $1 \mapsto -1$, we shall say that every shuffle cycle is characterized by a different Lyndon $\{-1,+1\}$-sequence and that every possible Lyndon $\{-1,+1\}$-sequence of length k with k being a divisor of d characterizes a different shuffle cycle.

2.2 Eigenvalues, Spectral Sets, and Computation Tools

Let A be the adjacency matrix of an undirected graph $G = (V, E)$ (with multiple edges allowed; the entry a_{ij} is the number of edges between nodes i and j). In the rest of this paper, we identify G and A. Let $n = |V|$ denote the number of vertices, and let I_n denote the $n \times n$ unit matrix. Then the polynomial $\chi(A; z) = \det(z \cdot I_n - A)$ is the *characteristic polynomial* of G, and the set $\mathrm{SpS}(G) = \{\lambda \mid \chi(A; \lambda) = 0\}$ is the *spectral set* of roots of $\chi(A; z)$. Such a root is called *eigenvalue*. In this paper, we do not consider the multiplicities of the eigenvalues.

Let A and B be two matrices. The *Kronecker* product $A \otimes B$ is the matrix one obtains from A by replacing entry a_{ij} by $a_{ij} \cdot B$.

A $(q \cdot p) \times (q \cdot p)$ matrix B is called (p, q)*-block circulant* iff there are $p \times p$ matrices $B_1, \ldots, B_q$ such that

$$B = \begin{pmatrix} B_1 & B_2 & \cdots & B_q \\ B_q & B_1 & \cdots & B_{q-1} \\ \vdots & \vdots & \ddots & \vdots \\ B_2 & B_3 & \cdots & B_1 \end{pmatrix} =: \langle B_1, B_2, \cdots, B_q \rangle \ .$$

If $p = 1$, B is called *circulant*. Block circulant matrices are well studied (e. g., see [Dav79]). In particular, there is a nice way to determine $\chi(B; z)$ and to compute $\mathrm{SpS}(B)$. Let $\omega_q = \mathrm{e}^{2\pi \mathrm{i}/q} = \cos(2\pi/q) + \mathrm{i} \cdot \sin(2\pi/q)$ be a primitive q-th root of unity. Let $B(x) = \sum_{k=1}^{q} x^{k-1} \cdot B_k$. The following proposition on the characteristic polynomial and the spectral set of block circulant matrices is very useful for the computation of the spectral sets of neatly constructed graphs.

Proposition 1 ([Dav79]) *Let $B = \langle B_1, \ldots, B_q \rangle$ be a (p, q)-block circulant matrix. Then*

$$\chi(B; z) = \prod_{j=0}^{q-1} \chi(B(\omega_q^j); z) = \prod_{j=0}^{q-1} \chi\Big(\sum_{k=1}^{q} \omega_q^{j \cdot (k-1)} \cdot B_k; z \Big) \ .$$

For the spectral set, this means

$$\mathrm{SpS}(B) = \bigcup_{j=0}^{q-1} \mathrm{SpS}\Big(\sum_{k=1}^{q} \omega_q^{j \cdot (k-1)} \cdot B_k \Big) \ .$$

E. g., as the n-cycle C_n is $(1,n)$-block circulant with $C_n = \langle 0,1,0\ldots,0,1\rangle$, $\mathrm{SpS}(C_n) = \{\omega_n^j + \omega_n^{(n-1)j} \mid 0 \le j \le n-1\} = \{2\cos(2\pi j/n) \mid 0 \le j \le n-1\}$. Similarly, the spectra of d-dimensional tori can be computed in this way resulting in the spectral set mentioned in Sec. 1.

Proposition 1 can be used directly to prove the following useful observation.

Proposition 2 *Let G and X be $p\times p$ square matrices. For the $(p,2)$-block circulant matrix $\langle G, X\rangle$, we have $\chi(\langle G,X\rangle;z) = \chi(G+X;z)\cdot\chi(G-X;z)$.*

Proposition 3 *Let $B = \langle B_1,\ldots,B_q\rangle$ be a real (p,q)-block circulant matrix with $B_{q-j+1} = B_j^{\mathrm{T}}$, $1 \le j \le q$. Then the following holds.*

(a) $B(\omega_q^{-j}) = B(\omega_q^j)^{\mathrm{T}}$, $B(\omega_q^j)$ is self-adjoint, $1 \le j \le q$.

(b) $\chi(B(\omega_q^{-j});z) = \chi(B(\omega_q^j)^{\mathrm{T}};z)$.

(c) If q is odd, then there is a polynomial $g(z)$ such that $\chi(B;z) = \chi(B(1);z)\cdot g(z)^2$

(d) If q is even, then there is a polynomial $g(z)$ such that

$$\chi(B;z) = \chi(B(1);z)\cdot\chi(B(-1);z)\cdot g(z)^2$$

(e) By (c) and (d), all eigenvalues of B that do not come from $B(1)$ and $B(-1)$ occur in pairs and belong to two-dimensional eigenspaces.

For the proof of Proposition 3, (a) and (b) can be shown directly, and for (c) and (d), use Proposition 1, (b) and that $\omega_q^{-j} = \omega_q^{q-j}$, for all j.

The following well known facts will be essential for the proof that the spectral sets of $\mathrm{CCC}(d)$ and $\mathrm{SE}(d)$ are different if d is even.

Proposition 4 *Let G be a connected graph with maximal degree Δ.*

(a) [Bol98, p. 263] G is regular iff $\Delta \in \mathrm{SpS}(G)$.

(b) [Bol98, p. 263] If $-\Delta \in \mathrm{SpS}(G)$, then G is regular and bipartite.

(c) [BK05, p. 379] G is bipartite iff for all $\lambda \in \mathrm{SpS}(G)$, also $-\lambda \in \mathrm{SpS}(G)$.

2.3 Further Notation

For a sequence $\mathbf{s} = (s_k,\ldots,s_1) \in \mathbb{Z}^k$, and $k \le n$, let $D_n[\mathbf{s}]$ be the $n\times n$-diagonal matrix with $s_1,\ldots,s_k,0,\ldots,0$ in the main diagonal. In particular, $I_n = D_n[1^n]$ is the identity matrix.

$C_n = \langle 0,1,0,\ldots,0,1\rangle$ denotes the circulant adjacency matrix of the cycle of length n. For a sequence $\mathbf{s} = (s_k,\ldots,s_1) \in \mathbb{Z}^k$, and $k \le n$, $C_n[\mathbf{s}] = C_n + D_n[\mathbf{s}]$. For reasons of

consistency, we need a special definition for the cases $n = 1$ and $n = 2$: $C_1[s] = (2 + s)$ and $C_2[s_2, s_1] = \begin{pmatrix} s_1 & 2 \\ 2 & s_2 \end{pmatrix}$.

$L_n = (l_{ij})$ denotes the $n \times n$-adjacency matrix of the linear array of length n. It is identical to C_n except for the entries $l_{1n} = l_{n1} = 0$ (instead of being 1). $L_n[\mathbf{s}]$ is defined analogously to $C_n[\mathbf{s}]$.

3 The Spectral Set of CCC(*d*)

In order to compute the spectral set of $\mathrm{CCC}(d)$, we generalize the notion of *cube-connect-ed*ness.

Let d be a non-negative integer, and let G be a graph with n, $n \geq d$, nodes, numbered from 1 through n. The d-dimensional Cube-Connected G-network is the graph $\mathrm{CC}(G, d)$ with vertex set $\{1, \ldots, n\} \times \{0, 1\}^d$. Two nodes $(i, \mathbf{a})$ and $(j, \mathbf{a})$ are adjacent iff i and j are adjacent in G. Furthermore, two nodes $(i, \mathbf{a})$ and $(i, \mathbf{b})$ are adjacent iff $\mathbf{a}$ and $\mathbf{b}$ differ exactly at the ith bit. So, $\mathrm{CC}(G, d)$ consists of 2^d copies of G that are interconnected in a hypercubic way. Using the length-d cycle C_d as G, we have with $\mathrm{CC}(C_d, d)$ the famous Cube-Connected Cycles network.

Let $\mathbf{s} = (s_d, \ldots, s_1) \in \{-1, +1\}^d$. The graph $G[\mathbf{s}]$ is obtained from G by adding a self-loop with weight s_i to node i, for all $i \in \{1, \ldots, d\}$.

Theorem 1 *Let G be a graph with n, $n \geq d$, nodes. Then*

$$\chi(\mathrm{CC}(G, d); z) = \prod_{\boldsymbol{s} \in \{-1,+1\}^d} \chi(G[\boldsymbol{s}]; z) \ .$$

Proof. Let $R_{n,d}$ be the $n \times n$ matrix with all entries being 0 except for r_{dd} which is 1, and let $X_{d-1} = I_{2^{d-1}} \otimes R_{n,d}$. Then the adjacency matrix of $\mathrm{CC}(G, d)$ can be expressed as follows:

$$\mathrm{CC}(G, d) = \begin{pmatrix} \mathrm{CC}(G, d-1) & X_{d-1} \\ X_{d-1} & \mathrm{CC}(G, d-1) \end{pmatrix}$$

By Proposition 2, this means that the characteristic polynomial of the whole graph can be expressed as follows:

$$\begin{aligned} \chi(\mathrm{CC}(G, d); z) &= \chi(\mathrm{CC}(G, d-1) + X_{d-1}; z) \cdot \chi(\mathrm{CC}(G, d-1) - X_{d-1}; z) \\ &= \chi(\mathrm{CC}(G[0^{d-1}, 1], d-1); z) \cdot \chi(\mathrm{CC}(G[0^{d-1}, -1], d-1); z) \quad (1) \\ &= \prod_{\mathbf{s} \in \{-1,+1\}^{d-1}} \chi(G[\mathbf{s}, 1]; z) \cdot \prod_{\mathbf{s} \in \{-1,+1\}^{d-1}} \mathrm{SpS}(G[\mathbf{s}, -1]; z) \quad (2) \end{aligned}$$

For (1), note that $\mathrm{CC}(G, d-1) \pm X_{d-1}$ is a copy of $\mathrm{CC}(G, d-1)$ where all nodes $(d, \mathbf{a})$ get a self-loop added with weight $-1, +1$. (2) follows by induction. □ (Theorem 1)

By choosing G being the d-cycle C_d, we obtain:

Corollary 1 *For* $\mathrm{CCC}(d)$*,*

$$\begin{aligned}\chi(\mathrm{CCC}(d);z) &= \prod_{\boldsymbol{s}\in\{-1,+1\}^d} \chi(C_d[\boldsymbol{s}];z)\\ \mathrm{SpS}(\mathrm{CCC}(d)) &= \bigcup_{\boldsymbol{s}\in\{-1,+1\}^d} \mathrm{SpS}(C_d[\boldsymbol{s}])\ .\end{aligned}$$

Hence, the spectral set of $\mathrm{CCC}(d)$ is exactly the union of the spectral sets of all d-cycles where the nodes of the cycles are weighted with all possible $\{-1,+1\}$-sequences.

The application of Eq. (1) from the proof of Theorem 1 can be interpreted as editing the original graph. The resulting graph has exactly the same spectrum as the original graph. Fig.1 shows the corresponding graphs for $d = 3, 2, 1$, when $\mathrm{CCC}(3)$ is edited. In the end, the are the 3-cycles with weighted self-loops. In the light of the editing, we call them residual cycles.

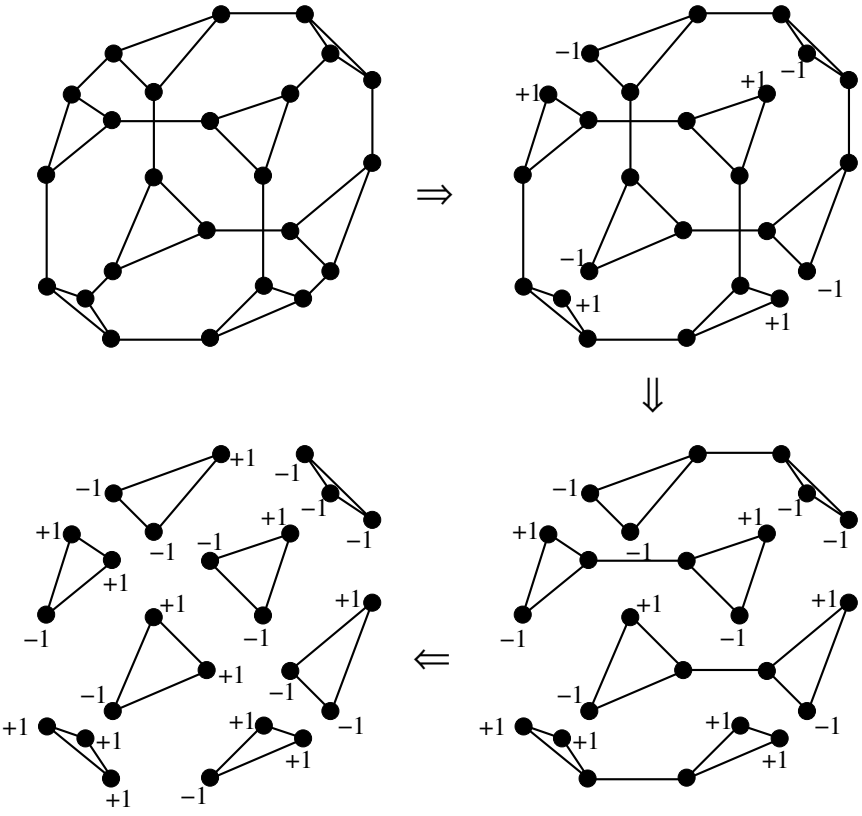


Figure 1: Editing the Cube-Connected Cycles network $\mathrm{CCC}(3)$.

Similarly, the spectrum of the Cube-Connected Lines network [Par86] can be characterized

in terms of linear arrays L_d.

4 The Spectral Set of SE(*d*)

In order to obtain the adjacency matrix of $\mathrm{SE}(d)$, we describe the shuffle edges and the exchange edges separately, i. e., $\mathrm{SE}(d) = \mathrm{Sh}(d) + \mathrm{Ex}(d)$.

Lemma 1 *(a) Let* $U(d) = \begin{pmatrix}1\\0\end{pmatrix} \otimes I_{2^{d-1}} \otimes \begin{pmatrix}1 & 0\end{pmatrix} + \begin{pmatrix}0\\1\end{pmatrix} \otimes I_{2^{d-1}} \otimes \begin{pmatrix}0 & 1\end{pmatrix}$.

Then, $\mathrm{Sh}(d) = U(d) + U(d)^{\mathrm{T}}$.

(b) $\mathrm{Ex}(d) = I_{2^{d-1}} \otimes \begin{pmatrix}0 & 1\\1 & 0\end{pmatrix}$

In order to prove Lemma 1, it suffices to identify the binary address **a** of a node with the number $(\mathbf{a})_2 + 1$.

Let $H_d = \frac{1}{2^{d/2}} \cdot \begin{pmatrix}1 & 1\\1 & -1\end{pmatrix}^{\otimes d}$, where $A^{\otimes d}$ denotes $\overbrace{A \otimes \cdots \otimes A}^{d}$. H_d is the well-known *Hadamard* matrix. Note that $H_d^{-1} = H_d$.

Lemma 2 *(a)* $H_d^{-1} \cdot \mathrm{Sh}(d) \cdot H_d = \mathrm{Sh}(d)$

(b) $H_d^{-1} \cdot \mathrm{Ex}(d) \cdot H_d = I_{2^{d-1}} \otimes \begin{pmatrix}1 & 0\\0 & -1\end{pmatrix}$.

For the proof, (b) can be shown easily by induction on d.

In order to show (a), a simple, but tedious computation shows that H_d commutes with both $U(d)$ and $U(d)^{\mathrm{T}}$, hence with $\mathrm{Sh}(d)$.

Theorem 2 *For the Shuffle-Exchange network* $\mathrm{SE}(d)$*,*

$$\chi(\mathrm{SE}(d); z) = \prod_{\substack{p,\\ p \text{ divisor of } d}} \prod_{\substack{\boldsymbol{a} \in \{-1,+1\}^p,\\ \boldsymbol{a} \text{ Lyndon } \{-1,+1\}\text{-sequence}}} \chi(C_p[\boldsymbol{a}]; z)$$

Proof. We have $\chi(\mathrm{SE}(d); z) = \chi(H_d^{-1} \cdot \mathrm{SE}(d) \cdot H_d; z)$. By Lemma 2,

$$\chi(\mathrm{SE}(d); z) = \chi(\mathrm{Sh}(d) + I_{2^{d-1}} \otimes \begin{pmatrix}1 & 0\\0 & -1\end{pmatrix}; z)$$

So the shuffle cycles are left unchanged, the exchange edges disappear, and all nodes get an additional $-1, +1$-self-loop, according to their binary addresses. Every shuffle cycle is characterized by Lyndon $\{0, 1\}$-sequences. Now the weights of the vertices of the

shuffle cycles are characterized by the respective Lyndon $\{-1,+1\}$-sequence, where 0s are replaced with $+1$ and 1s with -1.

Recall the special definition of $C_1[a]$ and $C_2[a,b]$. □ (Theorem 2)

As the computation of $H_d^{-1}\cdot \mathrm{SE}(d)\cdot H_d$ does not change the characteristic polynomial, this computation can again be regarded as editing the original graph. This time, the exchange edges are removed and their nodes receive an additional self-loop with weights $-1,+1$. The result of editing $\mathrm{SE}(3)$ is shown in Fig. 2 Again, we call the obtained cycles residual cycles.

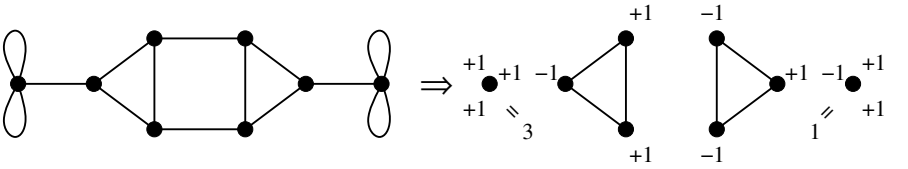


Figure 2: Result of editing the Shuffle Exchange network $\mathrm{SE}(3)$.

5 Spectral Relation between CCC and SE

The following Theorem shows the (surprisingly close) relation between the spectral set of $\mathrm{CCC}(d)$ and $\mathrm{SE}(d)$:

Theorem 3 *The spectral sets of the d-dimensional Cube-Connected Cycles network and the Shuffle-Exchange network are equal if and only if d is odd. Otherwise, the spectral set of the Shuffle-Exchange network is a proper subset of the spectral set of the Cube-Connected Cycles network.*

More formally, the following properties hold:

1. $\mathrm{SpS}(\mathrm{SE}(d)) \subseteq \mathrm{SpS}(\mathrm{CCC}(d))$
2. *d odd:* $\mathrm{SpS}(\mathrm{SE}(d)) = \mathrm{SpS}(\mathrm{CCC}(d))$
3. *d even:* $\mathrm{SpS}(\mathrm{SE}(d)) \subsetneq \mathrm{SpS}(\mathrm{CCC}(d))$, *since* $-3 \in \mathrm{SpS}(\mathrm{CCC}(d)) \setminus \mathrm{SpS}(\mathrm{SE}(d))$

The rest of this section is devoted to the proof of Theorem 3. We start with the first case:

Proof of Theorem 3, part 1:

Take any factor $\chi(C_p[\mathbf{a}];z)$ from $\chi(\mathrm{SE}(d);z)$, according to Theorem 2. There are two cases to be distinguished: If $p=d$, then $\chi(C_p[\mathbf{a}];z) = \chi(C_d[\mathbf{a}];z)$. Thus, by Corollary 1, $\chi(C_p[\mathbf{a}];z)$ is a factor of $\chi(\mathrm{CCC}(d);z)$. Consider now the case, that $p<d$. Let $d = p\cdot q$ and let δ_{ij} be the Kronecker delta, i. e., $\delta_{ij}=1$ if $i=j$, and $\delta_{ij}=0$ if $i\neq j$. Using the block circulant structure of $C_d[\mathbf{a}^q]$, it can be seen that $\chi(C_p[\mathbf{a}];z)$ is a factor of $\chi(C_d[\mathbf{a}^q];z)$:

$C_d[\mathbf{a}^q] = \langle A_0, A_1, ..., A_{q-1}\rangle$ is a block circulant matrix with $A_0 = L_p[\mathbf{a}]$, $A_1 = [\delta_{x,p}\delta_{y,1}]_{1\le x,y\le p}$, $A_{q-1} = A_1^{\mathrm{T}}$ and $A_2 = \cdots = A_{q-2} = (0)$, such that Propositions 2 and 3 are satisfied. One can see that $\chi(C_p[\mathbf{a}; z])$ is a factor of $\chi(C_d[\mathbf{a}^q]; z)$ by noting from Proposition 1 that

$$\chi(C_d[\mathbf{a}^q]; z) = \prod_{j=0}^{q-1} \chi(A_0 + \omega_q^j A_1 + \omega_q^{-j} A_{q-1}; z) \ ,$$

hence $\chi(C_p[\mathbf{a}; z]) = \chi(A_0 + \omega_q^0 \cdot A_1 + \omega_q^0 A_{q-1}; z)$ is also a factor of $\chi(\mathrm{CCC}(d); z)$. □

Proof of Theorem 3, part 2:

We already know that for all d, $\mathrm{SpS}(\mathrm{SE}(d)) \subseteq \mathrm{SpS}(\mathrm{CCC}(d))$. Thus, for odd d, it remains to be shown that $\mathrm{SpS}(\mathrm{SE}(d)) \supseteq \mathrm{SpS}(\mathrm{CCC}(d))$. Take any factor $\chi(C_d[\mathbf{b}]; z)$ from $\chi(\mathrm{CCC}(d); z)$, there are again two cases:

If $\mathbf{b} \in \{-1, +1\}^d$ is aperiodic, then $\chi(C_d[\mathbf{b}]; z)$ is a factor of $\chi(\mathrm{SE}(d); z)$, since $\mathbf{b}$ is already minimal with respect to periodicity, as presented in Theorem 2. If $\mathbf{b} \in \{-1, +1\}^d$ is periodic, then $\mathbf{b} = \mathbf{a}^q$ for some aperiodic $\mathbf{a} \in \{-1, +1\}^p$ with $d = p \cdot q$. Using Proposition 3, we know that $\chi(C_d[\mathbf{b}]; z) = \chi(C_d[\mathbf{a}]; z) \cdot g(z)^2$. Note that the corresponding eigenspace of the double eigenvalues that result from $g(z)^2$ is two-dimensional.

A sequence $\mathbf{b}' \in \{-1, +1\}^d$ that differs from $\mathbf{b}$ in one single (arbitrary) position – w.l.o.g. in the first position – is always aperiodic, thus $\chi(C_d(\mathbf{b}); z)$ is a factor of $\chi(\mathrm{SE}(d); z)$. We show that the characteristic polynomial factors as

$$\chi(C_d[\mathbf{b}']; z) = g(z) \cdot h(z) \ ,$$

which proves this part of the Theorem:

Take any of the two-dimensional eigenspaces of $C_d[\mathbf{b}]$ belonging to some root λ of $g(z)$. The vectors which have 0 in their first component form a one-dimensional (at least) subspace of eigenvectors of $C_d[\mathbf{b}']$ for the same eigenvalue λ, thus all these eigenvalues λ are preserved in $\chi(C_d[\mathbf{b}']; z)$ with multiplicity of at least 1. □

Note that because of Proposition 3(b) and (c) the above eigenspace argument only holds if d is odd. A counterexample for $d = 6$ is presented in Sec. 6.

Proof of Theorem 3, part 3:

Let d be even. $\mathrm{CCC}(d)$ is 3-regular and bipartite [LPS$^+$98]. So by Proposition 4, $-3 \in \mathrm{SpS}(\mathrm{CCC}(d))$.

If d is not a power of 2, then $\mathrm{SE}(d)$ contains cycles of odd length. Hence, $\mathrm{SE}(d)$ is not bipartite, so by Proposition 4, $-3 \notin \mathrm{SpS}(\mathrm{SE}(d))$. Even if d is a power of 2, then $\mathrm{SE}(d)$ at least contains self-loops at the nodes 0^d and 1^d, so here $\mathrm{SE}(d)$ is not bipartite, which means that $-3 \notin \mathrm{SpS}(\mathrm{SE}(d))$. □

Recall that -3 is not the only eigenvalue of $\mathrm{CCC}(d)$, d even and $d \in \{6, 10, 12, 14 \ldots\}$, not occurring in the spectral set of $\mathrm{SE}(d)$ (see the remark on the number of different eigenvalues in Sec. 1 and specifically the example in the next section).

6 An Instructive Example on the Eigenvalues of CCC(6)

Here we demonstrate by an example that there might be no simple correspondence between the eigenvalues of $\mathrm{CCC}(d)$ and $\mathrm{SE}(d)$, and that the eigenspace argument used in the proof of Theorem 3 might be necessary.

Corollary 1 and Theorem 2 state that the spectra of $\mathrm{CCC}(d)$ and $\mathrm{SE}(d)$ consist of the spectra of cycles where the vertices have self-loops with weights from $\{-1, +1\}$. As $\mathrm{SE}(d)$ consists of cycles of different sizes, there is no direct correspondence between the CCC-eigenvalues and the SE-eigenvalues.

For example, consider the case $d = 6$, and edit $\mathrm{CCC}(6)$ and $\mathrm{SE}(6)$ in order to get the residual cycles with self-loops from $\{-1, +1\}$.

For $\mathrm{CCC}(6)$, the residual cycle $C_6[-1, +1, -1, +1, -1, +1]$ which corresponds to the periodic binary sequence $010101 = (01)^3$ has the characteristic polynomial $(z^2-2)^2(z^2-5)$.

For $\mathrm{SE}(6)$, the residual cycle $C_2[-1, +1]$ which corresponds to the non-periodic binary sequence 01 has the characteristic polynomial $z^2 - 5$. So it is at this moment not yet clear whether the roots $\pm\sqrt{2}$ of $z^2 - 2$ originating, among others, from the CCC-cycle 010101 are eigenvalues of $\mathrm{SE}(6)$. In this case, they are because the characteristic polynomial of $C_6[-1, +1, -1, +1, +1, +1, +1,]$ which corresponds to the non-periodic binary sequence 000101 is $(z^2 - 2)(z^4 - 2z^3 - 5z^2 + 8z + 2)$.

On the other hand, for $\mathrm{CCC}(6)$, the residual cycle $C_6[-1, -1, +1, -1, -1, +1]$ which corresponds to the binary sequence 011011 has the characteristic polynomial $z(z-2)(z+1)(z+2)(z^2+z-4)$. The factor z^2+z-4 does not occur in any characteristic polynomial of the residual cycles of $\mathrm{SE}(6)$, so its roots $-\frac{1}{2} \pm \frac{1}{2}\sqrt{17}$ are not eigenvalues of $\mathrm{SE}(6)$, but only of $\mathrm{CCC}(6)$.

7 Conclusion

In this paper, we completely characterized the spectral sets of $\mathrm{CCC}(d)$ and $\mathrm{SE}(d)$ (Theorems 1 and 2, resp.). In order to compute the eigenvalues, we used a "graph editing" technique that illustrates the computation process.

Curiously, it turns out (Theorem 3) that the eigenvalue sets are identical if d is odd. If d is even, the set of eigenvalues of $\mathrm{SE}(d)$ is a proper subset of the set of eigenvalues of $\mathrm{CCC}(d)$. In order to show this result, we had to use the corresponding eigenspaces because there is no simple correspondence between the cycles of the residual graphs.

References

[ACG94] Noga Alon, Fan R. K. Chung, and Ronald L. Graham. Routing permutations on graphs via matchings. *SIAM Journal on Discrete Mathematics*, 7:513–530, 1994.

[BK05] Andreas Baltz and Lasse Kliemann. Spectral Analysis. In Ulrik Brandes and Thomas Erlebach, editors, *Network Analysis*, pages 373–417. Springer, 2005.

[Bol98] Béla Bollobás. *Modern Graph Theory*. Springer, New York, 1998.

[Bül97] J. Bültermann. A New Upper Bound for the Isoperimetric Number of deBruijn Networks. *Applied Mathematics Letters*, 10:97–100, 1997.

[CDS95] D. M. Cvetković, M. Doob, and H. Sachs. *Spectra of Graphs*. J. A. Barth Verlag, Heidelberg, 3rd edition, 1995.

[CFGM03] Francesc Comellas, Miguel Angel Fiol, Joan Gimbert, and Margarida Mitjana. The spectra of wrapped butterfly digraphs. *Networks*, 42:15–19, 2003.

[CFL58] K. T. Chen, R. H. Fox, and R. C. Lyndon. Free Differential Calculus, IV. The Quotient Groups of the Lower Central Series. *The Annals of Mathematics, Second Series*, 68(1):81–95, 1958.

[Chu97] Fan R. K. Chung. *Spectral Graph Theory*. American Mathematical Society, 1997.

[Dav79] Philip J. Davis. *Circulant Matrices*. Wiley, New York, 1979.

[DT98] Charles Delorme and Jean-Pierre Tillich. The spectrum of de Bruijn and Kautz graphs. *European Journal of Combinatorics*, 19(3):307–319, 1998.

[EKM03] Robert Elsässer, Rastislav Královič, and Burkhard Monien. Sparse topologies with small spectrum size. *Theoretical Computer Science*, 307:549–565, 2003.

[Lei92] F. Thomson Leighton. *Introduction to Parallel Algorithms and Architectures: Arrays, Trees, Hypercubes*. Morgan Kaufmann Publishers, San Mateo, CA, 1992.

[LPS+98] K. Li, Y. Pan, H. Shen, G. H. Young, and S. Q. Zheng. Lower Bounds for Dynamic Tree Embedding in Bipartite Networks. *Journal of Parallel and Distributed Computing*, 53:119–143, 1998.

[Par86] Ian Parberry. On recurrent and recursive interconnection patterns. *Information Processing Letters*, 22:285–289, 1986.

[PV81] Franco Preparata and Jean Vuillemin. The Cube-Connected Cycles: a versatile network for parallel computation. *Communications of the ACM*, 24:300–309, 1981.

[RSW98] Yuval Rabani, Alistair Sinclair, and Rolf Wanka. Local Divergence of Markov chains and the analysis of iterative load-balancing schemes. In *Proc. 39th IEEE Foundations of Computer Science (FOCS)*, pages 694–703, 1998.

[Sch01] Gunnar Schmidt. Über die Spektren wichtiger Graphklassen. Studienarbeit, Universität Paderborn, 2001.

[Sto71] H. S. Stone. Parallel Processing with the Perfect Shuffle. *IEEE Transactions on Computers*, C-20:153–161, 1971.

Flexible Scheduling and Thread Allocation for Synchronous Parallel Tasks

Christoph W. Kessler and Erik Hansson
IDA, Linköping University, 58183 Linköping, Sweden
{chrke,eriha}@ida.liu.se

Abstract: We describe a task model and dynamic scheduling and resource allocation mechanism for synchronous parallel tasks to be executed on SPMD-programmed synchronous shared-memory MIMD parallel architectures with uniform, unit-time memory access and strict memory consistency, also known in the literature as PRAMs (Parallel Random Access Machines).

Our task model provides a two-tier programming model for PRAMs that flexibly combines SPMD and fork-join parallelism within the same application. It offers flexibility by dynamic scheduling and late resource binding while preserving the PRAM execution properties within each task, the only limitation being that the maximum number of threads that can be assigned to a task is limited to what the underlying architecture provides. In particular, our approach opens for automatic performance tuning at run-time by controlling the thread allocation for tasks based on run-time predictions.

By a prototype implementation of a synchronous parallel task API in the SPMD-based PRAM language Fork and experimental evaluation with example programs on the SBPRAM simulator, we show that a realization of the task model on a SPMD-programmable PRAM machine is feasible with moderate runtime overhead per task.

1 Introduction

During the recent years, computer architectures available on the consumer market have switched from single-core architectures to multi-cores, and it is reasonable to assume that we enter the many-core era in the near future. The reason for this change is that hardware manufacturers try to keep up with the demand of more computation power and at the same time consume less energy. As a consequence, speed-up of legacy, single-threaded computer programs does not come for free any more but requires rewriting to leverage many cores. Even worse is that, even where providing a shared memory abstraction, these new architectures mainly follow NUMA and SMP designs that lack features that could ease parallel programming, such as strong memory consistency or deterministic execution.

To ease the burden for both application programmers and compiler engineers, some architecture projects [PBB+02, For10, WV08] are working towards supporting more powerful, deterministic parallel programming models such as the PRAM model [FW78, KKT01]. The PRAM model is often considered as only a theoretical programming model, but already in the 1990s it has been realized in hardware, albeit not on a single chip, e.g. the SB-PRAM [PBB+02, KKT01]. In a current project by VTT Oulu (Finland) a new architecture

called *Replica* is being developed. It supports both PRAM and NUMA mode, and features massively hardware-multithreaded configurable very long instruction word (VLIW) processor cores with chained functional units and a powerful 2D mesh on-chip combining network providing uniform access to on-chip distributed shared memory. Replica will be realized in hardware and is the successor of the *Total Eclipse* architecture [For10].

PRAMs are instruction-level synchronous MIMD parallel architectures with shared memory and are traditionally programmed in the SPMD execution style using PRAM languages such as Fork [KKT01, KS97a], e [For04] etc. that map the naturally available tight synchronization of the underlying hardware to the expression and statement level, allowing to reduce explicit synchronization in the code while maintaining deterministic parallel execution.[1] While following the SPMD style across the whole machine gives full control over the assignment of computation to execution resources, it becomes cumbersome for more irregular application scenarios that require adaptive resource allocation strategies.

In this work, we show how a flexible MIMD task model allowing multithreaded PRAM tasks, can be realized on top of a SPMD programmed PRAM platform. The data-driven, dynamic scheduling principle of our task model is inspired by current single-threaded task programming models such as StarPU and StarSs. Our work is part of a pre-study for some features of the Replica architecture's runtime system. As the Replica simulator and software toolchain is not completely finished yet, we use the similar SBPRAM simulator and Fork toolchain [KKT01] for the prototype implementation and evaluation.

2 Principle

We are given a PRAM with p hardware threads and with a low-level programming model based on SPMD execution style, i.e., all p threads execute `main` from the beginning and the hardware itself does not provide for dynamic creation and deletion of additional threads[2]. Hence, a software layer on top of a low-level programming environment will be responsible for providing a task-based programming model. In the following, we propose a task-based programming model with non-preemptive dynamic scheduling, where the tasks can be serial or PRAM-style synchronous parallel computations and thus might require one or several threads of the underlying PRAM machine for execution.

Thread pool A program that uses synchronous parallel tasks or asynchronous sequential tasks (or both) should in the beginning have a single thread initialize the task system by calling `init_tasksystem();` and then send a subset of the available hardware threads as *worker threads* into a central shared *thread pool* TP, where they wait for work. A thread joins the thread pool by calling `join_threadpool()`. If no tasks have been created yet at this time, at least one thread should continue execution to create work for the others, and may still join the thread pool later, thereby becoming an additional worker thread.

[1] The strict memory consistency model of PRAMs is the strongest possible shared memory consistency model, it is even stronger than sequential consistency.

[2] In the following, thread means hardware thread (also known as *virtual processor*) unless otherwise stated.

```
typedef struct vector {
 int vid;      // unique ID for debugging purposes
 int state;    // 0 = data not ready, 1 = data valid
 void *pdata; // address of the wrapped payload data array
 int n_elems; // number of elements
 int type;     // element type field, refers to type table
 struct sptaskdescriptor *consumers[MAXCONSUMERSPERVECTOR];
 int n_consumers; // number of registered consumer tasks
} Vector;
```

Figure 1: Implementation of the `Vector` container in the C-based PRAM language Fork.

The program terminates successfully (by each thread calling `exit(0)`) if all p worker threads are waiting idle in the thread pool and there is no task left in the task queue (see below). A global counter holding the current number of threads waiting idle in TP can easily be maintained using atomic prefix-add operations.

Containers A container is a wrapper data structure that encapsulates aggregate user data such as an array together with metadata such as information about its size, type and state (invalid/ready), and manages memory access information such as where its most recent contents is currently located if there are multiple kinds of memory in the system. In particular, containers can, on such systems, provide consistent access to data on request (i.e., a call to the container's flush operation) by enforcing a write-back to the default memory location. On a PRAM this latter feature is actually not required, while it can be useful on a NUMA system as it provides an object-based distributed shared memory.

The most common container, and the only one that we support by now, is `Vector`, inspired by the corresponding container type in the C++ STL. For the C-based PRAM language Fork, our `Vector` is internally defined as shown in Figure 1. However, following a modular design style, the application programmer (API user) should not access these fields directly but use predefined access functions and macros instead, some of which will be described in the following.

In C++, `Vector` is generic in the element data type. In the C-based Fork language, we have to represent the element type explicitly using a `type` field. The `consumers` are those task instances (see later) that take this operand container with access mode "in".

The function `Vector *new_Vector(void *array, int length, int type );` allocates a new Vector container for `array` of `length` elements. The payload data `array` is not copied, only a pointer to it is stored in the container. Hence, it is possible that multiple containers point to the same payload data. This is a way to avoid unnecessary copying; it is the programmer's responsibility that this sharing of payload data does not lead to data races. An example will be given at the end of this section.

The state of a vector `v` can be set as follows. A task is blocked until all its argument containers are in ready state. Tasks waiting for an argument container to become valid are registered in the container so they can be notified. `setREADY(v)` sets `v` to state valid; this operation is used for containers holding input data to a task-based computation.

```
typedef struct sptaskdescriptor {
 int tid;  // Rank in Frozen Queue FQ
 void (* func)( int argc, Vector **argv, Vector *ret );
 sync void (* sfunc)( sh int argc, sh Vector **argv, sh Vector *ret );
 int argc;  // number of arguments
 Vector **args;  // dynam. allocated shared array of argument containers
 Vector *retvalue;  // container that holds the return value
 int minnthreads, maxnthreads;  // lower and upper bound for #threads
 int *shmem;  // pointer to shmem, initially 0
 int shmemsize;
 int nthreads; // actual number of threads running this task as a group
} sptask;
```

Figure 2: Data structure for a SP-task descriptor in C/Fork. One such entry exists for each task in the global shared heap memory.

`setREADYandpromote(v)` additionally notifies the consumer tasks that depend on `v`, and promotes these to READY state where `v` was the last awaited argument.

Tasks can also return data in a container, which usually is then used as input to subsequent, data-dependent tasks. Also in this case, the consumers will be notified and promoted to READY state as applicable. Depending on the type of return data, one of the following three versions of YIELD should be used:

- `YIELD_VALUE(ret,type,value)` copies scalar base-`type` data to (the first element of) a (pre-allocated) payload array in a (pre-allocated) `Vector ret`.
- `YIELD_PTR(ret,ptr)` replaces the payload array field in the `Vector ret` by the new array pointed to by `ptr`.
- `YIELD_VOID(ret)` is a variant of YIELD with no assigned return value. This is useful for in-place updates of the payload array; we will later see an example of this.

SP-functions and SP-tasks Synchronous parallel functions (SP-functions) are executed by the calling group of hardware threads in lock-step mode, hence the execution will be deterministic (assuming that the resolution of possible concurrent write access conflicts is deterministic, too).

We define *synchronous parallel tasks* (*SP-tasks*) as instantiations (invocations) of such synchronous parallel functions by a group of threads. The special case of invocations of SP-tasks by a single-thread group is the ordinary sequential task model known from classical scheduling theory. SP-tasks are a special case of *malleable tasks*, which can be executed by an arbitrary number of threads but are internally not necessarily synchronous.

Tasks are, at runtime, represented by a *task descriptor*, a data structure defined in Figure 2, which contains the key parameters of a task, such as the SP-function to be called, the argument vector and return value, minimum and maximum specified thread allocation (or default values if unspecified), and also some non-public administrative entries such as the task state. The `shmemsize` field holds the size of the shared memory block `shmem` to be

allocated to the task before execution; it must be 0 for an asynchronous task and > 0 for a synchronous task to accommodate its group stack and heap.

These task properties are set upon creation (see below) or derived automatically; it is not intended to change them during execution (e.g., no reallocation of its shared memory segment while the task is running). In future work we may add some get functions or macros to allow for querying of certain task properties.

Creating new tasks and SP-tasks At the run-time system programming level, task descriptors for asynchronous and synchronous tasks are created explicitly by the constructors

```
sptask *new_task ( void (*foo)(int, Vector **, Vector *),
                   int argc, Vector **args, Vector *ret );

sptask *new_stask ( sync void (*foo)(sh int, sh Vector **, sh Vector *),
                    int argc, Vector **args, Vector *ret,
                    int minp, int maxp, int shmemsize );
```

which take a function name and its arguments. The static type checking of synchronicity and sharity in Fork requires different constructors for synchronous and asynchronous tasks. `minp` and `maxp` specify the minimum and maximum number of threads to be used for this task. The implementation enforces at runtime that the value for `minp` is at least 1, and that of `maxp` is automatically truncated to the maximum available number of workers if it is too large. Hence, it is safe (but possibly not most efficient) to oversize `maxp`.

A task (synchronous or asynchronous) t can be spawned explicitly by a spawn operation: `spawn_task(`t`)` creates a task descriptor with the parameters given by t and enqueues it to a central scheduler for execution concurrently with the continuation of the spawning thread; control returns thus immediately to the spawning thread.

Lifecycle, scheduling and synchronization of tasks During its lifecycle, a task's state changes from new to ready to running to terminated. When spawned, created tasks receive a unique task ID (`tid` field) and are sent to a *frozen queue* FQ of tasks that are not yet data ready. Once all its input arguments (containers) are in ready state, a task is promoted to ready state and enqueued in a central shared task queue TQ, from where idle worker threads fetch new work for execution. All synchronization between SP-tasks is data driven.

From the task queue TQ, idle threads fetch their next task for execution. An asynchronous task will be assigned to exactly one thread. For synchronous tasks, at least `minnthreads` and at most `maxnthreads` idle threads will be collected, barrier-synchronized and assigned as a synchronous group to the execution of the task's SP-function. Once the task terminates, the task status will be changed to `TERMINATED`.

For now, we implemented for SP-tasks the thread assignment policy *FIFO-FLEX*, i.e., the oldest task waiting in TQ will be assigned threads first, and dispatched as soon as at least `minnthreads` have been assigned; as multiple threads can become idle (almost) simultaneously, it is possible that, implementation defined, more threads, up to `maxnthreads` in total, could be allocated when the task starts execution. Further available threads will be reassigned to the next task(s). The current implementation is blocking, i.e., only one

```
#include <fork.h>
#include "forktasks.h"

#define N_A 2048  // (max) array size
sh int A[N_A];
sh Vector *s, *r;
// ... some minor details omitted

void main( void )
{
 ... // read / initialize array A
 if ($==0)
    init_tasksystem();
 barrier;
 if ($==0) {
    sptask *t;
    s = new_Vector( A, N_A );
    r = new_Vector( A, N_A ); // in-place
    t = new_stask( msort, 1, &s, r, 1, 1, 1000 );
    setREADY( s );
    spawn_task( t );  // spawn the initial task (msort)
 }
 barrier;
 join_threadpool();
 // once all work has been done, the workers return here
}
```

Figure 3: Mergesort example, the main program. The hardware thread (PRAM processor) with rank 0 initializes the task system and creates two vector containers `s` and `r` that both share the same payload array `A` of size `N_A`, for in-place sorting by the SP-task `msort` that takes `s` as input operand and `r` as output operand. After this task has been spawned, all hardware threads join the thread pool where they are assigned work. The code for the SP-tasks `msort` and `merge` can be found in Figures 4 and 5, respectively.

task can be assigned and dispatched at a time. In future extensions of this work, additional thread assignment policies such as smallest-task first or best-fit could be tried. Adaptive thread allocation as in [EKC06, KL08] could be tried as well.

Example Figures 3, 4 and 5 show an implementation of recursive parallel mergesort with explicitly parallel tasks. For mergesort there are two types of tasks required: `msort`, recursive mergesort tasks that form the divide step in the recursion tree, which create new subtasks with their containers in each instance (see Figure 4), and `merge`, the tasks forming the combine step, merging two subsolutions into one (see Figure 5). The instances of these tasks are connected by data flow edges via container objects. See the figure captions for further explanation of the code. The values used in the `new_stask()` calls for `minp` (1) and `maxp` (1) are motivated by the fact that `msort` tasks themselves do not perform much work but unfold the tree of `merge` tasks where almost all of the computational work is done. A `merge` task of size n with a (not work-optimal) fully parallel implementation can use up to $M(n) = n$ threads. In fact, this value M is a performance tuning parameter.

```
sync void msort ( sh int argc, sh Vector **argp, sh Vector *ret )
  // invariant: the Vector's are allocated by caller
{
 seq {
  Vector src = (Vector *)argp[0];  // container passed in
  int *arr = ((int *)(src->pdata));  // payload array
  int n = src->n_elems;
  if (n<=1) {
     // may call qsort(arr,n) here if threshold > 1
     YIELD_VOID( ret );  // return data in place
  }
  else {
     sptask *t1, *t2, *t3;
     Vector **s = (Vector**)shmalloc(2*sizeof(Vector*));
     Vector **r = (Vector**)shmalloc(2*sizeof(Vector*));
     s[0] = new_Vector( arr, n/2 );
     s[1] = new_Vector( &(arr[n/2]), n-n/2 );
     r[0] = new_Vector( arr, n/2 );
     r[1] = new_Vector( &(arr[n/2]), n-n/2 );
     t1 = new_stask( msort, 1, s, r[0], 1, 1, 1000 );
     t2 = new_stask( msort, 1, s+1, r[1], 1, 1, 1000 );
     setREADY( s[0] );
     setREADY( s[1] );
     spawn_task( t1 );
     spawn_task( t2 );
     // synchronization on r1 and r2 is automatic by scheduler
     t3 = new_stask( merge, 2, r, ret, 1, M(n), 1000 );
     spawn_task( t3 );  // delegates the writing of ret
}}}
```

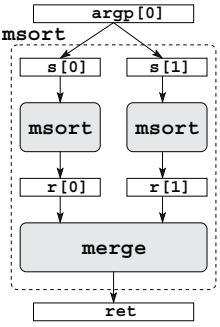


Figure 4: Mergesort example (cont.), code for the `msort` (mergesort) tasks. A `msort` task takes 1 argument, passed in `argp[0]`: the vector to be sorted. It returns the sorted vector in `ret`, with the same payload data for in-place sorting. In the `else` branch, a new level of the task graph is unfolded. Fresh vector container objects (`s[0]`, `s[1]`, `r[0]`, `r[1]`) are dynamically allocated for all intermediate operands of created subtasks, see the illustration, while the payload array space can be reused, thus avoiding copying and memory management. The data flow dependencies between the `msort` subtasks `t1`, `t2` and the `merge` subtask `t3` are given explicitly by the container references.

3 Implementation Details

A prototype of a runtime system and API has been implemented in Fork for the SBPRAM, for which we use the cycle-accurate instruction-level simulator `pramsim`.

Our implementation uses for the general case (*mixed-mode parallelism*, i.e. allowing both synchronous parallel and asynchronous sequential tasks to occur in the same application) two central, blocking, shared task queues as main data structure for the frozen queue and for the ready queue, respectively, which are implemented as bounded buffers of size $O(maxT)$ where $maxT$ is the maximum number of tasks that could be active simultaneously; this parameter can be adapted if necessary. The queue implementations make extensive use of the SBPRAM's nonblocking, constant-time multiprefix-add operations.

```
sync void merge ( sh int argc, sh Vector **argp, sh Vector *ret )
  // invariant: the Vector's are allocated by caller
{
 sh Vector *s1 = (Vector *)argp[0],  *s2 = (Vector *)argp[1];
 sh int *arr1 = (int *)(s1->pdata),  *arr2 = (int *)(s2->pdata);
 sh int n1 = s1->n_elems,            n2 = s2->n_elems;
 /* ... merge arr1 and arr2 in place, code omitted */
 seq
  YIELD_VOID( ret );  // as ret also points to arr1
}
```

Figure 5: Mergesort example (cont.), code for the `merge` tasks. Merge tasks take two input parameters, namely two vector containers passed in `argp`, pointing to two adjacent subarrays in an array where they are to be merged in-place. The result vector container points to the first subarray head (`arr1`). Once the subarrays have been merged, the `ret` vector container is advanced to ready state by `YIELD_VOID`.

Each new synchronous parallel task is, upon dispatch, allocated a new shared stack segment from global shared heap memory, which it keeps during its lifetime and releases upon termination. The code for startup and finalization of parallel tasks is (as already for ordinary Fork programs) written in SBPRAM assembler because some hardware thread (PRAM processor) registers for addressing the new shared stack segment must be saved and set up resp. restored properly.

For mixed-mode parallel applications, dispatch of data-ready tasks is, in the current prototype, serialized because the implementation needs to make sure that lower and upper bounds for allocating available threads to all ready tasks in the *FIFO-FLEX* scheduler are properly addressed as stated by each individual parallel task. Simpler (non-individual) task allocation policies or less fair dispatch schemes might allow for a more efficient, non-serializing implementation, which is a subject for future extension.

For programs that use asynchronous tasks only, we have an alternative implementation with lower overhead and a completely parallel (and non-blocking) task queue.

4 Experimental Evaluation

Sequential tasks only We use Fibonacci (the computation of the Nth Fibonacci number by the well-known recursive algorithm) as a very simple example that contains almost no computation, hence it reflects very well the overhead that is incurred by the task management system. Table 1 (left) shows runtime results taken on the SBPRAM simulator (given in thousand SBPRAM clock cycles) for $N = 17$ with different numbers of SBPRAM processors and for the two implementations of the shared task queue data structures: (i) blocking get_task and nonblocking insert, and (ii) completely non-blocking. Fib(17) recursively unfolds 7751 tasks in total, and creates 10335 operand containers. The task queue buffers were dimensioned with 8K entries each. The average overhead per task is about 2000 clock cycles with the nonblocking task queue and only slightly higher with

Table 1: Test runs for Fibonacci number calculation (left) and Mergesort (right), all times are in thousand SBPRAM clock cycles.

Overall execution time for computing the 17th Fibonacci number, creating 7751 tasks.

Hardw. Thr.	Time w. Blocking TQ (i)	Time with Non-blocking TQ (ii)
1	16086	14148
2	8158	7090
4	4466	3579
8	2839	1842
16	2209	1068
32	2120	926
64	2099	897
128	2080	893
256	2056	892

Overall execution time for Mergesort of 2048 integers, creating 6143 tasks and 8191 vectors. M denotes the choice for the upper bound `maxp` for `merge` tasks of size N.

HW Thr.	Time (blocking TQ)				
	$M = N/2$	$= N/4$	$= N/8$	$M = 1$	seq.
1	25033	25037	25039	24935	*18912*
2	13615	13118	12855	12933	*9686*
4	7189	6764	6532	6992	*5016*
8	4158	3730	3796	4312	*2990*
16	2572	2320	2425	3265	*2244*
32	2116	2006	*2000*	3104	2166
64	1896	1858	*1829*	3077	2147
128	1799	*1777*	1793	3065	2135

the blocking one. Note that program execution (and timing) on SBPRAM is completely deterministic, therefore a single test run per scenario is sufficient for the measurements.

One fundamental problem that this example reveals is that Fibonacci creates the tasks in a LIFO way, i.e., the earliest-created task is executed last of all, hence the maximum number of tasks that (could be) simultaneously alive almost equals the overall number of tasks, requiring an equally large dimensioning of the task queue data structures to avoid overflow and possibly also (too) many containers that are alive simultaneously. Similar behavior will be encountered with many divide-and-conquer algorithms, too. Where space becomes a critical resource, recursive programs thus might need to be reformulated in order to limit the amount of simultaneously alive tasks and containers.

Mergesort Our second example is the parallel mergesort program as shown above. Mergesort (`msort`) tasks are set up to use exactly 1 worker thread, and `merge` tasks use at least one and up to N workers for merging of size-N vectors. Results are shown in Table 1 (right) for a Mergesort of 2048 integers and different PRAM sizes. As 6143 tasks are generated, the average granularity is approximately 3000 instructions per task with serial ("seq.") and 4000 with parallel merging, including the dispatch overhead of about 2000 instructions. This makes also clear that the granularity is too fine for most of the tasks, as the overhead dominates. Coarsening the task granularity, e.g. by replacing spawning of light-weight `msort` tasks with inlined computation, is a way of tuning performance; this can be an issue for future work on auto-tuning optimizations.

We experimented with different choices for the upper limit M of the number of threads for `merge` calls using fully parallel merging, which is not work-optimal. M is a tuning parameter; we found empirically that e.g. $M = N/4$ and $M = N/8$ work better than $M = N$, $M = N/2$, $M = \log_2 N$ or $M = 1$. As expected, these do basically not differ in the case of a single worker thread. Using a sequential merge routine (work-optimal) leads to lower cost for small machine sizes but does not scale beyond 16 threads.

5 Related work

The synchronous parallel task concept is inspired by the join statement of Fork [KS97b, KKT01]. The main difference is that join is intended to implement synchronous parallel critical sections, so there will, at any time, be at most one instantiation of any synchronous parallel function (join body) running, while here several instances of the same SP-function could run simultaneously on disjoint thread subsets. The concept of parallel critical sections is motivated by the need of protecting certain code sections against race conditions caused by unsynchronized concurrent updates. While strict sequentialization using mutex locks is an option, the deterministic synchronous execution of PRAM systems opens for another more scalable way of avoiding race conditions. The join construct was demonstrated in Fork for parallel heap memory allocation and accelerated I/O processing [KKT01], operations that otherwise require mutual exclusion of individual threads.

The optimization of thread allocation to synchronous tasks was solved by Eriksson et al. [EKC06] for the special case where subtasks generated by recursive calls in parallel divide-and-conquer computations were executed in-line, either in parallel on disjoint thread subgroups or in serial by the entire thread group. Execution time of tasks is predicted from closed formulas that depend on problem and group size, and that are calibrated from timing data on the target machine (here, SBPRAM). Here, we generalize over this work by decoupling the subtask execution from the caller task, adding more flexibility and possibly sacrificing predictability.

StarPU [ATN10] is a run-time system for single-threaded and multi-threaded (but non-PRAM) tasks that can execute on different kinds of execution units such as CPU cores, GPUs or other programmable accelerators. In contrast to our model, StarPU does not support nested or recursive tasks. StarPU tasks are serial[3] and run on a single CPU or single GPU; support for multi-CPU OpenMP tasks is an issue of ongoing work. StarPU keeps track of each task's recent execution time history depending on input sizes, such that future decisions can be based on preditions made from collected history data.

StarSs (Star-superscalar) [PBAL09] is a family of languages and runtime systems implemented for different kinds of parallel target platforms, such as CellSs, GPUSs, OMPSs, ClusterSs. Similarly as StarPU, the StarSs model extends sequential computing by discovering and scheduling data-ready sequential tasks, which are defined by invocations of specific user functions, at run-time to some available execution unit, such as an idle CPU core, a GPU or a Cell SPU. While StarPU uses a specific API, StarSs uses language extensions to mark up task functions with their input and output parameters.

Wimmer and Träff [WT11b, WT11a] have considered multiple-thread allocation in a work-stealing scenario on distributed task queues, in order to gather a set of several threads for executing a parallel (but non-PRAM) algorithm. They use the concept of *mixed-mode parallelism* to support both task-based algorithms, such as divide and conquer, and SPMD (single program multiple data) algorithms in the same application where one task can

[3] GPU tasks in StarPU are of course internally massively parallel as they are run on many or all cores of a GPU, but to the task scheduler they look like an ordinary serial task, and the entire GPU is treated as a single resource for scheduling.

spawn other tasks. Their approach is based on classical work-stealing with independently working processors with their own queues where communication is only done when they are out of work. Wimmer and Träff organize processor groups in a binary tree topology. At level 0, each processor is in its own group; on higher levels they work together in groups of groups, called teams. Creating a team is done by work stealing in a deterministic way by visiting so-called partners on each level until work is found. The teams are needed to execute parallel tasks that require more than one thread. A team can "live" longer than a task, e.g. be used to execute tasks that need at most the number of threads available in the team. The implementation uses standard lock free data structures. Apart from not being limited to group sizes that are powers of two, a main difference from their work is that we can afford the luxury of having a central shared work queue without time penalty since we have a Combining CRCW PRAM architecture.

6 Conclusion

We have introduced and evaluated a task model for flexible dynamic scheduling and resource allocation mechanism for synchronous parallel tasks executing on a PRAM architecture. Our proof-of-concept prototype implementation shows that we can realize it with a low runtime overhead per task. It provides the option of dynamic task scheduling and thread allocation on top of a SPMD-programmed PRAM machine that was mostly designed for single-task applications. It combines the flexibility of task-based runtime systems with the power of SPMD-controlled, naturally synchronized PRAM execution within the SP-tasks.

Future work will consider high-level programming support that avoids low-level coding of calls to the run-time system API. Possible approaches include (1) high-level language constructs such as `spawn`, (2) a library of skeleton functions for frequently occuring parallel algorithmic design patterns such as *parallel divide-and-conquer*, or (3) graphical programming languages for specifying task graphs with parallel tasks (similarly to the illustration in Fig. 4) from which Fork source code can be generated automatically [KSF10]. Experiments with further thread allocation strategies and, in particular, static and dynamic autotuning of thread allocation will be considered in future work. Finally, porting our Fork-based prototype implementation to the new VTT Replica architecture and system software can be done as soon as the complete toolchain is available.

Acknowledgments This research is funded by VTT, project REPLICA, and by SeRC. We thank the anonymous reviewers for their helpful comments.

References

[ATN10] Cedric Augonnet, Samuel Thibault, and Raymond Namyst. Automatic Calibration of Performance Models on Heterogeneous Multicore Architectures. In *Proc. HPPC-2009,*

in Euro-Par 2009 Workshops, volume 6043 of *Lecture Notes in Computer Science*, pages 56–65. Springer Berlin / Heidelberg, 2010.

[EKC06] Mattias Eriksson, Christoph Kessler, and Mikhail Chalabine. Load Balancing of Irregular Parallel Divide-and-Conquer Algorithms in Group-SPMD Programming Environments. In *Proc. 8th Workshop on Parallel Systems and Algorithms (PASA 2006), Frankfurt am Main, Germany, GI Lecture Notes in Informatics (LNI), vol. P-81*, pages 313–322, March 2006.

[For04] M. Forsell. Designing NOCs with a parallel extension of C. In *FDL'04*, pages 463–475, 2004.

[For10] M. Forsell. TOTAL ECLIPSE – An Efficient Architectural Realization of The Parallel Random Access Machine. *Parallel and Distributed Comput., Ed. A. Ros, IN-TECH, Wien*, pages 39–64, 2010.

[FW78] S. Fortune and J. Wyllie. Parallelism in random access machines. In *Proc. 10th Symposium on the Theory of Computation (STOC)*, pages 114–118, 1978.

[KKT01] Jörg Keller, Christoph Kessler, and Jesper Träff. *Practical PRAM Programming*. Wiley Interscience, 2001.

[KL08] Christoph Kessler and Welf Löwe. A Framework for Performance-Aware Composition of Explicitly Parallel Components. In *Proc. ParCo-2007 conference, Jülich/Aachen, Germany, Sep. 2007. In C. Bischof et al. (eds.): Parallel Computing: Architectures, Algorithms and Applications, Advances in Parallel Computing Series, Volume 15, IOS Press*, pages 227–234, February 2008.

[KS97a] Christoph W. Keßler and Helmut Seidl. The Fork95 Parallel Programming Language: Design, Implementation, Application. *Int. J. of Par. Programming*, 25(1):17–50, February 1997.

[KS97b] Christoph W. Keßler and Helmut Seidl. Language Support for Synchronous Parallel Critical Sections. In *Proc. APDC'97 Int. Conf. on Advances in Parallel and Distributed Computing, Shanghai, China*. IEEE CS press, March 1997.

[KSF10] Christoph W. Kessler, Wladimir Schamai, and Peter Fritzson. Platform-independent modeling of explicitly parallel programs. In *Proc. PARS'10: 23rd PARS-Workshop on parallel Systems and Algorithms, Hannover, Germany, Feb. 2010. In: M. Beigl and F. Cazorla-Almeida (Eds.): ARCS'10 Workshop Proceedings*, pages 83–93. VDE-Verlag Berlin/Offenbach, Germany, February 2010.

[PBAL09] Judit Planas, Rosa M. Badia, Eduard Ayguadé, and Jesús Labarta. Hierarchical Task-Based Programming With StarSs. *IJHPCA*, 23(3):284–299, 2009.

[PBB+02] Wolfgang J. Paul, Peter Bach, Michael Bosch, Jörg Fischer, Cédric Lichtenau, and Jochen Röhrig. Real PRAM Programming. In *Proc. Euro-Par'02*, August 2002.

[WT11a] Martin Wimmer and Jesper Larsson Träff. Work-stealing for mixed-mode parallelism by deterministic team-building. In *23rd ACM Symposium on Parallelism in Algorithms and Architectures (SPAA 2011)*, pages 105–115, 2011.

[WT11b] Martin Wimmer and Jesper Larsson Träff. A work-stealing framework for mixed-mode parallel applications. In *16th Int. Worksh. on Multi-threaded Architectures and Applications (MTAAP) at Int. Par. and Distr. Processing Symp. (IPDPS 2011)*, 2011.

[WV08] X. Wen and U. Vishkin. FPGA-based prototype of a PRAM-on-chip processor. In *Proc. ACM Computing Frontiers, Ischia, Italy*, May 2008.

GI-Edition Lecture Notes in Informatics

P-1 Gregor Engels, Andreas Oberweis, Albert Zündorf (Hrsg.): Modellierung 2001.

P-2 Mikhail Godlevsky, Heinrich C. Mayr (Hrsg.): Information Systems Technology and its Applications, ISTA'2001.

P-3 Ana M. Moreno, Reind P. van de Riet (Hrsg.): Applications of Natural Lan-guage to Information Systems, NLDB'2001.

P-4 H. Wörn, J. Mühling, C. Vahl, H.-P. Meinzer (Hrsg.): Rechner- und sensorgestützte Chirurgie; Workshop des SFB 414.

P-5 Andy Schürr (Hg.): OMER – Object-Oriented Modeling of Embedded Real-Time Systems.

P-6 Hans-Jürgen Appelrath, Rolf Beyer, Uwe Marquardt, Heinrich C. Mayr, Claudia Steinberger (Hrsg.): Unternehmen Hochschule, UH'2001.

P-7 Andy Evans, Robert France, Ana Moreira, Bernhard Rumpe (Hrsg.): Practical UML-Based Rigorous Development Methods – Countering or Integrating the extremists, pUML'2001.

P-8 Reinhard Keil-Slawik, Johannes Magenheim (Hrsg.): Informatikunterricht und Medienbildung, INFOS'2001.

P-9 Jan von Knop, Wilhelm Haverkamp (Hrsg.): Innovative Anwendungen in Kommunikationsnetzen, 15. DFN Arbeitstagung.

P-10 Mirjam Minor, Steffen Staab (Hrsg.): 1st German Workshop on Experience Management: Sharing Experiences about the Sharing Experience.

P-11 Michael Weber, Frank Kargl (Hrsg.): Mobile Ad-Hoc Netzwerke, WMAN 2002.

P-12 Martin Glinz, Günther Müller-Luschnat (Hrsg.): Modellierung 2002.

P-13 Jan von Knop, Peter Schirmbacher and Viljan Mahni_ (Hrsg.): The Changing Universities – The Role of Technology.

P-14 Robert Tolksdorf, Rainer Eckstein (Hrsg.): XML-Technologien für das Semantic Web – XSW 2002.

P-15 Hans-Bernd Bludau, Andreas Koop (Hrsg.): Mobile Computing in Medicine.

P-16 J. Felix Hampe, Gerhard Schwabe (Hrsg.): Mobile and Collaborative Business 2002.

P-17 Jan von Knop, Wilhelm Haverkamp (Hrsg.): Zukunft der Netze –Die Verletzbarkeit meistern, 16. DFN Arbeitstagung.

P-18 Elmar J. Sinz, Markus Plaha (Hrsg.): Modellierung betrieblicher Informationssysteme – MobIS 2002.

P-19 Sigrid Schubert, Bernd Reusch, Norbert Jesse (Hrsg.): Informatik bewegt – Informatik 2002 – 32. Jahrestagung der Gesellschaft für Informatik e.V. (GI) 30.Sept.-3. Okt. 2002 in Dortmund.

P-20 Sigrid Schubert, Bernd Reusch, Norbert Jesse (Hrsg.): Informatik bewegt – Informatik 2002 – 32. Jahrestagung der Gesellschaft für Informatik e.V. (GI) 30.Sept.-3. Okt. 2002 in Dortmund (Ergänzungsband).

P-21 Jörg Desel, Mathias Weske (Hrsg.): Promise 2002: Prozessorientierte Methoden und Werkzeuge für die Entwicklung von Informationssystemen.

P-22 Sigrid Schubert, Johannes Magenheim, Peter Hubwieser, Torsten Brinda (Hrsg.): Forschungsbeiträge zur "Didaktik der Informatik" – Theorie, Praxis, Evaluation.

P-23 Thorsten Spitta, Jens Borchers, Harry M. Sneed (Hrsg.): Software Management 2002 – Fortschritt durch Beständigkeit

P-24 Rainer Eckstein, Robert Tolksdorf (Hrsg.): XMIDX 2003 – XML-Technologien für Middleware – Middleware für XML-Anwendungen

P-25 Key Pousttchi, Klaus Turowski (Hrsg.): Mobile Commerce – Anwendungen und Perspektiven – 3. Workshop Mobile Commerce, Universität Augsburg, 04.02.2003

P-26 Gerhard Weikum, Harald Schöning, Erhard Rahm (Hrsg.): BTW 2003: Datenbanksysteme für Business, Technologie und Web

P-27 Michael Kroll, Hans-Gerd Lipinski, Kay Melzer (Hrsg.): Mobiles Computing in der Medizin

P-28 Ulrich Reimer, Andreas Abecker, Steffen Staab, Gerd Stumme (Hrsg.): WM 2003: Professionelles Wissensmanagement – Er-fahrungen und Visionen

P-29 Antje Düsterhöft, Bernhard Thalheim (Eds.): NLDB'2003: Natural Language Processing and Information Systems

P-30 Mikhail Godlevsky, Stephen Liddle, Heinrich C. Mayr (Eds.): Information Systems Technology and its Applications

P-31 Arslan Brömme, Christoph Busch (Eds.): BIOSIG 2003: Biometrics and Electronic Signatures

P-32 Peter Hubwieser (Hrsg.): Informatische Fachkonzepte im Unterricht – INFOS 2003

P-33 Andreas Geyer-Schulz, Alfred Taudes (Hrsg.): Informationswirtschaft: Ein Sektor mit Zukunft

P-34 Klaus Dittrich, Wolfgang König, Andreas Oberweis, Kai Rannenberg, Wolfgang Wahlster (Hrsg.): Informatik 2003 – Innovative Informatikanwendungen (Band 1)

P-35 Klaus Dittrich, Wolfgang König, Andreas Oberweis, Kai Rannenberg, Wolfgang Wahlster (Hrsg.): Informatik 2003 – Innovative Informatikanwendungen (Band 2)

P-36 Rüdiger Grimm, Hubert B. Keller, Kai Rannenberg (Hrsg.): Informatik 2003 – Mit Sicherheit Informatik

P-37 Arndt Bode, Jörg Desel, Sabine Rathmayer, Martin Wessner (Hrsg.): DeLFI 2003: e-Learning Fachtagung Informatik

P-38 E.J. Sinz, M. Plaha, P. Neckel (Hrsg.): Modellierung betrieblicher Informationssysteme – MobIS 2003

P-39 Jens Nedon, Sandra Frings, Oliver Göbel (Hrsg.): IT-Incident Management & IT-Forensics – IMF 2003

P-40 Michael Rebstock (Hrsg.): Modellierung betrieblicher Informationssysteme – MobIS 2004

P-41 Uwe Brinkschulte, Jürgen Becker, Dietmar Fey, Karl-Erwin Großpietsch, Christian Hochberger, Erik Maehle, Thomas Runkler (Edts.): ARCS 2004 – Organic and Pervasive Computing

P-42 Key Pousttchi, Klaus Turowski (Hrsg.): Mobile Economy – Transaktionen und Prozesse, Anwendungen und Dienste

P-43 Birgitta König-Ries, Michael Klein, Philipp Obreiter (Hrsg.): Persistance, Scalability, Transactions – Database Mechanisms for Mobile Applications

P-44 Jan von Knop, Wilhelm Haverkamp, Eike Jessen (Hrsg.): Security, E-Learning. E-Services

P-45 Bernhard Rumpe, Wofgang Hesse (Hrsg.): Modellierung 2004

P-46 Ulrich Flegel, Michael Meier (Hrsg.): Detection of Intrusions of Malware & Vulnerability Assessment

P-47 Alexander Prosser, Robert Krimmer (Hrsg.): Electronic Voting in Europe – Technology, Law, Politics and Society

P-48 Anatoly Doroshenko, Terry Halpin, Stephen W. Liddle, Heinrich C. Mayr (Hrsg.): Information Systems Technology and its Applications

P-49 G. Schiefer, P. Wagner, M. Morgenstern, U. Rickert (Hrsg.): Integration und Datensicherheit – Anforderungen, Konflikte und Perspektiven

P-50 Peter Dadam, Manfred Reichert (Hrsg.): INFORMATIK 2004 – Informatik verbindet (Band 1) Beiträge der 34. Jahrestagung der Gesellschaft für Informatik e.V. (GI), 20.-24. September 2004 in Ulm

P-51 Peter Dadam, Manfred Reichert (Hrsg.): INFORMATIK 2004 – Informatik verbindet (Band 2) Beiträge der 34. Jahrestagung der Gesellschaft für Informatik e.V. (GI), 20.-24. September 2004 in Ulm

P-52 Gregor Engels, Silke Seehusen (Hrsg.): DELFI 2004 – Tagungsband der 2. e-Learning Fachtagung Informatik

P-53 Robert Giegerich, Jens Stoye (Hrsg.): German Conference on Bioinformatics – GCB 2004

P-54 Jens Borchers, Ralf Kneuper (Hrsg.): Softwaremanagement 2004 – Outsourcing und Integration

P-55 Jan von Knop, Wilhelm Haverkamp, Eike Jessen (Hrsg.): E-Science und Grid Adhoc-Netze Medienintegration

P-56 Fernand Feltz, Andreas Oberweis, Benoit Otjacques (Hrsg.): EMISA 2004 – Informationssysteme im E-Business und E-Government

P-57 Klaus Turowski (Hrsg.): Architekturen, Komponenten, Anwendungen

P-58 Sami Beydeda, Volker Gruhn, Johannes Mayer, Ralf Reussner, Franz Schweiggert (Hrsg.): Testing of Component-Based Systems and Software Quality

P-59 J. Felix Hampe, Franz Lehner, Key Pousttchi, Kai Ranneberg, Klaus Turowski (Hrsg.): Mobile Business – Processes, Platforms, Payments

P-60 Steffen Friedrich (Hrsg.): Unterrichtskonzepte für inforrmatische Bildung

P-61 Paul Müller, Reinhard Gotzhein, Jens B. Schmitt (Hrsg.): Kommunikation in verteilten Systemen

P-62 Federrath, Hannes (Hrsg.): „Sicherheit 2005“ – Sicherheit – Schutz und Zuverlässigkeit

P-63 Roland Kaschek, Heinrich C. Mayr, Stephen Liddle (Hrsg.): Information Systems – Technology and ist Applications

P-64 Peter Liggesmeyer, Klaus Pohl, Michael Goedicke (Hrsg.): Software Engineering 2005

P-65 Gottfried Vossen, Frank Leymann, Peter Lockemann, Wolffried Stucky (Hrsg.): Datenbanksysteme in Business, Technologie und Web

P-66 Jörg M. Haake, Ulrike Lucke, Djamshid Tavangarian (Hrsg.): DeLFI 2005: 3. deutsche e-Learning Fachtagung Informatik

P-67 Armin B. Cremers, Rainer Manthey, Peter Martini, Volker Steinhage (Hrsg.): INFORMATIK 2005 – Informatik LIVE (Band 1)

P-68 Armin B. Cremers, Rainer Manthey, Peter Martini, Volker Steinhage (Hrsg.): INFORMATIK 2005 – Informatik LIVE (Band 2)

P-69 Robert Hirschfeld, Ryszard Kowalcyk, Andreas Polze, Matthias Weske (Hrsg.): NODe 2005, GSEM 2005

P-70 Klaus Turowski, Johannes-Maria Zaha (Hrsg.): Component-oriented Enterprise Application (COAE 2005)

P-71 Andrew Torda, Stefan Kurz, Matthias Rarey (Hrsg.): German Conference on Bioinformatics 2005

P-72 Klaus P. Jantke, Klaus-Peter Fähnrich, Wolfgang S. Wittig (Hrsg.): Marktplatz Internet: Von e-Learning bis e-Payment

P-73 Jan von Knop, Wilhelm Haverkamp, Eike Jessen (Hrsg.): “Heute schon das Morgen sehen“

P-74 Christopher Wolf, Stefan Lucks, Po-Wah Yau (Hrsg.): WEWoRC 2005 – Western European Workshop on Research in Cryptology

P-75 Jörg Desel, Ulrich Frank (Hrsg.): Enterprise Modelling and Information Systems Architecture

P-76 Thomas Kirste, Birgitta König-Riess, Key Pousttchi, Klaus Turowski (Hrsg.): Mobile Informationssysteme – Potentiale, Hindernisse, Einsatz

P-77 Jana Dittmann (Hrsg.): SICHERHEIT 2006

P-78 K.-O. Wenkel, P. Wagner, M. Morgenstern, K. Luzi, P. Eisermann (Hrsg.): Land- und Ernährungswirtschaft im Wandel

P-79 Bettina Biel, Matthias Book, Volker Gruhn (Hrsg.): Softwareengineering 2006

P-80 Mareike Schoop, Christian Huemer, Michael Rebstock, Martin Bichler (Hrsg.): Service-Oriented Electronic Commerce

P-81 Wolfgang Karl, Jürgen Becker, Karl-Erwin Großpietsch, Christian Hochberger, Erik Maehle (Hrsg.): ARCS´06

P-82 Heinrich C. Mayr, Ruth Breu (Hrsg.): Modellierung 2006

P-83 Daniel Huson, Oliver Kohlbacher, Andrei Lupas, Kay Nieselt and Andreas Zell (eds.): German Conference on Bioinformatics

P-84 Dimitris Karagiannis, Heinrich C. Mayr, (Hrsg.): Information Systems Technology and its Applications

P-85 Witold Abramowicz, Heinrich C. Mayr, (Hrsg.): Business Information Systems

P-86 Robert Krimmer (Ed.): Electronic Voting 2006

P-87 Max Mühlhäuser, Guido Rößling, Ralf Steinmetz (Hrsg.): DELFI 2006: 4. e-Learning Fachtagung Informatik

P-88 Robert Hirschfeld, Andreas Polze, Ryszard Kowalczyk (Hrsg.): NODe 2006, GSEM 2006

P-90 Joachim Schelp, Robert Winter, Ulrich Frank, Bodo Rieger, Klaus Turowski (Hrsg.): Integration, Informationslogistik und Architektur

P-91 Henrik Stormer, Andreas Meier, Michael Schumacher (Eds.): European Conference on eHealth 2006

P-92 Fernand Feltz, Benoît Otjacques, Andreas Oberweis, Nicolas Poussing (Eds.): AIM 2006

P-93 Christian Hochberger, Rüdiger Liskowsky (Eds.): INFORMATIK 2006 – Informatik für Menschen, Band 1

P-94 Christian Hochberger, Rüdiger Liskowsky (Eds.): INFORMATIK 2006 – Informatik für Menschen, Band 2

P-95 Matthias Weske, Markus Nüttgens (Eds.): EMISA 2005: Methoden, Konzepte und Technologien für die Entwicklung von dienstbasierten Informationssystemen

P-96 Saartje Brockmans, Jürgen Jung, York Sure (Eds.): Meta-Modelling and Ontologies

P-97 Oliver Göbel, Dirk Schadt, Sandra Frings, Hardo Hase, Detlef Günther, Jens Nedon (Eds.): IT-Incident Mangament & IT-Forensics – IMF 2006

P-98 Hans Brandt-Pook, Werner Simonsmeier und Thorsten Spitta (Hrsg.): Beratung in der Softwareentwicklung – Modelle, Methoden, Best Practices

P-99 Andreas Schwill, Carsten Schulte, Marco Thomas (Hrsg.): Didaktik der Informatik

P-100 Peter Forbrig, Günter Siegel, Markus Schneider (Hrsg.): HDI 2006: Hochschuldidaktik der Informatik

P-101 Stefan Böttinger, Ludwig Theuvsen, Susanne Rank, Marlies Morgenstern (Hrsg.): Agrarinformatik im Spannungsfeld zwischen Regionalisierung und globalen Wertschöpfungsketten

P-102 Otto Spaniol (Eds.): Mobile Services and Personalized Environments

P-103 Alfons Kemper, Harald Schöning, Thomas Rose, Matthias Jarke, Thomas Seidl, Christoph Quix, Christoph Brochhaus (Hrsg.): Datenbanksysteme in Business, Technologie und Web (BTW 2007)

P-104 Birgitta König-Ries, Franz Lehner, Rainer Malaka, Can Türker (Hrsg.) MMS 2007: Mobilität und mobile Informationssysteme

P-105 Wolf-Gideon Bleek, Jörg Raasch, Heinz Züllighoven (Hrsg.) Software Engineering 2007

P-106 Wolf-Gideon Bleek, Henning Schwentner, Heinz Züllighoven (Hrsg.) Software Engineering 2007 – Beiträge zu den Workshops

P-107 Heinrich C. Mayr, Dimitris Karagiannis (eds.) Information Systems Technology and its Applications

P-108 Arslan Brömme, Christoph Busch, Detlef Hühnlein (eds.) BIOSIG 2007: Biometrics and Electronic Signatures

P-109 Rainer Koschke, Otthein Herzog, Karl-Heinz Rödiger, Marc Ronthaler (Hrsg.) INFORMATIK 2007 Informatik trifft Logistik Band 1

P-110 Rainer Koschke, Otthein Herzog, Karl-Heinz Rödiger, Marc Ronthaler (Hrsg.) INFORMATIK 2007 Informatik trifft Logistik Band 2

P-111 Christian Eibl, Johannes Magenheim, Sigrid Schubert, Martin Wessner (Hrsg.) DeLFI 2007: 5. e-Learning Fachtagung Informatik

P-112 Sigrid Schubert (Hrsg.) Didaktik der Informatik in Theorie und Praxis

P-113 Sören Auer, Christian Bizer, Claudia Müller, Anna V. Zhdanova (Eds.) The Social Semantic Web 2007 Proceedings of the 1st Conference on Social Semantic Web (CSSW)

P-114 Sandra Frings, Oliver Göbel, Detlef Günther, Hardo G. Hase, Jens Nedon, Dirk Schadt, Arslan Brömme (Eds.) IMF2007 IT-incident management & IT-forensics Proceedings of the 3rd International Conference on IT-Incident Management & IT-Forensics

P-115 Claudia Falter, Alexander Schliep, Joachim Selbig, Martin Vingron and Dirk Walther (Eds.) German conference on bioinformatics GCB 2007

P-116 Witold Abramowicz, Leszek Maciszek (Eds.) Business Process and Services Computing 1st International Working Conference on Business Process and Services Computing BPSC 2007

P-117 Ryszard Kowalczyk (Ed.) Grid service engineering and manegement The 4th International Conference on Grid Service Engineering and Management GSEM 2007

P-118 Andreas Hein, Wilfried Thoben, Hans-Jürgen Appelrath, Peter Jensch (Eds.) European Conference on ehealth 2007

P-119 Manfred Reichert, Stefan Strecker, Klaus Turowski (Eds.) Enterprise Modelling and Information Systems Architectures Concepts and Applications

P-120 Adam Pawlak, Kurt Sandkuhl, Wojciech Cholewa, Leandro Soares Indrusiak (Eds.) Coordination of Collaborative Engineering - State of the Art and Future Challenges

P-121 Korbinian Herrmann, Bernd Bruegge (Hrsg.) Software Engineering 2008 Fachtagung des GI-Fachbereichs Softwaretechnik

P-122 Walid Maalej, Bernd Bruegge (Hrsg.) Software Engineering 2008 - Workshopband Fachtagung des GI-Fachbereichs Softwaretechnik

P-123 Michael H. Breitner, Martin Breunig, Elgar Fleisch, Ley Pousttchi, Klaus Turowski (Hrsg.)
Mobile und Ubiquitäre Informationssysteme – Technologien, Prozesse, Marktfähigkeit
Proceedings zur 3. Konferenz Mobile und Ubiquitäre Informationssysteme (MMS 2008)

P-124 Wolfgang E. Nagel, Rolf Hoffmann, Andreas Koch (Eds.)
9th Workshop on Parallel Systems and Algorithms (PASA)
Workshop of the GI/ITG Speciel Interest Groups PARS and PARVA

P-125 Rolf A.E. Müller, Hans-H. Sundermeier, Ludwig Theuvsen, Stephanie Schütze, Marlies Morgenstern (Hrsg.)
Unternehmens-IT:
Führungsinstrument oder Verwaltungsbürde
Referate der 28. GIL Jahrestagung

P-126 Rainer Gimnich, Uwe Kaiser, Jochen Quante, Andreas Winter (Hrsg.)
10th Workshop Software Reengineering (WSR 2008)

P-127 Thomas Kühne, Wolfgang Reisig, Friedrich Steimann (Hrsg.)
Modellierung 2008

P-128 Ammar Alkassar, Jörg Siekmann (Hrsg.)
Sicherheit 2008
Sicherheit, Schutz und Zuverlässigkeit
Beiträge der 4. Jahrestagung des Fachbereichs Sicherheit der Gesellschaft für Informatik e.V. (GI)
2.-4. April 2008
Saarbrücken, Germany

P-129 Wolfgang Hesse, Andreas Oberweis (Eds.)
Sigsand-Europe 2008
Proceedings of the Third AIS SIGSAND European Symposium on Analysis, Design, Use and Societal Impact of Information Systems

P-130 Paul Müller, Bernhard Neumair, Gabi Dreo Rodosek (Hrsg.)
1. DFN-Forum Kommunikations-technologien Beiträge der Fachtagung

P-131 Robert Krimmer, Rüdiger Grimm (Eds.)
3rd International Conference on Electronic Voting 2008
Co-organized by Council of Europe, Gesellschaft für Informatik and E-Voting. CC

P-132 Silke Seehusen, Ulrike Lucke, Stefan Fischer (Hrsg.)
DeLFI 2008:
Die 6. e-Learning Fachtagung Informatik

P-133 Heinz-Gerd Hegering, Axel Lehmann, Hans Jürgen Ohlbach, Christian Scheideler (Hrsg.)
INFORMATIK 2008
Beherrschbare Systeme – dank Informatik
Band 1

P-134 Heinz-Gerd Hegering, Axel Lehmann, Hans Jürgen Ohlbach, Christian Scheideler (Hrsg.)
INFORMATIK 2008
Beherrschbare Systeme – dank Informatik
Band 2

P-135 Torsten Brinda, Michael Fothe, Peter Hubwieser, Kirsten Schlüter (Hrsg.)
Didaktik der Informatik –
Aktuelle Forschungsergebnisse

P-136 Andreas Beyer, Michael Schroeder (Eds.)
German Conference on Bioinformatics GCB 2008

P-137 Arslan Brömme, Christoph Busch, Detlef Hühnlein (Eds.)
BIOSIG 2008: Biometrics and Electronic Signatures

P-138 Barbara Dinter, Robert Winter, Peter Chamoni, Norbert Gronau, Klaus Turowski (Hrsg.)
Synergien durch Integration und Informationslogistik
Proceedings zur DW2008

P-139 Georg Herzwurm, Martin Mikusz (Hrsg.)
Industrialisierung des Software-Managements
Fachtagung des GI-Fachausschusses Management der Anwendungsentwicklung und -wartung im Fachbereich Wirtschaftsinformatik

P-140 Oliver Göbel, Sandra Frings, Detlef Günther, Jens Nedon, Dirk Schadt (Eds.)
IMF 2008 - IT Incident Management & IT Forensics

P-141 Peter Loos, Markus Nüttgens, Klaus Turowski, Dirk Werth (Hrsg.)
Modellierung betrieblicher Informations-systeme (MobIS 2008)
Modellierung zwischen SOA und Compliance Management

P-142 R. Bill, P. Korduan, L. Theuvsen, M. Morgenstern (Hrsg.)
Anforderungen an die Agrarinformatik durch Globalisierung und Klimaveränderung

P-143 Peter Liggesmeyer, Gregor Engels, Jürgen Münch, Jörg Dörr, Norman Riegel (Hrsg.)
Software Engineering 2009
Fachtagung des GI-Fachbereichs Softwaretechnik

P-144 Johann-Christoph Freytag, Thomas Ruf, Wolfgang Lehner, Gottfried Vossen (Hrsg.)
Datenbanksysteme in Business, Technologie und Web (BTW)

P-145 Knut Hinkelmann, Holger Wache (Eds.)
WM2009: 5th Conference on Professional Knowledge Management

P-146 Markus Bick, Martin Breunig, Hagen Höpfner (Hrsg.)
Mobile und Ubiquitäre Informationssysteme – Entwicklung, Implementierung und Anwendung
4. Konferenz Mobile und Ubiquitäre Informationssysteme (MMS 2009)

P-147 Witold Abramowicz, Leszek Maciaszek, Ryszard Kowalczyk, Andreas Speck (Eds.)
Business Process, Services Computing and Intelligent Service Management
BPSC 2009 · ISM 2009 · YRW-MBP 2009

P-148 Christian Erfurth, Gerald Eichler, Volkmar Schau (Eds.)
9th International Conference on Innovative Internet Community Systems
I^{2}CS 2009

P-149 Paul Müller, Bernhard Neumair, Gabi Dreo Rodosek (Hrsg.)
2. DFN-Forum Kommunikationstechnologien
Beiträge der Fachtagung

P-150 Jürgen Münch, Peter Liggesmeyer (Hrsg.)
Software Engineering
2009 - Workshopband

P-151 Armin Heinzl, Peter Dadam, Stefan Kirn, Peter Lockemann (Eds.)
PRIMIUM
Process Innovation for Enterprise Software

P-152 Jan Mendling, Stefanie Rinderle-Ma, Werner Esswein (Eds.)
Enterprise Modelling and Information Systems Architectures
Proceedings of the 3rd Int'l Workshop EMISA 2009

P-153 Andreas Schwill, Nicolas Apostolopoulos (Hrsg.)
Lernen im Digitalen Zeitalter
DeLFI 2009 – Die 7. E-Learning Fachtagung Informatik

P-154 Stefan Fischer, Erik Maehle Rüdiger Reischuk (Hrsg.)
INFORMATIK 2009
Im Focus das Leben

P-155 Arslan Brömme, Christoph Busch, Detlef Hühnlein (Eds.)
BIOSIG 2009:
Biometrics and Electronic Signatures
Proceedings of the Special Interest Group on Biometrics and Electronic Signatures

P-156 Bernhard Koerber (Hrsg.)
Zukunft braucht Herkunft
25 Jahre »INFOS – Informatik und Schule«

P-157 Ivo Grosse, Steffen Neumann, Stefan Posch, Falk Schreiber, Peter Stadler (Eds.)
German Conference on Bioinformatics 2009

P-158 W. Claupein, L. Theuvsen, A. Kämpf, M. Morgenstern (Hrsg.)
Precision Agriculture Reloaded – Informationsgestützte Landwirtschaft

P-159 Gregor Engels, Markus Luckey, Wilhelm Schäfer (Hrsg.)
Software Engineering 2010

P-160 Gregor Engels, Markus Luckey, Alexander Pretschner, Ralf Reussner (Hrsg.)
Software Engineering 2010 – Workshopband
(inkl. Doktorandensymposium)

P-161 Gregor Engels, Dimitris Karagiannis Heinrich C. Mayr (Hrsg.)
Modellierung 2010

P-162 Maria A. Wimmer, Uwe Brinkhoff, Siegfried Kaiser, Dagmar Lück-Schneider, Erich Schweighofer, Andreas Wiebe (Hrsg.)
Vernetzte IT für einen effektiven Staat
Gemeinsame Fachtagung Verwaltungsinformatik (FTVI) und Fachtagung Rechtsinformatik (FTRI) 2010

P-163 Markus Bick, Stefan Eulgem, Elgar Fleisch, J. Felix Hampe, Birgitta König-Ries, Franz Lehner, Key Pousttchi, Kai Rannenberg (Hrsg.)
Mobile und Ubiquitäre Informationssysteme
Technologien, Anwendungen und Dienste zur Unterstützung von mobiler Kollaboration

P-164 Arslan Brömme, Christoph Busch (Eds.)
BIOSIG 2010: Biometrics and Electronic Signatures Proceedings of the Special Interest Group on Biometrics and Electronic Signatures

P-165 Gerald Eichler, Peter Kropf, Ulrike Lechner, Phayung Meesad, Herwig Unger (Eds.)
10th International Conference on Innovative Internet Community Systems (I^2CS) – Jubilee Edition 2010 –

P-166 Paul Müller, Bernhard Neumair, Gabi Dreo Rodosek (Hrsg.)
3. DFN-Forum Kommunikationstechnologien
Beiträge der Fachtagung

P-167 Robert Krimmer, Rüdiger Grimm (Eds.)
4th International Conference on Electronic Voting 2010
co-organized by the Council of Europe, Gesellschaft für Informatik and E-Voting.CC

P-168 Ira Diethelm, Christina Dörge, Claudia Hildebrandt, Carsten Schulte (Hrsg.)
Didaktik der Informatik
Möglichkeiten empirischer Forschungsmethoden und Perspektiven der Fachdidaktik

P-169 Michael Kerres, Nadine Ojstersek Ulrik Schroeder, Ulrich Hoppe (Hrsg.)
DeLFI 2010 - 8. Tagung der Fachgruppe E-Learning der Gesellschaft für Informatik e.V.

P-170 Felix C. Freiling (Hrsg.)
Sicherheit 2010
Sicherheit, Schutz und Zuverlässigkeit

P-171 Werner Esswein, Klaus Turowski, Martin Juhrisch (Hrsg.)
Modellierung betrieblicher Informationssysteme (MobIS 2010)
Modellgestütztes Management

P-172 Stefan Klink, Agnes Koschmider Marco Mevius, Andreas Oberweis (Hrsg.)
EMISA 2010
Einflussfaktoren auf die Entwicklung flexibler, integrierter Informationssysteme
Beiträge des Workshops der GI-Fachgruppe EMISA (Entwicklungsmethoden für Informationssysteme und deren Anwendung)

P-173 Dietmar Schomburg, Andreas Grote (Eds.)
German Conference on Bioinformatics 2010

P-174 Arslan Brömme, Torsten Eymann, Detlef Hühnlein, Heiko Roßnagel, Paul Schmücker (Hrsg.)
perspeGKtive 2010
Workshop „Innovative und sichere Informationstechnologie für das Gesundheitswesen von morgen“

P-175 Klaus-Peter Fähnrich, Bogdan Franczyk (Hrsg.)
INFORMATIK 2010
Service Science – Neue Perspektiven für die Informatik
Band 1

P-176 Klaus-Peter Fähnrich, Bogdan Franczyk (Hrsg.)
INFORMATIK 2010
Service Science – Neue Perspektiven für die Informatik
Band 2

P-177 Witold Abramowicz, Rainer Alt, Klaus-Peter Fähnrich, Bogdan Franczyk, Leszek A. Maciaszek (Eds.)
INFORMATIK 2010
Business Process and Service Science – Proceedings of ISSS and BPSC

P-178 Wolfram Pietsch, Benedikt Krams (Hrsg.)
Vom Projekt zum Produkt
Fachtagung des GI-Fachausschusses Management der Anwendungsentwicklung und -wartung im Fachbereich Wirtschafts-informatik (WI-MAW), Aachen, 2010

P-179 Stefan Gruner, Bernhard Rumpe (Eds.)
FM+AM`2010
Second International Workshop on Formal Methods and Agile Methods

P-180 Theo Härder, Wolfgang Lehner, Bernhard Mitschang, Harald Schöning, Holger Schwarz (Hrsg.)
Datenbanksysteme für Business, Technologie und Web (BTW)
14. Fachtagung des GI-Fachbereichs „Datenbanken und Informationssysteme“ (DBIS)

P-181 Michael Clasen, Otto Schätzel, Brigitte Theuvsen (Hrsg.)
Qualität und Effizienz durch informationsgestützte Landwirtschaft, Fokus: Moderne Weinwirtschaft

P-182 Ronald Maier (Hrsg.)
6th Conference on Professional Knowledge Management
From Knowledge to Action

P-183 Ralf Reussner, Matthias Grund, Andreas Oberweis, Walter Tichy (Hrsg.)
Software Engineering 2011
Fachtagung des GI-Fachbereichs Softwaretechnik

P-184 Ralf Reussner, Alexander Pretschner, Stefan Jähnichen (Hrsg.)
Software Engineering 2011
Workshopband
(inkl. Doktorandensymposium)

P-185 Hagen Höpfner, Günther Specht, Thomas Ritz, Christian Bunse (Hrsg.) MMS 2011: Mobile und ubiquitäre Informationssysteme Proceedings zur 6. Konferenz Mobile und Ubiquitäre Informationssysteme (MMS 2011)

P-186 Gerald Eichler, Axel Küpper, Volkmar Schau, Hacène Fouchal, Herwig Unger (Eds.) 11th International Conference on Innovative Internet Community Systems (I^2CS)

P-187 Paul Müller, Bernhard Neumair, Gabi Dreo Rodosek (Hrsg.) 4. DFN-Forum Kommunikations-technologien, Beiträge der Fachtagung 20. Juni bis 21. Juni 2011 Bonn

P-188 Holger Rohland, Andrea Kienle, Steffen Friedrich (Hrsg.) DeLFI 2011 – Die 9. e-Learning Fachtagung Informatik der Gesellschaft für Informatik e.V. 5.–8. September 2011, Dresden

P-189 Thomas, Marco (Hrsg.) Informatik in Bildung und Beruf INFOS 2011 14. GI-Fachtagung Informatik und Schule

P-190 Markus Nüttgens, Oliver Thomas, Barbara Weber (Eds.) Enterprise Modelling and Information Systems Architectures (EMISA 2011)

P-191 Arslan Brömme, Christoph Busch (Eds.) BIOSIG 2011 International Conference of the Biometrics Special Interest Group

P-192 Hans-Ulrich Heiß, Peter Pepper, Holger Schlingloff, Jörg Schneider (Hrsg.) INFORMATIK 2011 Informatik schafft Communities

P-193 Wolfgang Lehner, Gunther Piller (Hrsg.) IMDM 2011

P-194 M. Clasen, G. Fröhlich, H. Bernhardt, K. Hildebrand, B. Theuvsen (Hrsg.) Informationstechnologie für eine nachhaltige Landbewirtschaftung Fokus Forstwirtschaft

P-195 Neeraj Suri, Michael Waidner (Hrsg.) Sicherheit 2012 Sicherheit, Schutz und Zuverlässigkeit Beiträge der 6. Jahrestagung des Fachbereichs Sicherheit der Gesellschaft für Informatik e.V. (GI)

P-198 Stefan Jähnichen, Axel Küpper, Sahin Albayrak (Hrsg.) Software Engineering 2012 Fachtagung des GI-Fachbereichs Softwaretechnik

P-200 Gero Mühl, Jan Richling, Andreas Herkersdorf (Hrsg.) ARCS 2012 Workshops

The titles can be purchased at:

Köllen Druck + Verlag GmbH
Ernst-Robert-Curtius-Str. 14 · D-53117 Bonn
Fax: +49 (0)228/9898222
E-Mail: druckverlag@koellen.de

Gesellschaft für Informatik e.V. (GI)

publishes this series in order to make available to a broad public recent findings in informatics (i.e. computer science and information systems), to document conferences that are organized in cooperation with GI and to publish the annual GI Award dissertation.

Broken down into
• seminars
• proceedings
• dissertations
• thematics
current topics are dealt with from the vantage point of research and development, teaching and further training in theory and practice. The Editorial Committee uses an intensive review process in order to ensure high quality contributions.

The volumes are published in German or English.

Information: http://www.gi.de/service/publikationen/lni/

ISSN 1617-5468
ISBN 978-88579-294-9

ARCS 2012 Workshops is the eighth event in a conference series focusing on a broad range of modeling topics from a variety of perspectives. With its emphasis on lively discussions and cross-fertilization of academia and industry, it provides a valuable platform to further advance the state of the art in topics such as modeling foundations, methodologies, applications, and tools. This volume contains contributions from the refereed workshop program and a tutorial paper.