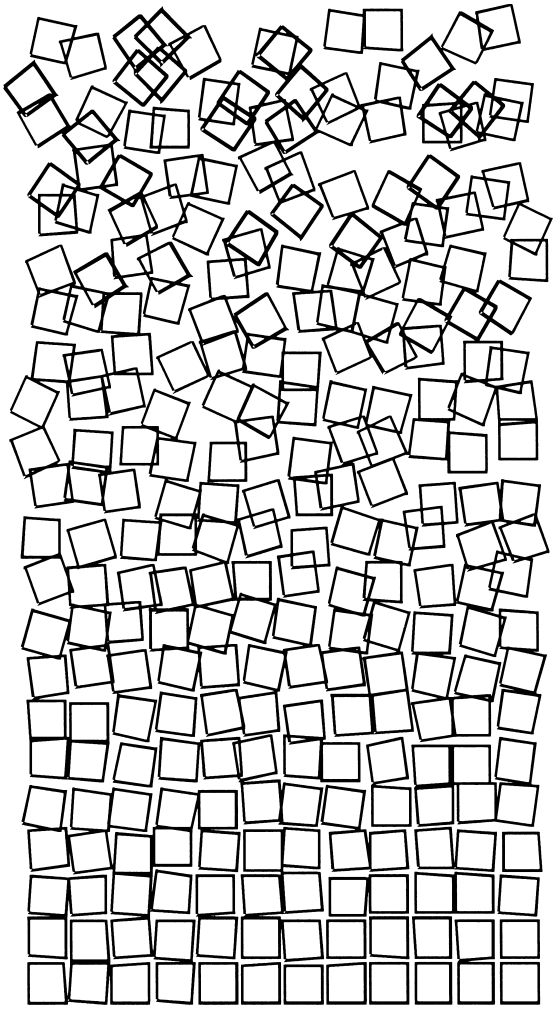
GESELLSCHAFT FÜR INFORMATIK E.V.

PARALLEL-ALGORITHMEN, -RECHNERSTRUKTUREN UND -SYSTEMSOFTWARE

PARS

INFORMATIONSTECHNISCHE GESELLSCHAFT IM VDE



Computergraphik von: Georg Nees, Generative Computergraphik

I n h a l t

Aktuelle PARS-Aktivitäten unter:

- **http://fg-pars.gi.de/**

ISSN 0177-0454

PARS-Mitteilungen

Gesellschaft für Informatik e.V.,
Parallel-Algorithmen, -Rechnerstrukturen
und -Systemsoftware

Offizielle bibliographische Bezeichnung bei Zitaten:
Mitteilungen - Gesellschaft für Informatik e. V.,
Parallel-Algorithmen und Rechnerstrukturen, ISSN 0177 - 0454

Die PARS-Mitteilungen erscheinen in der Regel einmal pro Jahr. Sie befassen sich mit allen Aspekten paralleler Algorithmen und deren Implementierung auf Rechenanlagen in Hard- und Software.

Die Beiträge werden nicht redigiert, sie stellen die Meinung des Autors dar. Ihr Erscheinen in diesen Mitteilungen bedeutet keine Einschränkung anderweitiger Publikation.

Die Homepage

http://fg-pars.gi.de/

vermittelt aktuelle Informationen über PARS.

CALL FOR PAPERS

12th Workshop on Parallel Systems and Algorithms

PASA 2016

in conjunction with

International Conference on Architecture of Computing Systems (ARCS 2016)
Nuremberg, Germany, April 4-5, 2016

organized by
GI/ITG-Fachgruppe 'Parallel-Algorithmen, -Rechnerstrukturen und -Systemsoftware' (PARS) and
GI-Fachgruppe 'Algorithmen' (ALGO)

The PASA workshop series has the goal to build a bridge between theory and practice in the area of parallel systems and algorithms. In this context practical problems which require theoretical investigations as well as the applicability of theoretical approaches and results to practice shall be discussed. An important aspect is communication and exchange of experience between various groups working in the area of parallel computing, e.g. in computer science, electrical engineering, physics or mathematics.

Topics of Interest include, but are not restricted to:

- *parallel architectures & storage systems*
- *parallel embedded systems*
- *ubiquitous and pervasive systems*
- *reconfigurable parallel computing*
- *data stream-oriented computing*
- *interconnection networks*
- *network and grid computing*
- *distributed and parallel multimedia systems*
- *parallel and distributed algorithms*
- *models of parallel computation*
- *scheduling and load balancing*
- *parallel programming languages*
- *software engineering for parallel systems*
- *parallel design patterns*
- *performance evaluation of parallel systems*

The workshop will comprise invited talks on current topics by leading experts in the field as well as submitted papers on original and previously unpublished research. Accepted papers will be published in the ARCS Workshop Proceedings as well as in the PARS Newsletter (ISSN 0177-0454). The conference languages are English (preferred) and German. Papers are required to be in English.

A prize of 500 € will be awarded to the best contribution presented personally based on a student's or Ph.D. thesis or project. Co-authors are allowed, the PhD degree should not have been awarded at the time of submission. Candidates apply for the prize by e-mail to the organizers when submitting the contribution. We expect that candidates are or become members of one the groups ALGO or PARS.

Important Dates

29th January 2016: Deadline for submission of full papers of 10 pages (in English, as pdf, using Springer LNCS style, see http://www.springer.de/comp/lncs/authors.html)
under: https://easychair.org/conferences/?conf=pasa20160 (no typo, 20160 is correct)
12th February 2016: Notification of the authors
18th February 2016: Final version for workshop proceedings

Program Committee

S. Albers (Munich), H. Burkhart (Basel), M. Dietzfelbinger (Ilmenau), A. Doering (Zurich), N. Eicker (Jülich)
T. Fahringer (Innsbruck), D. Fey (Erlangen), T. Hagerup (Augsburg), V. Heuveline (Heidelberg)
R. Hoffmann (Darmstadt), K. Jansen (Kiel), B. Juurlink (Berlin), W. Karl (Karlsruhe)
J. Keller (Hagen), Ch. Lengauer (Passau), E. Maehle (Lübeck), E. W. Mayr (Munich), U. Meyer (Frankfurt)
F. Meyer auf der Heide (Paderborn), W. Nagel (Dresden), M. Philippsen (Erlangen), K. D. Reinartz (Höchstadt)
Ch. Scheideler (Paderborn), H. Schmeck (Karlsruhe), B. Schnor (Potsdam), U. Schwiegelshohn (Dortmund)
P. Sobe (Dresden), T. Ungerer (Augsburg), R. Wanka (Erlangen)

Organisation

Prof. Dr. Jörg Keller, FernUniversität in Hagen, Fac. Math and Computer Science, 58084 Hagen, Germany, Phone/Fax +49-2331-987-376/308, E-Mail joerg.keller at fernuni-hagen.de

Prof. Dr. Rolf Wanka, Univ. Erlangen-Nuremberg, Dept. of Computer Science, 91058 Erlangen, Germany, Phone/Fax +49-9131-8525-152/149, E-Mail rwanka at cs.fau.de

12. PASA-Workshop (Full Papers)

Seite

Synchronization of MPI One-Sided Communication on a Non-Cache-Coherent Many-Core System

Steffen Christgau and Bettina Schnor
Institute for Computer Science
University of Potsdam
August-Bebel-Straße 89
14482 Potsdam, Germany
{christgau,schnor}@cs.uni-potsdam.de

***Abstract*—This paper discusses the design and implementation of MPI's general active target synchronization on the Intel Single-Chip Cloud Computer, a non-cache-coherent many-core CPU. Measurements show a performance benefit of a factor of four compared to the default SCC-tuned MPI implementation and demonstrate the feasibility of implementing efficiently a shared memory protocol despite the lack of cache coherence. Further, a classification of implementation designs of MPI's general active target synchronization is presented.**

I. Introduction

Cache coherence has been present in multi-CPU and multi-core CPUs since decades. However, with increasing memory bandwidths the bandwidth of the coherence interconnect traffic becomes challenging [1]. This problem gets even more critical with increasing core count in many-core CPUs. Therefore, the investigation of algorithms for non-cache-coherent architectures becomes an important topic.

Previous work [2] has shown that true one-sided communication based on shared memory systems is feasible even when cache coherence is managed in software. This paper presents the design and performance analysis of MPI's synchronization methods used in the SCOSCo approach [2] for one-sided communication (OSC) on the Intel Single Chip Cloud-Computer (SCC) [3], a non-cache-coherent (nCC) architecture.

Since version 2.0, the Message Passing Interface (MPI) standard includes one-sided communication which has been extended in the subsequent versions [4]. Figure 1 shows pseudo-code for only two communicating processes.

Within the MPI standard, memory for remote memory access (RMA) operations is logically attached to a *window* object. Remote memory is addressed together with that window object and the *rank*, the numerical process identifier. The creation of a window object is a collective operation, i.e. all processes inside a group (*communicator*) have to participate in the construction. `MPI_WIN_CREATE` can be used to create a window inside a given communicator and associates that communicator to the created window object. Subsequent RMA operations, like `PUT`, `GET` and `ACCUMULATE` [4, §11.3], operate with the returned window object, but must be non-blocking according to the standard.

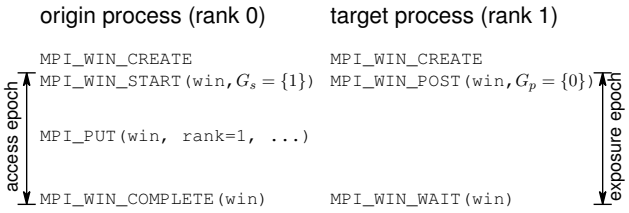


Fig. 1. PSCW Synchronization in MPI One-Sided Communication

II. OSC Synchronization

While message-passing communication includes both data transfer and synchronization between sender and receiver, these functions are separated in the OSC model. In general, RMA operations are only allowed after synchronization calls have been issued. An *origin* performs accesses during an *access epoch* only. Vice versa, a *target* allows such accesses only within an *exposure epoch*.

Previous work already discussed the optimization of MPI's fence synchronization [5]. This paper addresses *general active target communication*, which provides a flexible mechanism to synchronize processes. Based on the names of the methods that have to be invoked in that scheme, the synchronization is often also referred to as PSCW synchronization. An example of its usage for one origin and one target process is shown in Figure 1.

In contrast to the fence synchronization, PSCW allows to synchronize only a subset of the processes that created the window object. In addition, it allows an application to explicitly open access and exposure epochs. An origin opens and closes an access epoch with the `MPI_WIN_START` and the `MPI_WIN_COMPLETE` calls. In the `START` operation, an origin names the processes it (potentially) communicates with during the opened access epoch. The processes are given as a list of ranks, called *start group* G_s. On the other hand, a target invokes `MPI_WIN_POST` to open an exposure epoch and names the processes from which RMA operations are allowed (*post group*, G_p). However, the named processes are actually not required to access the target's window, but only these are allowed to do so. At the end of an exposure epoch, `MPI_WIN_WAIT` is issued to wait for a notification

TABLE I
CLASSIFICATION OF SYNCHRONIZATION PROTOCOLS FOR MPI ONE-SIDED COMMUNICATION.

class	epoch start	communication	overlap
deferred	non-blocking	delayed to epoch's end	no
immediate	blocking	prompt	yes
trigger-only	non-blocking	prompt	yes

that RMA operations have been completed and the window can be accesses without remote interference.

III. CLASSIFICATION OF MPI SYNCHRONIZATION IMPLEMENTATIONS

MPI implementations may implement the synchronization calls in different ways. Gropp and Thakur [6] define two options: *immediate* and *deferred*.

For deferred synchronization (used in MPICH, e.g.), the execution of methods that open epochs and perform RMA operation is delayed until the end of an epoch. This allows an MPI implementation to merge and optimize multiple of such communication calls. The downside is that optimizations like the overlap of communication and computation are not possible.

Within the immediate class (employed by MVAPICH for InfiniBand), the synchronization calls at the beginning of an epoch (`POST` and `START`) perform the synchronization immediately when they are invoked. Usually, this leads to blocking implementations. However, origin and target are ready for communication after synchronization. Since communication calls need to be non-blocking, immediate synchronization offers the possibility for communication computation overlap.

In addition to the classification from [6], we identify a third class that combines the advantages from deferred and immediate synchronization (see Table I). In the *trigger-only* variant, the starting synchronization calls initiate operations but do not block to wait for completion of the synchronization. This task is shifted to the communication calls, like `PUT` and `GET`, that check for those target processes to be synchronized. If the synchronization is still not finalized, the communication call blocks until its single target process has synchronized. In such a case, the call violates the standard. The benefit of this approach (presented in [7]) is that a process waits for process synchronization when it is actually required. In addition, after the initial synchronization with a particular target is completed, all subsequent communication with that process can be performed promptly.

IV. MPI PROCESS SYNCHRONIZATION ON THE SCC

No current MPI implementation efficiently supports PSCW synchronization on an nCC many-core chip. This section presents the design of such a scheme for the Intel SCC.

A. The Intel SCC

The Intel SCC [3] is an experimental tiled many-core chip with 48 Pentium (P54C) cores. Each tile contains two

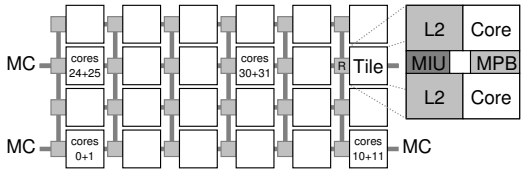


Fig. 2. Overview of the Intel SCC

cores. A router connects each of the 24 tiles to an on-chip network as illustrated in Figure 2. Also, four DDR3 memory controllers are attached to the network. The mesh interface (MIU) unit on each tile translates memory accesses by the cores to network packets and vice-versa. The unit performs a configurable address translation that determines the actual destination of a memory access. If more than one MIUs are configured to translate to the same network address shared memory is created. Further, the MIU contains one atomic test-and-set register per core that has mutex-like semantics.

Although each core has two cache levels, the hardware provides no support for cache coherence at all. This complicates the usage of shared memory in the common sense. In consequence, message passing is the most adequate programming model for this on-chip cluster. To support this, each tile possesses a 16 KB SRAM-based so-called *Message Passing Buffer (MPB)* that enables fast on-chip data transfer without using the external DDR3 memory.

In the default configuration, the MIU provides access to 16 MB per memory controller that is shared between all cores and is called *legacy shared memory*. However, accesses to this memory have to be performed carefully due to the missing cache coherence.

In addition, different memory types are supported. Most relevant for this work is the non-cacheable memory (NCM) type which bypasses all cache levels. It has been shown that usage of this memory can provide better latencies over other (cached) memory types [8].

On top of SCC's hardware, the MPI implementation RCKMPI [9] was developed. It is based on MPICH [10], thus inherently message-based, and exploits the MPB. Due to the derivation from MPICH, the implementation of one-sided communication, including the process synchronization, is also message-based [11].

B. SCOSCo process synchronization

The design of the PSCW synchronization is developed in the context of the implementation of SCOSCo [2], a software-managed cache coherence protocol for one-sided communication.

To efficiently implement the PSCW synchronization pattern on the SCC, the message based approach of RCKMPI is dropped. Instead, the presented solution is based on an optimization of MVAPICH for shared memory systems [7] which belongs to the trigger-only class. It is demonstrated in the following that this approach can be successfully implemented on the non-cache-coherent Intel SCC.

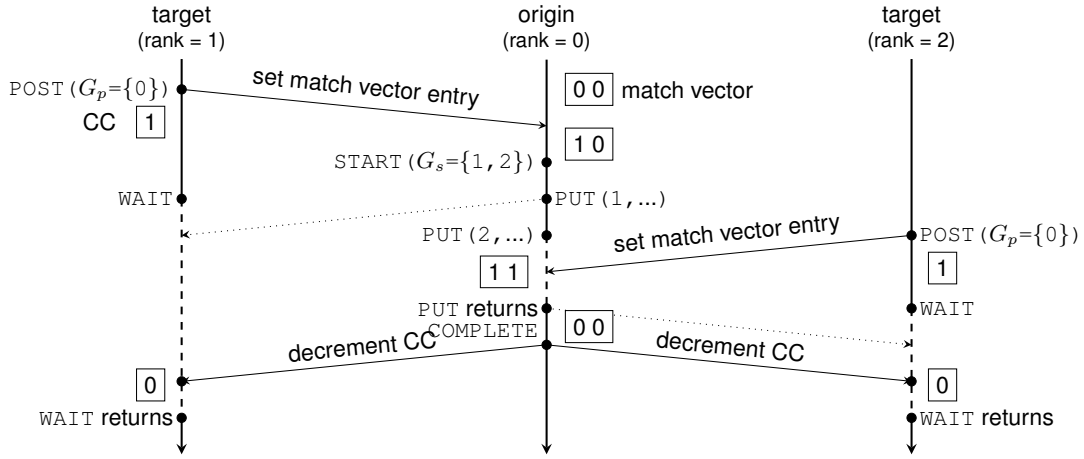


Fig. 3. Sequence diagram of the implemented synchronization protocol for two target processes (left and right) and a single origin (center).

The SCC's ability to define shared memory is exploited to store and access synchronization data. For this purpose, a memory region called *window database* is reserved inside the legacy shared memory (see above). Therein, the required synchronization data structures are allocated. A bit vector for the start of an epoch (called *match vector*) and completion counters are used as space-efficient means for synchronization. For polling these data structures efficiently, uncached memory is used to circumvent coherence problems.

1) Window Creation: The synchronization data structures are allocated in the legacy shared memory which is accessible by all processes in the system. The allocation is distributed among the four memory controllers to avoid contention of the controllers when the match vector and completion counters are polled. This issue was observed in preliminary experiments especially when synchronization was executed in memory-bound applications. Therefore, depending on the core a MPI process is running on, it allocates the synchronization data inside the legacy shared memory such that the nearest (Manhattan distance) memory controller is used for accesses.

The actual allocation is performed during the collective window creation. Since it is unknown to a process whether it becomes target or origin for the created window, space for both match vector (for origins) and completion counter (for targets) is reserved. The size of the bit vector is equal to the window's communicator size. Memory is allocated atomically (with the help of the test-and-set registers) by reading and advancing an offset, which is stored in the windows database, by the amount of allocated memory.

The obtained offset of the reserved memory is exchanged by an MPI all-to-all operation between all the n processes which create the window. Consequently, each process knows the base location of all other processes' synchronization data. Since the match vector size is known locally (and is equal among all processes), the position of the completion counter can be derived locally as well. This approach leads to data duplication of the exchanged offsets which scales with n^2. However, it can be compensated by storing all offsets in the shared memory as well. Anyhow, this is left out of the prototypical implementation.

2) START and POST: When a `POST` operation is issued by a target process, it iterates through the post group G_p that contains all origins (see Section II). To notify each of them, the address of the according match vector in the legacy shared memory is determined. Then, the test-and-set register of the origin's core is locked to prevent concurrent modifications (see Algorithm 1) Subsequently, the byte containing the according bit is read, locally modified and written back using uncached memory. Cached memory is unsuitable for this use-case since it would require an explicit write-back of the according cache line. Such an operation is not supported by the SCC's cores. Only a write-back including an invalidation of the whole cache is available in the instruction set. However, it is obviously far slower than using uncached memory. In addition, the match vector is not accessed by the targets for any other purpose which makes caching unnecessary anyway.

Algorithm 1 Pseudocode for START and POST

```
function START(G_s: Group)
    start_ranks ← ranks of procs ∈ G_s
end function

function POST(G_p: Group)
    CC ← |G_p|                          ▷ init completion counter
    for all origins ∈ G_p do            ▷ notify all origins
        core ← CORE_OF_PROC(origin)
        LOCK_TSR(core)
        match_vector[core][local_rank] ← 1
        UNLOCK_TSR(core)
    end for
end function
```

On the origin side, the `START` function performs only bookkeeping operations, like storing the ranks of the processes

from the start group G_s (see Section II). Polling the match vector is shifted to the communication calls, like `PUT` and `GET`. However, these calls only check for the according target process to have synchronized. Thus, the implementation is classified as trigger-only (see Section III).

Polling is performed with uncached memory since cached reads would prevent the origin to observe a post operation. In addition, caching of the match vector is dangerous in any case. Suppose a match vector is stored in the origin's cache and that it can successfully communicate with all targets of the current access epoch. Independent from this, another target of a subsequent access epoch (but of the same window) performs its `POST` call and modifies the match vector in main memory. Due to the missing coherence, the cached copy is unchanged, but now becomes invalid. In case the origin's cache evicts the line (due to memory accesses by computation, e.g.) that contains the cached and outdated match vector, it overrides the manipulated match vector in the main memory. This would cause a deadlock as the origin would check for a post operation that actually took place but its effect was destroyed by the origin itself and leads to an infinite loop.

However, to speed up polls of already synchronized targets, a local and cacheable copy of the match vector is employed. Inside a communication call the cached vector copy is checked first. If there was no `POST` operation, the uncached match vector is polled and upon detection of the target's post operation, the cached copy is updated.

3) COMPLETE and WAIT: At the end of its access epoch, i.e. in `COMPLETE`, the origin resets the entries corresponding to the targets in G_s in the match vector and in its copy. Subsequently, it decrements the targets' completion counters. However, the notification of completing an access epoch can only be performed after an origin has successfully started its matching exposure epoch. Only then, the target's completion counter is in a valid state and can be modified when the origin completes. Thus, an origin first ensures that every targets in G_s has synchronized (see Algorithm 2).

Then, it iterates through all targets and decrements their completion counter. Similar to the targets' accesses on the match vector, the completion counter is accessed with uncached memory (see Figure 3). Further, to make the decrement atomic, the test-and-set registers of the target's core are used.

On the target's side, the `WAIT` call polls the completion counter with uncached memory as well to observe the decrements made by the origins. Polling is performed until the counter reaches zero.

Algorithm 2 Pseudocode for COMPLETE and WAIT

```
function COMPLETE
    for all targets ∈ start_ranks do                ▷ see START
        repeat                                      ▷ busy wait for all targets
        until match_vector[local_core][target] == 1
        match_vector[local_core][target] ← 0
    end for

    for all targets ∈ start_ranks do                ▷ notify all targets
        core ← CORE_OF_PROC(target)
        LOCK_TSR(core)
        CC[core] ← CC[core] − 1                     ▷ decrement remote CC
        UNLOCK_TSR(core)
    end for
end function

function WAIT
    repeat                                          ▷ busy waiting
    until CC == 0                                   ▷ poll with uncached reads
end function
```

V. Experimental Evaluation

To evaluate the performance of the SCOSCo approach, the runtime required for synchronization is analyzed. The experiments were conducted on a SCC system with cores clocked at 533 MHz and 800 MHz for the mesh network and the memory controllers. A total of 32 GB of RAM was installed on the system. Each core runs Linux 3.1.4 with platform-relevant patches applied. Software is cross-compiled using GCC 4.4.6, and MPICH 3.1.3 was used as the foundation MPI implementation. The MPB-based CH3 channel from RCKMPI was merged together with the modifications from [11] into the MPICH sources. The synchronization functions were overridden to implement the approach presented in this work. The resulting MPI library was compiled with optimization enabled (`-O2`).

A. Microbenchmark

For measuring the synchronization performance, a microbenchmark was created that does not include any one-sided communication like PUT and GET. This enables a fair comparison with other implementations, for example MPICH's deferred approach that performs queued communication in the `COMPLETE` routine.

Algorithm 3 Microbenchmark Pseudocode

```
for i = 0 ... 1000 do
    if rank == 0 then
        t_{s,i} ← TIME(START(G_s = {1 ... k}))
        t_{c,i} ← TIME(COMPLETE)
    else
        t_{p,i} ← TIME(POST(G_p = {0}))
        t_{w,i} ← TIME(WAIT)
    end if
end for
```

Consequently, the microbenchmark only uses the PSCW methods (cf. Figure 1). Further, it consists of a single origin process that synchronizes with a given number of k targets as illustrated in Algorithm 3. The runtime of each of the four PSCW routines is measured by reading the per CPU timestamp counter with the `RDTSC` instruction before and after the call. The synchronization sequence is repeated 1001 times. In case of the targets, all recorded times $t_{p,i}$ and $t_{w,i}$ from all k

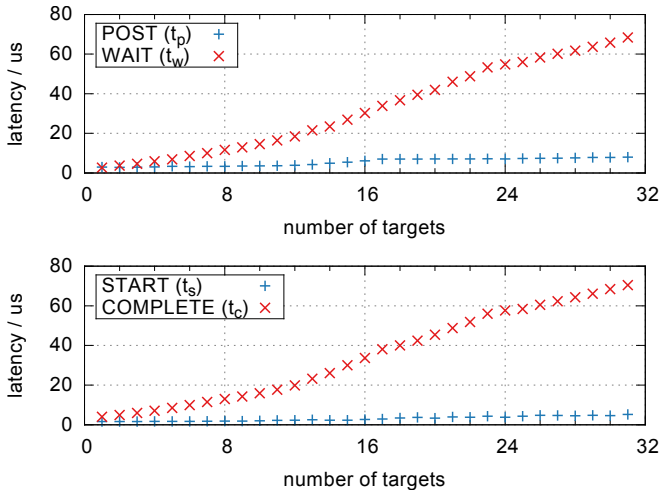


Fig. 4. Scaling of PSCW synchronization on the SCC for the target (top) and origin (bottom) processes.

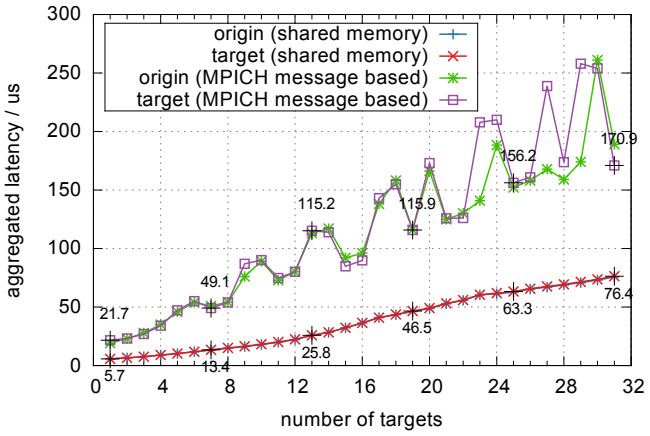


Fig. 5. Comparison of target and origin synchronization times t_o and t_t. Numbered labels inside the plot are given for the target processes.

targets are collected and the medians t_p and t_w are obtained from the $k * 1001$ samples. Besides outliers caused by process scheduling only little deviation from the reported values was observed: For all measurements, the first and third quartile of the four measured timings never deviated by more than 5% for origin and 10% for targets from the according median.

A synchronization at the beginning of each synchronization loop is omitted since after one iteration the processes are synchronized anyway. Additionally, using an MPI barrier caused fluctuations in the measurements because the origin process might leave the barrier later than some targets depending on the number of started processes. In the experiments, the benchmark was compiled with optimization enabled (`-O2`).

B. Scaling

The microbenchmark was used to analyze the scaling of the implemented synchronization scheme. It was executed for up to 32 processes (1 origin, up to $k = 31$ targets). The SCC provides more cores but using (nearly) all of them questions the usage of the PSCW scheme. In such cases, fence is more appropriate.

The results for target and origins are shown in Figure 4. The cores with numbers up to 31 (see Figure 2) were used for this analysis.

The presented results show a nearly constant runtime for the `START` and `POST` operations. For `POST`, this can be accounted to the usage of only one origin process in the experiment. In case of `START`, the constant runtime is attributed to the trigger-only design of the synchronization protocol which does not involve any communication in that routine. Thus, the observed latency is introduced by MPI library overhead only.

Contrary, the `COMPLETE` and `WAIT` runtime exhibit linear scaling. Concerning the `COMPLETE` call, this has two reasons. First, the origin needs to notify all targets which is done in a loop and thus causes linear scaling behavior. To do so, it has to wait for all target to be synchronized (cf. Section IV-B) which also scales linearly. This consequently affects the `WAIT` on the target side which therefore shows linear scaling as well.

C. Comparison to MPICH's Message based approach

Next, the SCOSCo synchronization was compared against the default RCKMPI implementation that uses MPICH's message based CH3 synchronization [11] but transfers messages over the on-chip MPB (see Section IV-A). The microbenchmark and the methodology from Section V-B were reused. However, the recorded median times of the origin, i.e. t_s and t_c, were summed, giving the total time t_o required to perform the PSCW synchronization on the origin side. The same was done for the recorded median of the target times (t_p and t_w) leading to t_t. Figure 5 shows the obtained t_o and t_s for both RCKMPI/MPICH and SCOSCo as well as for different number of targets.

The results reveal that despite RCKMPI's usage of the fast on-chip message passing buffer [11], the performance of the message-based synchronization from MPICH delivers latencies more than four times higher (e.g., 13 targets: 25.8 µs vs. 115.2 µs) than the SCC-aware approach. This can be explained by the overhead of message assembly, sending, reception and processing by MPICH's internal progress engine. This underlines that even in presence of a tuned implementation for message transfer, a shared memory approach is more appropriate on the SCC. Bearing in mind, this is possbile without hardware support for cache coherence.

VI. Summary and Conclusion

We presented the design and implementation of the SCOSCo PSCW synchronization on the Intel SCC, a non-cache-coherent many-core system. The developed approach leverages match vector and completion counters located in shared memory. Even without cache coherence, the approach delivers linear scaling and outperforms optimized message based synchronization that utilizes fast on-chip memory. This

shows that efficient synchronization is feasible also on nCC architectures.

REFERENCES

[1] T. P. Morgan, "More Knights Landing Xeon Phi Secrets Unveiled," Mar. 2015, accessed 2015-11-02. [Online]. Available: http://www.nextplatform.com/2015/03/25/more-knights-landing-xeon-phi-secrets-unveiled/

[2] S. Christgau and B. Schnor, "Software-managed cache coherence for fast one-sided communication," in *Proceedings of the Seventh International Workshop on Programming Models and Applications for Multicores and Manycores, Barcelona, Spain*, P. Balaji and K.-C. Leung, Eds., 2016, to appear.

[3] J. Howard *et al.*, "A 48-core IA-32 message-passing processor with DVFS in 45nm CMOS," in *Solid-State Circuits Conference Digest of Technical Papers (ISSCC), 2010 IEEE International*, Feb. 2010, pp. 108 –109.

[4] Message Passing Interface Forum, "MPI: A Message-Passing Interface Standard, Version 3.1," online, Jun. 2015, http://www.mpi-forum.org/docs/mpi-3.1/mpi31-report.pdf.

[5] P. Reble, C. Clauss, and S. Lankes, "One-sided communication and synchronization for non-coherent memory-coupled cores," in *International Conference on High Performance Computing & Simulation, HPCS 2013*. IEEE, 2013, pp. 390–397.

[6] W. D. Gropp and R. Thakur, "An evaluation of implementation options for MPI one-sided communication," in *Recent Advances in Parallel Virtual Machine and Message Passing Interface, 12th European PVM/MPI Users' Group Meeting, Proceedings*, ser. LNCS, vol. 3666. Springer, 2005, pp. 415–424.

[7] P. Lai, S. Sur, and D. K. Panda, "Designing truly one-sided MPI-2 RMA intra-node communication on multi-core systems," *Computer Science - R&D*, vol. 25, no. 1-2, pp. 3–14, 2010.

[8] R. Rotta, "On efficient message passing on the Intel SCC," in *Proceedings of the 3rd MARC Symposium, Ettlingen, Germany, July 5-6, 2011*. KIT Scientific Publishing, Karlsruhe, 2011, pp. 53–58.

[9] I. A. C. Ureña, M. Riepen, and M. Konow, "RCKMPI - lightweight MPI implementation for Intel's single-chip cloud computer (SCC)," in *Recent Advances in the Message Passing Interface - 18th European MPI Users' Group Meeting, EuroMPI 2011. Proceedings*, ser. LNCS, vol. 6960. Springer, 2011, pp. 208–217.

[10] W. Gropp, "MPICH2: A new start for MPI implementations," in *Recent Advances in Parallel Virtual Machine and Message Passing Interface, 9th European PVM/MPI Users' Group Meeting, Proceedings*, ser. LNCS, vol. 2474. Springer, 2002, p. 7.

[11] S. Christgau and B. Schnor, "One-sided communication in RCKMPI for the Single-Chip Cloud Computer," in *Proceedings of the 6th MARC Symposium, 19-20 July 2012, Toulouse, France*. ONERA, The French Aerospace Lab, 2012, pp. 19–23.

Simulation based Analysis of Memory Access Conflicts for Heterogeneous Multi-Core Platforms

Jens Brandenburg and Benno Stabernack
Fraunhofer Institute for Telecommunications, Heinrich Hertz Institute
Video Coding & Analytics Department, Embedded Systems Group
Einsteinufer 37, 10587 Berlin, Germany
{jens.brandenburg,benno.stabernack}@hhi.fraunhofer.de

***Abstract*—Besides aspects of HW/SW partitioning, resource allocation and mapping, also the optimization of the memory subsystem plays a crucial role during the complex HW/SW co-design and co-optimization process. Especially for memory bound applications, like state of the art video codecs, the memory subsystem has become one of the bottlenecks limiting the performance gains from parallelization and HW accelerated approaches. Memory access conflicts, due to the concurrent access to a shared memory location, are a major source of this bottleneck. To develop counter strategies and to optimize the design, an in-depth analysis of all memory access conflicts is necessary and required. In order to provide this analysis, we propose a flexible tracing and profiling methodology, which provides a timing-accurate memory access conflict analysis for SystemC-based platform simulation models. In a case study this memory access conflict analysis is performed for a heterogeneous platform running a parallel high efficiency video coding (HEVC) intra encoder application. This analysis leads to an optimized design, which reduces the number of memory access conflicts and shows significant performance gains for the target video encoder application.**

I. Introduction

Nowadays the trend in video coding applications goes towards more complex algorithms and more data to process. For example, the high efficiency video coding (HEVC) standard supports at its highest level a video resolution of 8K (8192×4320) with 120 frames per second, which results in a raw data input rate of approx. 6TB [1] per second. A typical approach to speed up such an application is the parallelization of the algorithm and/or the acceleration of the most performance requiring parts of the algorithm by dedicated HW components. According to Amdahl's law, the theoretical improvement of such an approach can be calculated based on the relationship between improved and not improved portions of the algorithm but unfortunately most systems did not show the expected behaviour. One aspect limiting the expected parallelization improvements can be found in the shared memory, where the growing number of memory access conflicts increases the average memory access latency for all components. To avoid or reduce these memory access conflicts, an analysis of the developed system and its memory behaviour is an important requirement.

In order to aid the developer with this optimization task, we present a performance and memory access analysis tool based on the tracing and profiling of SystemC-based simulation models. This tool can be used in the early design phase of the HW/SW partitioning, when only a platform simulation model is available, as well as in later design phases, like the HW/SW co-optimization and refinement. The tool collects different low level trace data events and combines these data to analyse the memory access characteristics, in particular the memory access conflicts, of an observed platform simulation model. A demonstration of the tool's capabilities is given for a video coding use case where the analysis of the memory access conflicts leads to an optimized design, which improves the overall application performance.

[1] Assuming a 4:2:0 color coding scheme with a bit-depth of 8Bits per pixel.

II. Related Work

Memory access analysis approaches can be classified into intrusive and non-intrusive approaches. Intrusive approaches collect the trace data by executing a modified version of the analyzed software. This modification changes the execution of the software, which affects the timing accuracy of the collected trace data. Contrastingly, non-intrusive approaches collect the trace data without modifying the software, typically by the usage of modified hardware components. For this use, non-intrusive approaches mostly utilize simulation techniques, which leads to high execution times, especially in comparison to intrusive approaches, where the instrumented software often can be executed directly on the host processor.

However, also intrusive approaches may use simulation techniques to model some part of the target architecture as proposed for example in [1]. In this approach a cache simulator is added to the Valgrind framework [2] to model and analyze cache related metrics, e.g. number of cache accesses and cache misses for a generic cache implementation. In this context the Valgrind framework itself provides the infrastructure for the instrumentation of the observed software, based on a dynamic binary analysis (DBA) at runtime. A similar trace data collection approach is presented in [3], which implements a Valgrind based profiling tool to recognize memory access patterns in computational kernels. Therefore all memory accesses are traced, and then the tool checks if the access pattern of multiple accesses can be matched to a set of classifiers with predefined memory access characteristics.

Hardware component instrumentation based approaches can be found in [4], [5] and [6]. All three approaches are using

the SoCLib [7] system simulation component library to collect the trace data, either by adding special hardware monitoring components [6] or by modifying existing hardware components [4] and [5]. Hereby, the analysis in [6] is focussed on the latency of communication channels and their basic data structures. In comparison, the analysis in [4] is focussed on tracing memory accesses to detect and identify potential data races in a parallel algorithm. An extension of this data race detection tool is proposed in [5]. This tool additionally traces the memory access latencies to identify portions in the source code where a certain contention pattern arises.

A more generic instrumentation approach for SystemC simulation models is presented in [8]. This approach is based on the aspect oriented programming paradigm to separate the implementation of the trace data code from the functional code of the platform simulation model. In a process called aspect weaving the trace data code is inserted by a pre-processor into the platform simulation model to trace different events from the implementation of the hardware components. By using this instrumentation approach, [9] automatically traces all transaction level modeling (TLM) connections between different SystemC modules, to verify different pre-defined timing conditions for TLM based data exchanges.

In this paper we present an extension of our non-intrusive tracing and profiling tool [10]. This extension enables the collection of memory access characteristics and provides a memory access conflict analysis. Due to the simulation based approach, the tool can perform a timing-accurate analysis of all memory accesses. Moreover, the tool is capable of analyzing heterogeneous platforms and can also incorporate memory access data from HW only components into the conflict analysis. In comparison to the other discussed SystemC-based approaches, the tool does not require the instrumentation of the platform simulation model to collect the necessary trace data. Instead, an automatic platform design analysis is performed to configure the trace data collection module and to provide an easy adaption mechanism for new platform simulation models.

The remainder of this paper is organized as follows. In Section III the simulation based memory access conflict analysis methodology is described. Based on this analysis methodology, a case study for a heterogeneous and parallel HEVC intra video encoder is shown in Section IV. Results for the suggested optimization of the memory layout are discussed in Section V and finally a concluding summary is provided in Section VI.

III. Memory Access Conflict Analysis

A. Non-intrusive Trace Data Collection

Given a software based executable of the platform simulation model, a simple way to collect the trace data is to instrument this platform simulation model with a set of trace primitives, which have to be included at appropriate places. A downside of this approach is the lack of flexibility, because the instrumentation code has to be maintained, whenever the implementation is changed. Furthermore introducing new components or switching to a new platform simulation model requires from the developer the additional effort to add this instrumentation code. In case of SystemC, each platform simulation model is composed of basic standard components, e.g. SystemC modules, ports and signals, which enables a more flexible way to collect the required trace data by instrumenting these standard components directly.

In SystemC different platform components are either connected by a set of signals, in case of bit accurate designs, or by a TLM socket, in case of a transaction level model (TLM) design. Based on this observation it is sufficient to instrument the SystemC signal class and the TLM socket class to trace a memory access. For this instrumentation we choose the signal update function in the common SystemC signal base class and the transaction start/end functions in the common TLM socket base class. By adding callbacks from these functions to our trace data collection module, all signal changes and all TLM access within each SystemC-based simulation model can be traced.

Of course for complex designs with hundreds or more signals, it is not feasible and also not required to trace all changes. To reduce the set of traced events, a set of relevant events within the platform simulation model has to be identified. This identification is based on an automatic design analysis, which is performed at runtime, after the platform simulation model has been created. After the SystemC elaboration phase, all instantiated modules and signals are inspected by the trace data collection module and a map of design elements and their interconnections is created. Additionally the trace data collection module analyses static C++ design elements, to enable also the tracing of changes of SystemC module member variables. This information is relevant to observe also static design elements, like a program counter or a register set in an instruction set simulator, which is required to correlate software and hardware events in the observed system. Based on this automatically generated map of SystemC modules and their interconnections only the tracing of memory access signals is enabled inside the SystemC simulation kernel, without requiring to instrument or modify the used platform simulation model.

B. Memory Access Conflict Detection

To create memory access traces, it is necessary to collect information about the accessed memory address, the accessed size and the duration of this access. But in case the simulation model utilizes signals to carry out a memory access, this information has to be extracted from a set of multiple signals, each playing a specific role during the memory access process. Based on a meta-data specification of the signal roles for different data exchange protocols, the traced signal changes are combined to provide the required memory access information. An exemplary signal role configuration for the virtual component interface (VCI) standard is shown in Fig. 1.

Given the combined memory access traces from multiple platform components, e.g. processing elements and/or HW accelerators, the detection of memory access conflicts can be realized. In this case a conflict is detected, whenever two different memory accesses from different sources temporally

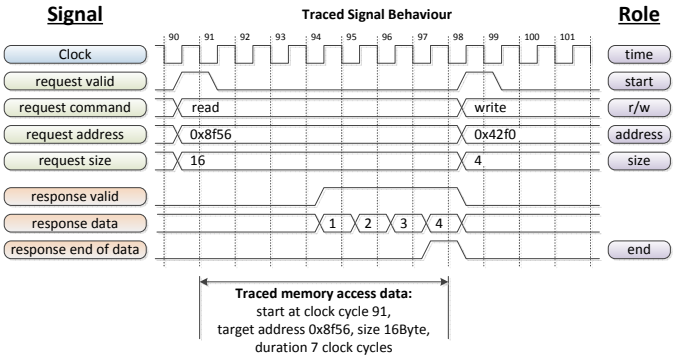


Fig. 1. Combining the traced signal changes to collect different memory access information.

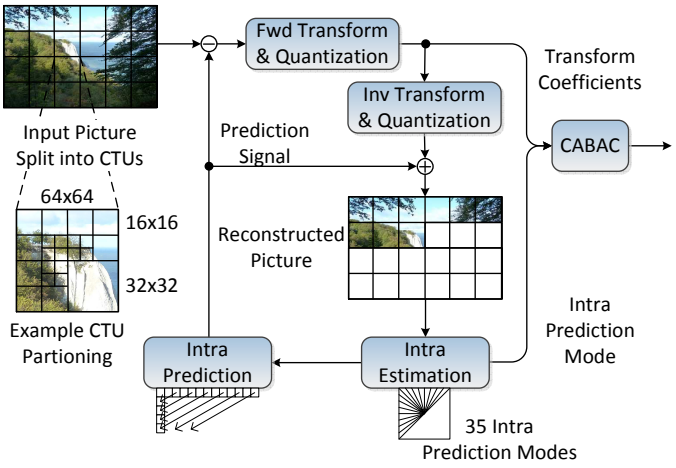


Fig. 2. HEVC intra encoder model.

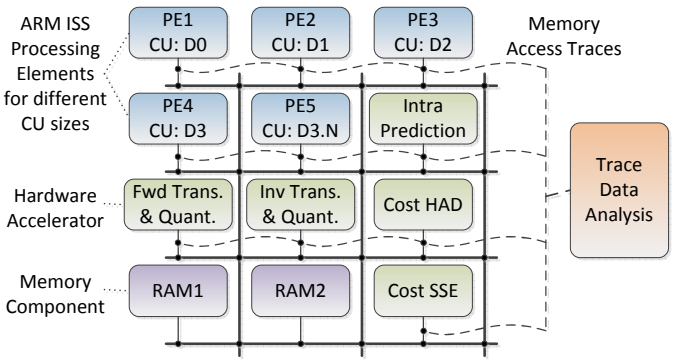


Fig. 3. Instrumented platform simulation model.

overlap and the execution of at least one of these accesses is delayed. The implemented conflict detection process is based on a windowed approach, where each memory access is stored in a FIFO buffer and removed, when the end time of the access is below the start time of a newly detected access. Therefore only the small set of memory accesses in the FIFO has to be compared with each other to detect all memory access conflicts during a simulation model run. For the evaluation of these conflicts each conflict is either attributed to different data structures by its traced memory addresses or to different executed program instructions by tracing the current program counter.

IV. Case Study

A. High Efficiency Video Coding

The HEVC standard is an example of a block-based hybrid video coding scheme. In this scheme the input frame is split into a set of smaller blocks called coding units (CUs), and each CU is encoded by a combination of prediction, transform and entropy coding algorithms, as shown in Fig. 2. In case of the HEVC main intra profile the prediction uses only reconstructed samples from the current frame facilitating the spatial correlation between adjacent samples.

For a high video compression efficiency it is beneficial to adapt the CU size (block size) to the input video data content. Therefore the encoder supports different CU sizes ranging from 64×64 to 8×8 samples. To adapt the CU size to the input video data content, the encoder can decide to use either the whole CU for the compression algorithm or to split the CU recursively into four smaller CUs, which leads to a tree like coding structure called coding tree unit (CTU).

Because these different CU size compression stages could be executed independently from each other, the encoder algorithm has been parallelized internally by using one task for each CU size compression stage. This results in five parallel CU compression kernels, which will be denoted by their depth in the CTU tree by D0 for size 64×64, D1 for size 32×32, D2 for size 16×16, D3 for size 8×8 and D3.N for size 8×8 with an additional split of the prediction unit. Additionally different time consuming parts of the algorithm, like transform, quantization, intra prediction or cost calculation, have been implemented as HW-accelerated components, to further speedup the platform simulation model. Each of these HW accelerator components has a set of control registers for programming and a DMA controller for data exchange. A more detailed description of the used HEVC intra encoder is given in [11].

The implemented platform simulation model for this use case analysis contains five ARM instruction set simulators (ISS) for each of the parallel CU size compression stages, five HW accelerator components, a worm-hole switched network on chip (NoC) and two memory components, as shown in Fig. 3. Most commonly used synchronization data structures are located in the second memory component *RAM2*, as described in [11]. The components for this platform simulation model are cycle accurate and bit accurate (CABA) components provided by the SoCLib project [7]. In order to detect memory access conflicts, all outgoing data connections from processing elements and HW accelerators will be analyzed and evaluated, as shown in Fig. 3.

B. HEVC Encoder Analysis

To visualize the distribution of the memory access conflicts, the collected data is sampled at different time intervals and for different memory regions forming a spatial temporal grid. In

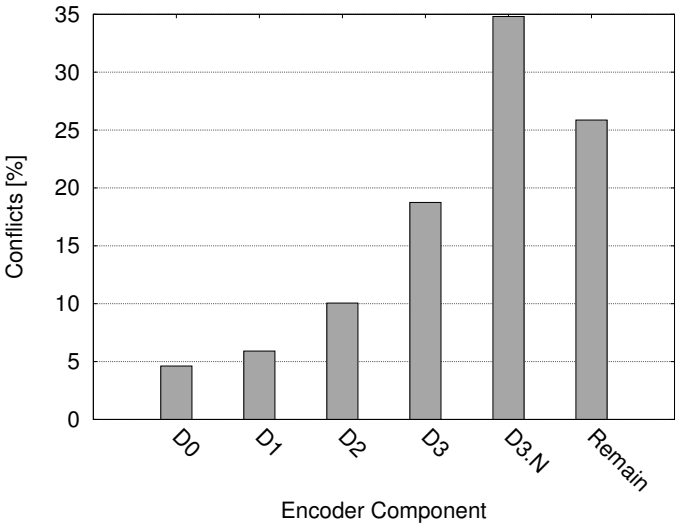


Fig. 5. Distribution of memory access conflicts per parallel CU compression stage.

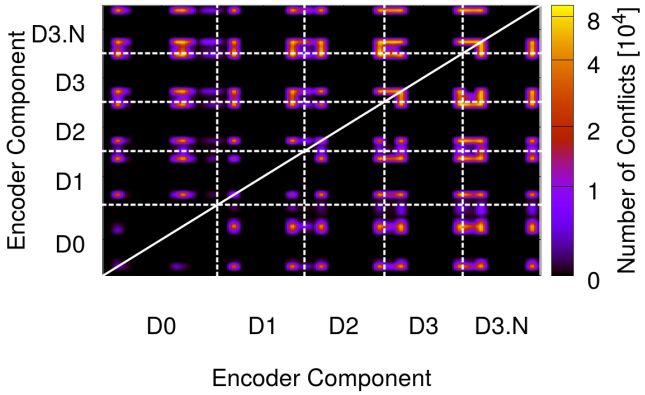


Fig. 6. Memory access conflict matrix for CU compression stages.

Fig. 4 this sampled data is shown for an exemplary encoder run, where the first three CTUs have been encoded. As can be seen in this figure, most conflicts are found in the memory regions marked by the horizontal lines, which contain the data structures for each CU compression stage. Furthermore no conflicts can be found in the initialization phase, because in this phase only one processing element is allocating and initializing all data structures. Afterwards, the memory access conflict pattern is quite similar for each encoded CTU, because the parallel algorithm has frequent synchronization points inbetween, which results in a similar conflict behaviour.

A summary of the memory access conflicts distribution in percent for data structures used only by the different CU compression stages is given in Fig. 5. As can be seen, in summary most of the memory access conflicts (approx. 74%) can be contributed to these data structures, with an increase of conflicts towards the lower depth CU compression stages. The remaining part *Remain* contains conflicts in data structures shared between the CU compression stages, like video data input, bit-stream data output or the reconstructed picture buffers.

To visualize the conflicts between the CU compression stages, Fig. 6 shows a memory access conflict matrix, where a conflict is registered for each pair of involved memory addresses. As can be seen, each stage interferes with each other stage in a similar way, but most conflicts are located at the beginning and at the end of each CU compression stage data structure.

In summary most conflicts can be located in the data structures of each CU compression stage with an increase towards the lower CU compression stages. These memory structures are quite small (approx. 512KB) in comparison to the overall memory footprint of the encoder. Therefore a simple counter measure to reduce the number of memory access conflicts is, to add further memory components to the platform and to map the CU compression data structures to each of these new separate memory components. Due to the NoC interconnect, multiple concurrent accesses become possible as long as these accesses are routed to different memory components.

V. Experimental Results

In a first experiment each of the five different CU compression data structures is mapped to a separate memory component added to the platform simulation model. For each memory component the access latency is configured in a range from 5 to 50 clock cycles to model different main memory access delays. As can be seen in Fig. 7, the speedup between the basic platform simulation model and the model with the additional memory components increases for higher memory access delays. In case the memory access delay is small, which in terms means a fast main memory, the improvement of the execution time is small, when adding additional memory blocks. On the other side, systems with slow main memory will benefit from adding additional memory blocks, which means for a final design a trade-off is possible between the memory access delay and the number of memory components.

As shown in the previous section Sec. IV-B, the number of memory access conflicts increases for lower CU compression stages. Therefore in a second experiment only a subset of the CU compression data structures is mapped to the additional memory blocks, whereas the memory access latency is fixed at 20 clock cycles. In one configuration, shown on the left side of Fig. 8, only one single CU compression component is mapped to a new memory component. Additionally, in another configuration, shown on the right side of Fig. 8, the number of mapped CU compression data structure is successively increased, starting from the lowest CU compression stage, until all data structures have been mapped to different memory blocks.

As expected, mapping the D3.N data structures to a separate memory component, shows the highest speedup in case only one separate memory component is available. In case multiple

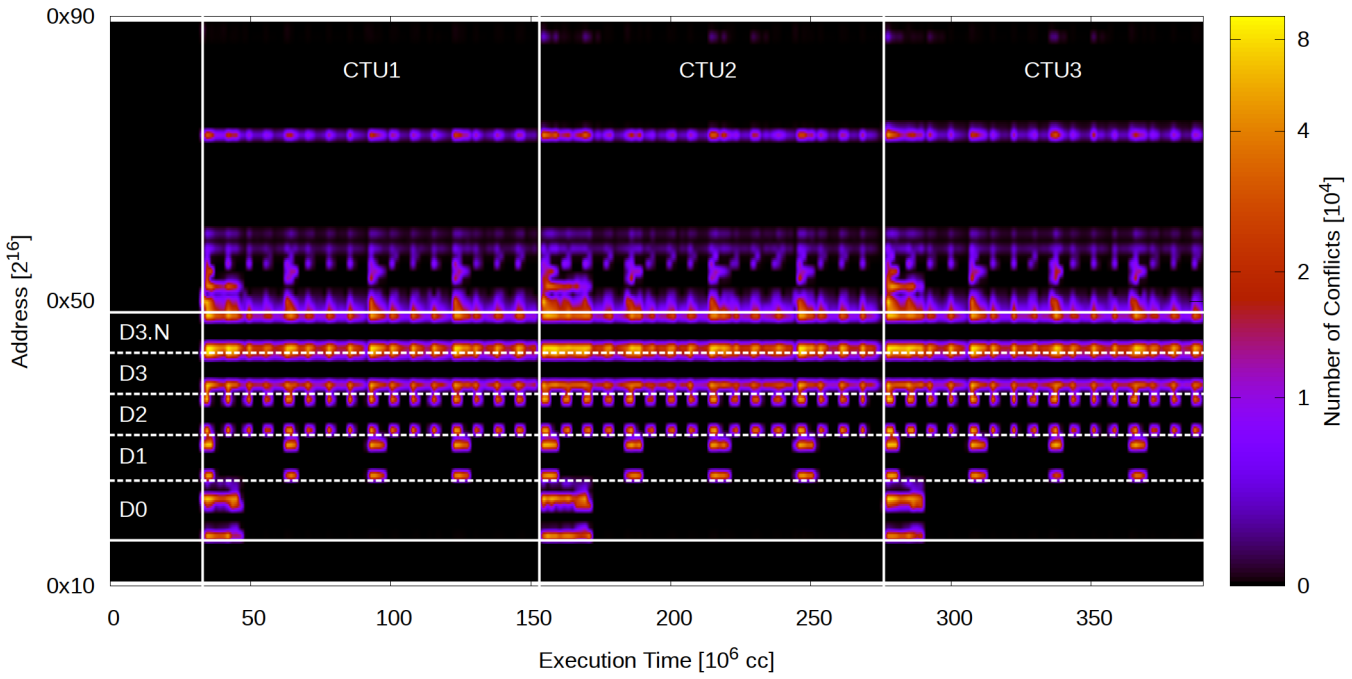


Fig. 4. Memory access conflict trace over execution time for encoding three CTUs. Vertical lines depict the beginning of a new encoded CTU. Horizontal lines mark the memory interval containing CU compression data for each stage.

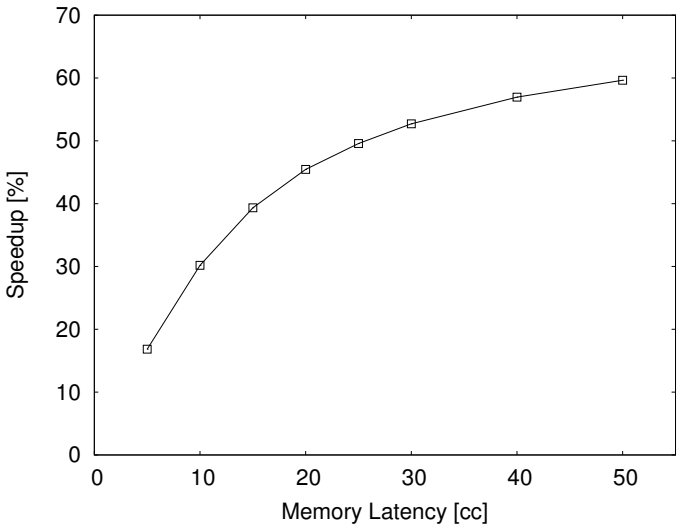


Fig. 7. Encoder execution time speedup of the optimized platform simulation model for different memory access latency parameters.

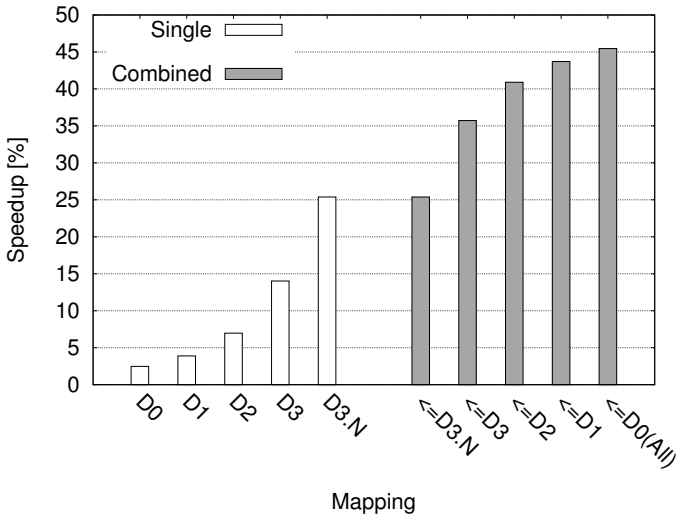


Fig. 8. Encoder execution time speedup of the optimized platform simulation model for different memory mappings.

separate memory blocks are available, the distribution of memory access conflicts can be used to prioritise the mapping of the different data structures and to explore a trade-off between performance and HW costs for additional memory components.

VI. Conclusion

This paper presents a simulation based analysis of memory access conflicts for a heterogeneous multi-core platform. The memory access trace data is collected non-intrusively from a SystemC-based simulation model. Therefore the SystemC simulation kernel has been instrumented, which facilitates a flexible and generic trace data collection approach. In order to control the trace data collection module, an automatic design analysis of the SystemC-based simulation model is performed at runtime. The analysis of memory access conflicts is based on the collection of all memory access data from each platform component, e.g. processing elements and HW accelerators.

By this approach all sources for memory access conflicts are observed and also heterogeneous designs can be analyzed.

To show the capabilities of the implemented memory access conflict analysis, a heterogeneous and parallel HEVC intra video encoder has been examined in a case study. In this case study the main sources for memory access conflicts have been identified, which leads to an optimized platform design. The speedup of this optimized design depends on the initial memory access latency but even for small latencies (10cc) a performance gain of 30% and more could be observed. In future work we plan to extend the analysis tool to perform an automatic hotspot analysis and to provide automatically a segmentation of the involved data structures to derive possible platform optimizations.

REFERENCES

[1] J. Weidendorfer, M. Kowarschik, and C. Trinitis, "A tool suite for simulation based analysis of memory access behavior," in *Computational Science - ICCS 2004*, ser. Lecture Notes in Computer Science, M. Bubak, G. van Albada, P. Sloot, and J. Dongarra, Eds. Springer Berlin Heidelberg, 2004, vol. 3038, pp. 440–447. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-24688-6_58

[2] N. Nethercote and J. Seward, "Valgrind: A framework for heavyweight dynamic binary instrumentation," *SIGPLAN Not.*, vol. 42, no. 6, pp. 89–100, Jun. 2007. [Online]. Available: http://doi.acm.org/10.1145/1273442.1250746

[3] E. Park, C. Kartsaklis, T. Janjusic, and J. Cavazos, "Trace-driven memory access pattern recognition in computational kernels," in *Proceedings of the Second Workshop on Optimizing Stencil Computations*, ser. WOSC '14. New York, NY, USA: ACM, 2014, pp. 25–32. [Online]. Available: http://doi.acm.org/10.1145/2686745.2686748

[4] D. Hedde and F. Petrot, "A non intrusive simulation-based trace system to analyse multiprocessor systems-on-chip software," in *Proc. 22nd IEEE Int Rapid System Prototyping (RSP) Symp*, 2011, pp. 106–112.

[5] S. Lagraa, A. Termier, and F. Pétrot, "Data mining MPSoC simulation traces to identify concurrent memory access patterns," in *Proceedings of the Conference on Design, Automation and Test in Europe*, ser. DATE '13. San Jose, CA, USA: EDA Consortium, 2013, pp. 755–760. [Online]. Available: http://dl.acm.org/citation.cfm?id=2485288.2485471

[6] D. Genius, "Measuring memory access latency for software objects in a NUMA system-on-chip architecture," in *Reconfigurable and Communication-Centric Systems-on-Chip (ReCoSoC), 2013 8th International Workshop on*, Jul. 2013, pp. 1–8.

[7] N. Pouillon, A. Becoulet, A. de Mello, F. Pecheux, and A. Greiner, "A generic instruction set simulator API for timed and untimed simulation and debug of MP2-SoCs," in *Rapid System Prototyping, 2009. RSP '09. IEEE/IFIP International Symposium on*, Jun. 2009, pp. 116–122.

[8] D. Déharbe and S. Medeiros, "Aspect-oriented design in SystemC: Implementation and applications," in *Proceedings of the 19th Annual Symposium on Integrated Circuits and Systems Design*, ser. SBCCI '06. New York, NY, USA: ACM, 2006, pp. 119–124. [Online]. Available: http://doi.acm.org/10.1145/1150343.1150378

[9] M. Kallel, Y. Lahbib, R. Tourki, and A. Baganne, "Verification of SystemC transaction level models using an aspect-oriented and generic approach," in *Design and Technology of Integrated Systems in Nanoscale Era (DTIS), 2010 5th International Conference on*, Mar. 2010, pp. 1–6.

[10] J. Brandenburg and B. Stabernack, "A generic and non-intrusive profiling methodology for SystemC multi-core platform simulation models," in *Proceedings of the 25th international conference on Architecture of Computing Systems*, ser. ARCS'12. Berlin, Heidelberg: Springer-Verlag, 2012, pp. 135–146.

[11] ——, "Exploring the concurrent execution of HEVC intra encoding algorithms for heterogeneous multi core architectures," in *Design and Architectures for Signal and Image Processing (DASIP), 2015 Conference on*, Sep. 2015, pp. 1–8.

Embedded Parallel Computing Accelorators for Smart Control Units of Frequency Converters

Steffen Vaas, Marc Reichenbach, Johannes Hofmann, Thomas Stadelmayer and Dietmar Fey
Department of Computer Science, Chair of Computer Architecture
Friedrich-Alexander-University Erlangen-Nürnberg (FAU), Germany
{Steffen.Vaas, Marc.Reichenbach, Johannes.Hofmann, Thomas.Stadelmayer, Dietmar.Fey}@fau.de

***Abstract*—**
Classical frequency converters are designed as embedded devices optimized for a specific application-field. But in times of *Industry 4.0* simple frequency converters change to smart control units and become more intelligent with analysis and reporting functions to build up smart grids in automation systems for reducing maintenance costs and increasing productivity. To realize these new functions, an evaluation is needed, which kind of computer architectures should be used for these new devices. Due to more complex algorithms, classical microcontrollers are not sufficient anymore. Therefore, we show in this paper, if and how microprocessors in smart control units can benefit from highly parallel hardware accelerators. Consequently, we propose to increase the performance of an *ARM Cortex-A9* processor by using an *Epiphany III E16* many-core processor as hardware accelerator for complex analysis tasks. Our results show, that a speedup of 1.78 can be achieved, while the power consumption is increased by only 9%.

***Index Terms*—Smart Sensors, Many-Core Processor, Heterogeneous Processor Architecture, Epiphany, Parallella, Embedded Preprocessing**

I. Introduction

For the control of most electrical engines frequency converters are needed. Therefore, the standard solution is to use embedded microcontrollers executing simple PID regulators. Depending on the application area of the electrical engines, there are different requirements for the control algorithms regarding performance, latency and power. Especially the embedded hardware of control units produced in quantity should be just powerful enough to meet the minimum requirements of a stable regulator. In other application-fields like power generator systems, they must be highly accurate to guarantee optimal control. Otherwise power would only be fed inefficiently to the grid. Moreover, robot arms need a low latency for fast movements. Beside controlling of electrical engines, frequency converters can also monitor them. To fulfill the requirements of the *Industry 4.0* a paradigm shift can be seen from simple frequency converters to more powerful smart control units with advanced analysis functions. Simple functions include the determine whether an engine is running or the analysis of different voltage levels. A more complex example is the spectrum analysis of the voltage and current waveforms, for example to detect failures of engines in a very early state, as it was shown in [1] to monitor electrical engines on a ropeway. Thus, maintenance costs can be reduced by additional monitoring information. This way, parts in an automation system that otherwise become defective and interfere with production, can be identified and replaced in time. Furthermore, monitoring information can be provided actively by the frequency converters itself within big automation systems. Another optimization, which can be done by power analysis is feature extraction. Instead of transmitting all acquired raw samples for monitoring, the amount of data to transmit can be reduced significantly by preprocessing the raw values already in the sensor. Thus, only the state of the according electrical engines has to be transmitted. Such *smart sensors*, which combine data acquisition and data preprocessing, can be the solution to realize big, complex systems. To execute all these additional tasks on the power controller, more computing power will be needed.

To gain more computing power, different approaches from the view of computing architectures are possible. Because most frequency converters uses only simple microcontrollers (e.g. ARM Cortex A9), a replacement with a more powerful device (e.g. ARM Cortex A57) is possible. Although this way seams to work, the new device becomes more cost intensive and is also "overpowered" if only a legacy frequency controller is needed. Therefore, we propose in this paper the usage of parallel hardware accelerators. Using this methodology, it is possible to optionally put computational power to a existing device to make it more intelligent or even smarter. In summary, a possible approach is to use a common platform with minimum required hardware resources to provide the main functionalities, like PID controls, which are always needed for every application. Furthermore, there are free slots, where additional accelerator cores can be mounted. These can be necessary to implement more complex filter operations, reducing latency by processing multiple channels in parallel, executing more complex regulator algorithms and the concurrent signal analysis of several channels. Thereby, it is possible to cover a wider application area and reducing the development, as well as the production costs. An abstract view how the described system could look like is shown in Fig. 1.

The goal of this paper is to determine if and how hardware accelerators can help to put more smartness into power control units. As starting point of this work, we have chosen the Epiphany-III many core processor with 16 RISC cores. Due many different channels which have to be processed in power control units, this chip seems to be a good choice for processing acceleration. Moreover it has a low power consumption

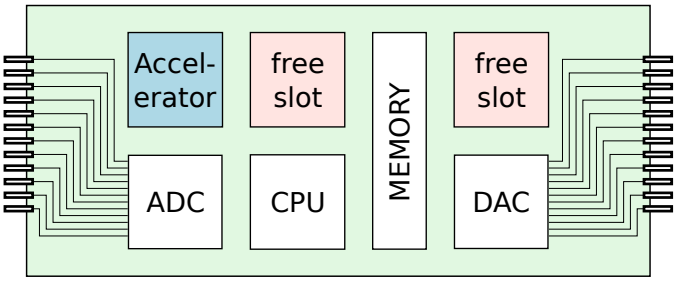


Fig. 1. Platform for an extendable frequency converter control unit.

and due to the Network-on-Chip architecture with off-chip connections, the computation power could be easily increased by putting multiple devices together. Finally, the Epiphany processor is programmable with high-level languages like C/C++ and does not require detailed hardware design knowledge as it is needed when using for example FPGAs as accelerators.

This paper is structured as follows. Section 1 discussed the need of smart control units and explains the need of new architectures for these devices. In the next section some work is presented, which relates to this topic. In Section 3 the algorithms, which run on such a smart control unit are analyzed in more detail. The implementation is described in Section 4. Section 5 presents an evaluation when the usage of hardware accelerators is useful. We will finish this paper with a conclusion and a short outlook.

II. Related Work

Most controllers of frequency converters were designed as highly optimized solutions to reach the needed performance. In [2], for example, the system is optimized for a small scaled robot, a controller for quadrocopters was shown in [3] and a PID using fuzzy logic was presented in [4]. Thus, it is difficult to find a common platform to save time and costs in development.

To gain more flexibility and cover a wider application area, a framework was presented in [5]. Another approach utilizing *Simulink* was shown in [6] In [7] an application-specific instruction set processor (ASIP) was developed to increase flexibility. However, all these proposals are only useful for the same kind of algorithms, like PID controls.

However, there are also control algorithms, which differ drastically from another. For fuzzy logic controllers [8], there are completely different requirements on the hardware than on self-tuning regulators [9]. Moreover, for supporting multiple channels an ability to execute those channels in parallel is needed. In [10] a robot arm with six degrees of freedom was presented, multiple channels have to be controlled in parallel. In [11] even a combination of self-organizing regulators using multiple channels was presented. The presented frameworks and tools are not flexible enough to cover such wide application-fields.

One approach to execute nearly all applications is to utilize FPGAs. As shown in [12], configurable hardware offers large flexibility, so the hardware can be reused. In [13] a combination of a microcontroller and a FPGA was presented to increase the performance by an accelerator core. However, a single architecture, can either be designed for high performance or low power application-fields. Moreover, applying a hardware description language, which is needed to configure FPGAs, increases the development time.

Requirements on the embedded can strongly differ. Beside applications using small scaled low-energy microcontrollers like in [14] and [15], there are other ones requiring more performant hardware, as in [16]. There a DSP serves as processing unit for high performance motor drives.

To cover also these different types of applications, a simple programmable and extendable architecture is required. So our approach is to use a standard embedded architecture, like an *ARM* processor as main control unit and many-core architectures as accelerators, which can be mounted optionally. A suitable many-core processor is the *Epiphany III E16* from *Adapteva*. *Zain Ul-Abdin* demonstrated the performance of the *Epiphany* core by executing radar processing algorithms on it in real-time [17], [18]. Moreover, *Reichenbach et al.* used the *Epiphany* core to execute correlation functions in real-time [19]. So, this architecture will provide the needed performance to serve as optional mountable accelerator core to speedup control and analysis algorithms requiring more performance.

III. Algorithms for Controlling and Analysis

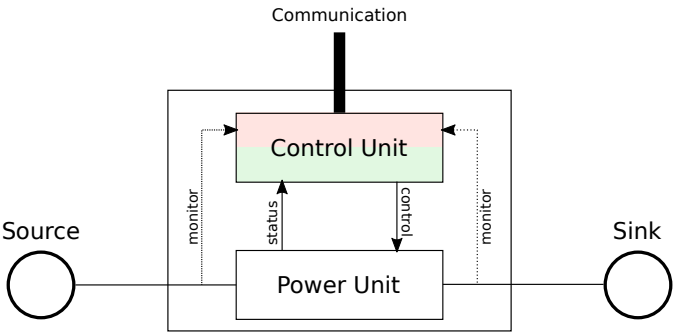


Fig. 2. Illustration of a frequency converter.

In Figure 2 a frequency converter is shown, converting frequency and voltage amplitude of the source signal. Thereby the frequency converter can be divided into a power electronics part, the *Power Unit* and into an embedded hardware controller, illustrated as *Control Unit*. To test an accelerator for a control unit, a reference design for a three phase current was implemented. This reference is presented in Figure 3. On the control unit two main applications are executed: the controlling, which processes the blocks in green and a monitoring application, executing the red marked block.

After sampling the voltage and current values of all 3 phases, filtering is needed at the control application, to achieve a stable regulation. To execute a PID algorithm, the gathered information of all channels has to be transformed into space vectors. Afterwards, another space vector modulation

is required to transform the data back. Then a pulse-width modulation (PWM), which is implemented as a peripheral of the microcontroller, is used to control the *Power Unit*.

For a simultaneous analysis while controlling the frequency converter, a *Monitoring* unit is implemented. There a compute-intensive FFT calculation is needed to get the spectrum of all sampled signals. This spectrum gets analyzed to detect failures in the system, such as damaged bearings of electric engines. These errors can then be reported by a communication interface.

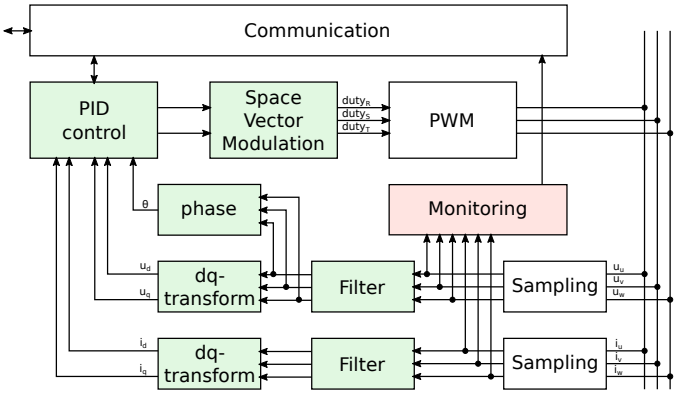


Fig. 3. Realized data flow within the control unit for frequency converters.

IV. Results

To evaluate the algorithms on different architectures, the *Parallella* board from *Adapteva* was used as reference platform. The board owns an *Epiphany III E16* many-core processor and a *Zynq* system-on-chip core from *Xilinx* with an integrated *ARM Cortex-A9* dual core processor. To validate the approach of utilizing hardware accelerators for frequency converter control units, firstly all functionalities were implemented on the *ARM* processor. Then parts of the algorithms were executed on the *Epiphany* accelerator to ascertain if there is a speedup.

A. Speedup of the Execution time

Since the control algorithms of this reference design do not need much processing power and the code is only poor parallelizable, even on the two cores of the *ARM Cortex-A9*, there was no need to speedup the calculations by using an accelerator.

For the monitoring algorithm a FFT is required, which is a highly compute-intensive task. Thus, it was outsourced to the *Epiphany* accelerator for relieving the main CPU. To determine the speedup by an accelerator core, the algorithm was implemented for FFT calculations of 1, 4 and 8 channels in parallel. Thereby two different amounts of input values were considered, with 4096 and 16384 input values. The results are illustrated in Figure 4. For the implementation on the ARM core the *FFTW 3.2.2 ARM* [20] library was used. To increase the performance of the monitoring algorithm, the FFT and the peak detection was calculated on the accelerator

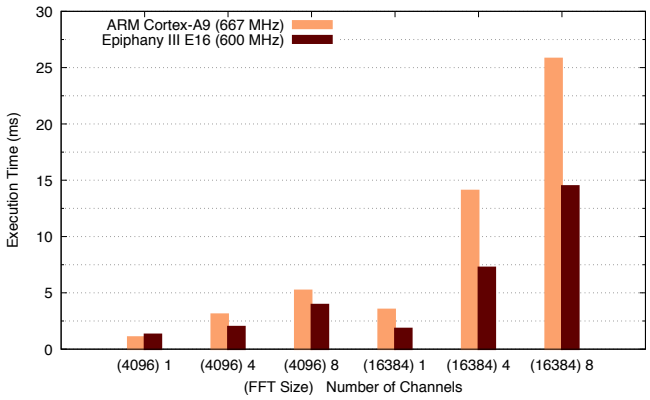


Fig. 4. Measured execution times of different monitoring algorithm configurations.

processor. On the 4096 FFT variant, 4 of 16 *Epiphany* cores were allocated for one FFT. So the monitoring algorithm for 1 channel utilizes the *Epiphany* only up to 25%, what resulted in a longer execution time. Only for the monitoring of 4 or 8 channels the core is utilized completely. On the FFT variant with 16384 input values, all 16 *Epiphany* cores were used for 1 FFT calculation. So the whole accelerator was already utilized completely for monitoring 1 channel. Monitoring multiple channels was also executed consecutively as on the ARM core. In this scenario, a speedup of up to 1.78 was reached.

B. Communication between Processor and Accelerator

With the *Epiphany* an even higher speedup could be reached theoretically, but measurements have shown, that the data transmission to and from the accelerator is the bottleneck of the system. The fastest way to transfer data between the *Cortex-A9* and the *Epiphany* is to use the shared memory of the *Parallella*. The memory read and write access times of both processors are shown in Figure 5 and Figure 6.

However, most tasks executed on the *Epiphany* are memory bound. Only computing-intensive tasks are suited to be executed on the accelerator keeping the amount of communication transfers between the processors low. Another way to increase the performance is to use a DMA controller, which moves raw data from input peripherals directly into the shared memory, as assumed in IV-A. Thus, the data transmission time can be reduced and the *ARM* processor gets relieved.

Due to high transmission times, for the reference design a benefit in performance is only given by outsourcing the compute-intensive parts of the analysis application. For the controlling task the execution time would rise, if the parts of the calculations would have to be transfered to the accelerator firstly. This was also estimated by a roofline model, which is shown in Figure 7.

C. Power Consumption

Apart from performance measurements, on embedded devices power consumption also has to be taken into account.

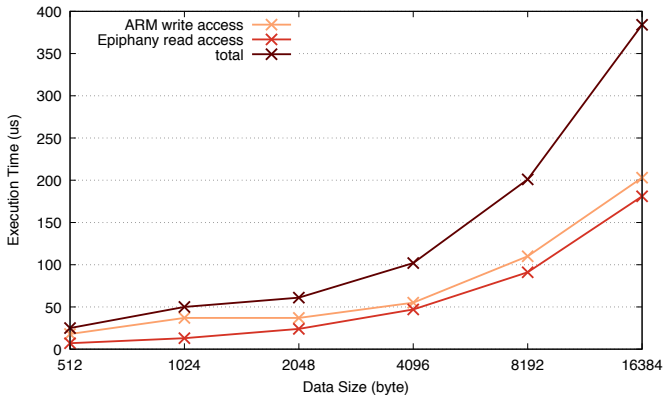


Fig. 5. Execution times to transfer data from *ARM Cortex-A9* to the *Epiphany* on the *Parallella* board.

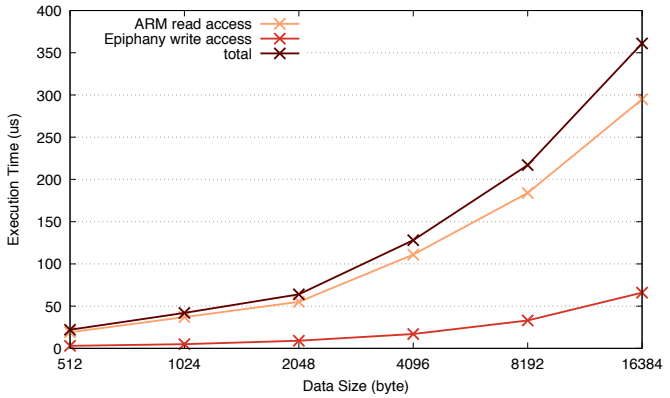


Fig. 6. Execution times to transfer data from the *Epiphany* to the *ARM Cortex-A9* on the *Parallella* board.

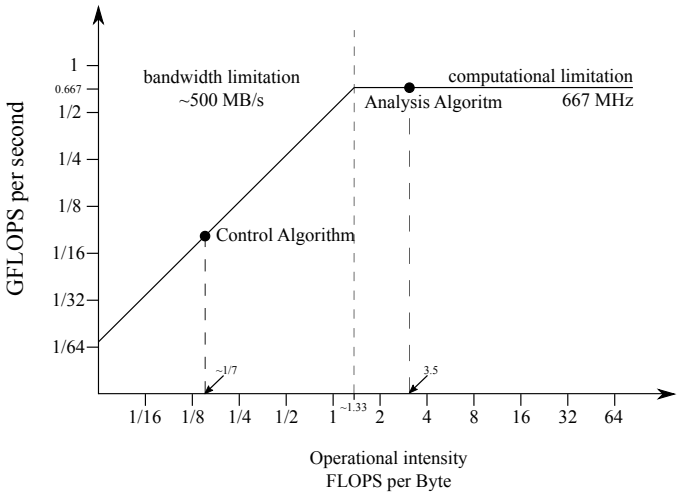


Fig. 7. Roofline Model

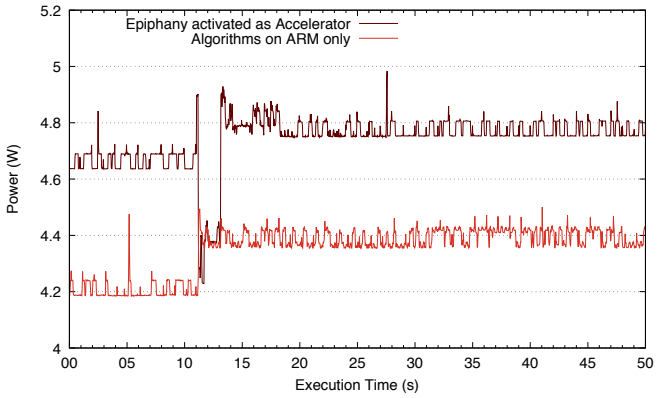


Fig. 8. Power measurement while executing all tasks on the *ARM* processor and enabling the *Epiphany* accelerator.

Therefore, the power consumption was measured during execution of all algorithms on the *ARM* processor and while the *Epiphany* was activated executing the monitoring algorithms. As illustrated in Figure 8 the consumption rose by 0.4 W. The results show, that the whole evaluation board needs 9% more power if the *Epiphany* is activated and the performance increases by 79%. Thus, only 60% of the energy is needed for the same calculation using the accelerator core.

V. Conclusion

To keep up with the requirements of *Industry 4.0*, power control units needs to become more intelligent with internal analysis functions. Therefore, new architecture concepts are necessary to fulfill these requirements. For a flexible and scalable solution, we have shown how hardware accelerators can be used to speedup calculation and allow complex data analysis already at the sensor.

In this paper we presented two applications, a PID controller as well as FFT for power monitoring which have to executed simultaneously at a smart power control unit. The results show, that for a PID controller a standard ARM 9 core is sufficient, while for more complex analysis operations a hardware accelerator is required. Using the Epiphany-III chip from *Adapteva*, a powerful high parallel embedded computing architecture is available which can do the FFT processing in real time and therefore free computational resources at the ARM core in favor of the PID control executed on it.

In this work, we first evaluated which architectures could be used for constructing new smart power control units. Using these results and the presented concept of a flexible and expendable architecture our next step is to realize an own PCB using an ARM 9 core with the capabilities to put several hardware accelerators in this system.

Acknowledgment

The authors would like to thank *Andreas Gröger* and *Dr. Marvin Tannhäuser* from the *Siemens AG* in *Erlangen* for the meaningful discussions their valuable inputs.

References

[1] Schaeffler Technologies AG & Co. KG, "Ina deutschland — mediathek — mobile zukunft – seilbahnüberwachung mit dem fag smartcheck," http://www.ina.de/content.ina.de/de/mediathek/library/library-detail-language.jsp?id=63364608\&ref=rss.

[2] W. Zhao, B. H. Kim, A. Larson, and R. Voyles, "FPGA implementation of closed-loop control system for small-scale robot," in *, 12th International Conference on Advanced Robotics, 2005. ICAR '05. Proceedings*, Jul. 2005, pp. 70–77.

[3] M. N. Duc, T. N. Trong, and Y. S. Xuan, "The quadrotor MAV system using PID control," in *2015 IEEE International Conference on Mechatronics and Automation (ICMA)*, Aug. 2015, pp. 506–510.

[4] J. Guo, G. Wu, and S. Guo, "Fuzzy PID algorithm-based motion control for the spherical amphibious robot," in *2015 IEEE International Conference on Mechatronics and Automation (ICMA)*, Aug. 2015, pp. 1583–1588.

[5] R. Frijns, A. Kamp, S. Stuijk, J. Voeten, M. Bontekoe, K. Gemei, and H. Corporaal, "Dataflow-Based Multi-ASIP Platform Approach for Digital Control Applications," in *2013 Euromicro Conference on Digital System Design (DSD)*, Sep. 2013, pp. 811–814.

[6] J. Lazaro, A. Astarloa, J. Arias, U. Bidarte, and A. Zuloaga, "Simulink/Modelsim Simulabel VHDL PID Core for Industrial SoPC Multiaxis Controllers," in *IECON 2006 - 32nd Annual Conference on IEEE Industrial Electronics*, Nov. 2006, pp. 3007–3011.

[7] D. Abramovitch, "A unified framework for analog and digital PID controllers," in *2015 IEEE Conference on Control Applications (CCA)*, Sep. 2015, pp. 1492–1497.

[8] R. Kazemi, R. Vesilo, and E. Dutkiewicz, "A Novel Genetic-Fuzzy Power Controller with Feedback for Interference Mitigation in Wireless Body Area Networks," in *Vehicular Technology Conference (VTC Spring), 2011 IEEE 73rd*, May 2011, pp. 1–5.

[9] X.-h. Liu and L. Xu, "Research on tension control system based on fuzzy self-tuning PID control," in *Control and Decision Conference (CCDC), 2010 Chinese*, May 2010, pp. 3385–3390.

[10] J.-S. Kim, H.-W. Jeon, and S. Jung, "Hardware implementation of nonlinear PID controller with FPGA based on floating point operation for 6-DOF manipulator robot arm," in *International Conference on Control, Automation and Systems, 2007. ICCAS '07*, Oct. 2007, pp. 1066–1071.

[11] H. Kazemian, "The SOF-PID controller for the control of a MIMO robot arm," *IEEE Transactions on Fuzzy Systems*, vol. 10, no. 4, pp. 523–532, Aug. 2002.

[12] L. Barreto, P. Praca, C. Cruz, and R. Bascope, "PID Digital Control Using Microcontroller and FPGA Applied to a Single-Phase Three-Level Inverter," in *APEC 2007 - Twenty Second Annual IEEE Applied Power Electronics Conference*, Feb. 2007, pp. 1443–1446.

[13] R. Alba-Flores, F. Rios-Gutierrez, and C. Jeanniton, "Qualitative evaluation of a PID controller for autonomous mobile robot navigation implemented in an FPGA card," in *2011 Seventh International Conference on Natural Computation (ICNC)*, vol. 3, Jul. 2011, pp. 1753–1757.

[14] S. Sarin, H. Hindersah, and A. Prihatmanto, "Fuzzy PID controllers using 8-Bit microcontroller for U-Board speed control," in *2012 International Conference on System Engineering and Technology (ICSET)*, Sep. 2012, pp. 1–6.

[15] C. Xu, D. Huang, Y. Huang, and S. Gong, "Digital PID controller for Brushless DC motor based on AVR microcontroller," in *IEEE International Conference on Mechatronics and Automation, 2008. ICMA 2008*, Aug. 2008, pp. 247–252.

[16] A. Rubaai, M. Castro-Sitiriche, and A. Ofoli, "DSP-Based Implementation of Fuzzy-PID Controller Using Genetic Optimization for High Performance Motor Drives," in *Conference Record of the 2007 IEEE Industry Applications Conference, 2007. 42nd IAS Annual Meeting*, Sep. 2007, pp. 1649–1656.

[17] Zain-ul-Abdin, A. Ahlander, and B. Svensson, "Energy-Efficient Synthetic-Aperture Radar Processing on a Manycore Architecture," in *2013 42nd International Conference on Parallel Processing (ICPP)*, Oct. 2013, pp. 330–338.

[18] Z. Ul-Abdin and M. Yangt, "Dataflow programming of real-time radar signal processing on manycores," in *2014 IEEE Global Conference on Signal and Information Processing (GlobalSIP)*, Dec. 2014, pp. 15–19.

[19] M. Reichenbach, M. Kasparek, M. Alawieh, K. Haublein, and D. Fey, "Real-time correlation for locating systems utilizing heterogeneous computing architectures," in *2015 Conference on Design and Architectures for Signal and Image Processing (DASIP)*, Sep. 2015, pp. 1–8.

[20] Vesperix Corporation, "FFTW 3.2.2 ARM," http://www.vesperix.com/arm/fftw-arm/.

Optimizing Parallel Runtime of Cryptanalytic Algorithms by Selecting Between Word-Parallel and Bit-Serial Variants of Program Parts

Patrick Eitschberger and Jörg Keller

Faculty of Mathematics and Computer Science
FernUniversität in Hagen
58084 Hagen, Germany
<firstname>.<lastname>@fernuni-hagen.de

Abstract: Cryptanalytic algorithms such as dictionary attacks, that test huge numbers of keys to decrypt a ciphertext to a certain plaintext, need lots of computational resources and efficient coding, but allow large scale parallelism such as many-cores plus GPUs. Some attacks have profited from a bit-serial data representation, that allows SIMD-like coding per thread and increases the degree of parallelism. We investigate the question how to decide for distinct parts of such algorithms whether to code them in a bit-serial or normal word-parallel manner. Given bit-serial and word-parallel variants for each part of the cryptographic algorithm, we benchmark the runtime of the variants, and additionally the runtime of the conversion between the different data representations. Then we model the resulting variant selection problem as a direct graph — in the fashion of a global composition optimization problem — and find the optimal runtime by computing the shortest path from source to sink node. We evaluate our approach with the Advanced Encryption Standard (AES) and demonstrate runtime advantages.

Keywords: Cryptanalytic Algorithm; Bit-Serial Computing; Global Optimization; Performance Tuning; Parallel Algorithm

1 Introduction

In recent years, parallel computing often means heterogeneous parallel computing with different processor architectures and accelerators such as GPUs. Finding the optimal allocation of work to different execution units thus becomes a non-trivial optimization problem. This optimization problem is complicated by other aspects as well, such as e.g. the best choice of a (parallel) sorting algorithm, which depends on the number of items to be sorted, their pre-sorting state, and the available implementations of different variants of sorting algorithms on the different execution units. In this manner, performance tuning of parallel applications has become a complex optimization problem that must be solved partly at algorithm design time, partly at compile time, and partly at execution time.

In the present work, we consider cryptographic applications like a dictionary attack (cf. e.g.

[MvOV97]), that execute millions to billions of decrypt operations on the same piece of ciphertext, but with different keys. Hence, these applications exhibit a tremendous amount of independent parallelism. As many encryption algorithms evaluate boolean functions during their execution, bit-serial (sometimes called bit-slice) computing in SIMD fashion (see Sect. 2) has been used for long to increase parallelism, improve speed, and reduce control-flow divergence on GPU architectures.

Our approach to further improve performance is to split a cryptographic algorithm into distinct parts, and provide both a normal (i.e. word-parallel) and a bit-serial implementation variant for each part. Additionally, we employ a known routine for data conversion between the two variants. Now, the application can be modelled in the fashion of a global composition optimization problem [HK14] as a directed graph, where nodes are variants attributed with their runtime, and arcs represent the flow of execution, and might be attributed with the conversion runtime if an arc's head and tail use different variants. By finding the shortest path from source to sink, the best combination of variants is found. To our knowledge, this represents a novel use for global composition of program variants, and has never been used to optimize parallel cryptographic algorithms.

As a case study, we apply our approach to an implementation of the Advanced Encryption Standard (AES) [Nat01], forecast an optimal mix of variants and demonstrate in experiments that runtime advantages over both purely bit-serial and word-parallel implementations are indeed possible.

The remainder of this work is structured as follows. In Sect. 2, we summarize basics about bit-serial computing, while Sect. 3 briefly reviews the global composition of program variants. In Sect. 4, we briefly summarize the AES algorithm as the object of our case study, apply the optimization algorithm from Sect. 3, and report our experimental results. Section 5 concludes and gives an outlook to future work.

2 Bit-serial Computing

Bit-slice processors, i.e. processors with a data width much smaller than a normal word width — in the extreme case called serial or bit-serial processors — have been known for long, e.g. in the Connection Machine [KH89]. Typically, a number of these processors work together in SIMD fashion to operate on data of normal width.

Also, the same concept has been known in software for decades. Biham [Bih97] "view(s) the processor as a SIMD computer, i.e., as 64 parallel one-bit processors computing the same instruction" and sees bit-serial computing mainly as a "non-standard representation" of data. While this may sound strange at first glance, it ensures that all data bits are used by parallelism, which is often not the case in normal computations, e.g. when the instructions operate on bytes or even on single bits while evaluating a logical expression.

We illustrate this concept with three small examples: one that favors bit-serial computing, one that favors normal data representation, and one where it depends on the circumstances which one is better.

Assume that we want to evaluate a boolean expression for many parameter values, e.g. $y_i = a_i \wedge b_i$ for $i = 0, 1, \ldots, 31$. Normally, we code

```
int i;
int y[32],a[32],b[32];
// ... set values in arrays a and b
for(i=0;i<32;i++) y[i] = a[i] & b[i];
```

and apply parallelism in the form of loop parallelization. Yet, if we transfer the lowest bits from each array element `a[i]` into one variable `as` such that the bit from `a[i]` is the ith bit in `as`, then we can simply write

```
int ys,as,bs;
// ... set values in as and bs
ys = as & bs;
```

If 2^{16} evaluations are to be done, then with 32 threads, each thread would have to do $2^{11} = 2048$ evaluations in normal representation, but only $2^5 = 32$ evaluations with bit-serial representation. If surrounding operations are also expressed in this manner, conversion of the representation is not necessary. Thus, bit-serial computing is advantageous in this case.

As a second example, consider that we want to do additions on 32-bit integer variables. The normal code is obvious and similar to the first example:

```
int i;
int y[32],a[32],b[32];
// ... set values in arrays a and b
for(i=0;i<32;i++) y[i] = a[i] + b[i];
```

In bit-serial representation, the data is organized in a manner orthogonal to the normal representation: variable `as[j]` contains bit j of each variable `a[i]` in bit i. This is illustrated in Fig. 1 for $j = 0$. Then, addition is performed bit by bit as in a full adder:

```
int j;
int ys[32],as[32],bs[32],cs[33];
// ... set arrays as and bs, and set array cs to 0
for(j=0;j<32;j++){ ys[j] = as[j] ^ bs[j] ^ cs[j];
cs[j+1] = (as[j] & bs[j]) | ((as[j] ^ bs[j]) & cs[j]); }
```

Please note that the conversion between the normal and the bit-serial data representations is nothing more than the transposition of a bit matrix with the variables `a[i]` and `as[i]` ($i = 0, \ldots, 31$) being the row vectors of the matrix and transposed matrix, respectively (cf. Fig. 2). An efficient algorithm for bit transposition is given in [RSD06], and we will use a variant in the sequel.

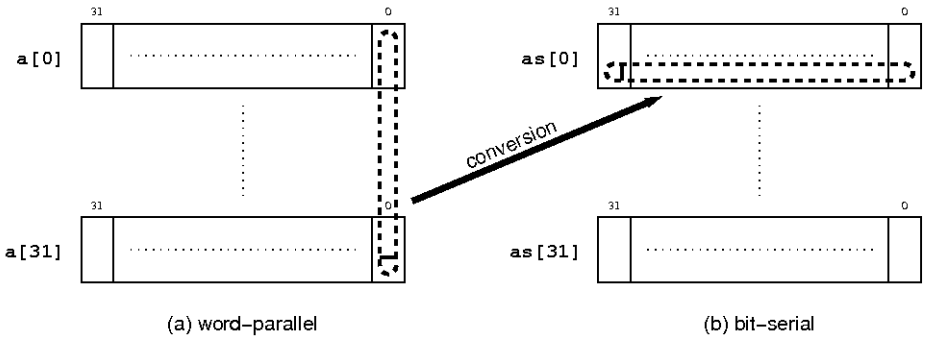


Figure 1: Corresponding word-parallel and bit-serial data representations.

Obviously, bit-serial representation is not advantageous in this second case, as an addition is replaced by 6 logical operations, and two assignments are used instead of one.

Finally, assume that we do a table lookup in a constant array defined over bytes.

```
int i;
uint8 y[32], a[32];
uint8 tab[256];
for(i=0;i<32;i++) y[i] = tab[a[i]];
```

Here, the bit-serial variant is not straightforward. If we use a 32-bit variable `ys[0]`, that contains the bit 0 of each variable `y[i]`, then bit i of `ys[0]` will depend on all bits of `a[i]` and on `tab`. Thus, a table for looking up `ys[0]` would be infeasibly large. However, we can express the dependence of each bit of `y[i]` on the 8 bits of `a[i]` and on `tab` by a boolean function in at most 8 variables, as tab is a constant array.

```
uint32 ys[8], as[8];
ys[0] = some boolean function on as[0] to as[7];
...
ys[7] = another boolean function on as[0] to as[7];
```

Hence, depending on the complexity of these boolean functions, the code might be slower or faster than the original table lookup. For example, if `tab[x]` would give the number of bits set in the binary representation of x (where $0 \leq x \leq 255$), then `ys[7]` to `ys[4]` would be 0, as the maximum number of bits set could be 8 (=00001000 in 8-bit binary), and `ys[3] = as[0] & ... & as[7]`.

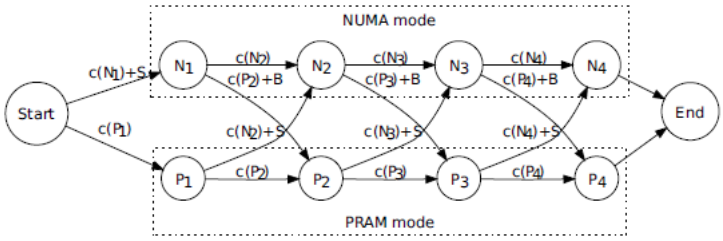


Figure 2: Graph with 4 program parts, each with 2 variants, taken from [HK14].

3 Global Optimization of Variants

To explain the global composition optimization, we use a very simple example. Consider an application that proceeds in rounds, where each round consists of two code parts. Each part is available in two variants A and B. The variants of each part consist of different code, and different data representation, so that combining variant A of part 1 with variant B of part 2 involves a conversion of the data representation between part 1 and 2, and another conversion between part 2 and part 1 of the next round. In the following, a_i and b_i denote the numbers of cycles for variants A and B of part i, respectively. Furthermore, c denotes the number of cycles needed for a conversion in either direction.

If $a_1 < b_1$ and $a_2 < b_2$, then clearly variant A should be chosen. The same holds if variant B is better in both parts. We consider the interesting case where $a_1 < b_1$ but $b_2 < a_2$. Let us assume that the differences are identical, i.e. $d = |a_1 - b_1| = |a_2 - b_2|$. Then the combination of variant A in part 1 and variant B in part 2 should be chosen if $2c < d$, because then

$$a_1 + c + b_2 + c < a_1 + a_2 = b_1 + b_2 \ .$$

Clearly, the same idea can be used if $|a_1 - b_1| \neq |a_2 - b_2|$, but more cases have to be considered to find the optimum. The idea can also be generalized to more than two parts and to more than two variants per part.

This problem has been investigated as *global composition optimization* [HK14], and treats the variant problem formally by modelling with a directed graph, where each variant for each program part is a node, attributed with its runtime, and arcs represent the flow of execution, where each variant of one part is connected to each variant of the following part (i.e. piecewise complete bipartite). If an arc connects different variants, it is attributed with the runtime of the conversion code. An example with 4 parts is depicted in Fig. 2 taken from [HK14]. The best combination of variants is found by computing the shortest path from source to sink. Note that optimization can be done over several rounds [HK16], so that performance improvements might even be possible if $c > d/2$.

4 Case Study

An application domain where mainly boolean operations and table lookups are performed is encryption. Thus, from the above, bit-serial representation looks advantageous. Also fixed bit permutations, which frequently arise in encryption, are easy with bit-serial representation as only the indices of the `as` variables have to be permuted accordingly. Biham [Bih97] already demonstrated that bit-serial representation leads to more efficient implementation of the Data Encryption Standard (DES). Similar approaches has been implemented for its successor Advanced Encryption Standard (AES) [KS09, RSD06].

We therefore illustrate the so far rather abstract idea of mixing word-parallel and bit-serial program parts with Advanced Encryption Standard (AES) as a concrete example. AES is a standard for a symmetric block cipher based on the Rijndael algorithm [DR00], chosen in 2000 by NIST as the successor to DES and published in 2001 as a standard [Nat01]. Encryption of a data block consists of a number of rounds (10 to 14, depending on key size), where each round consists of four steps operating on a 4×4-matrix of bytes: bytewise substitution, rotating the matrix rows by different stepwidths, mix the columns by multiplication with a constant matrix, and bitwise addition of the pre-computed round key to the data matrix. The following pseudo-code illustrates the computations.

```
AES(uint8 w[4][4]){ // input byte matrix
for(rnd=0..9){ // 10 rounds for 128-bit key
for(i,j=0..3) w[i][j] = tab[w[i][j]]; // byte substitution
for(i,j=0..3) wtmp[i][j] = w[i][(j+step[j])%4]; // shift rows
for(i,j=0..3){ w[i][j] = 0;
 for(k=0..3) w[i][j] += mul(cnst[i][k],wtmp[k][j]); } // mixcols
for(i,j=0..3) w[i][j] = w[i][j]^rndkey[rnd][i][j]; // add rndkey
}
return w;  }
```

There have been high-performance AES implementations in software for 8-bit and 32-bit microprocessors (e.g. the add-round-key step greatly profits on a 32-bit architecture), and also implementations in hardware for ASICs and FPGAs. In addition, there have been bit-serial implementations, where all steps have been expressed as evaluation of boolean functions, so that 32 block encryptions can go on in parallel if 32-bit variables are used [KS09, RSD06]. The reader might notice that the steps correspond closely to our code examples from Sect. 2, and the cited implementations proceed like this, in particular they give a formulation of the subbytes and mixcolumns steps expressed as boolean function evaluation.

We have implemented both variants[1] for a block and key size of 16 bytes and measured the runtimes in Tab. 1 on a Lenovo W530 with Intel Core i7-3630QM (Ivybridge) quad-core CPU (up to 2.4 GHz, with 3.4 GHz turbo), 20 GByte of RAM, Windows 7 operating and OpenWatcom C compiler. We encrypt one block of 16 bytes for 10 million times, and compute the resulting runtime. As the computation is independent of the concrete content of the byte matrix `w`, we used the same block in all encryptions. We do not claim to have

[1] For the word-parallel variant, we multiplied the measured times by 32 to get comparable results.

Part	word-parallel	bit-serial
subbytes	14337	38563
shiftrows	5788	0
mixcolumns	27908	5991
addroundkey	6427	10062
conversion	11856	11856

Table 1: Runtimes of bit-serial and word-parallel implementations of AES. Runtimes are given without dimension, as they are computed with `clock()` over 10 million repetitions.

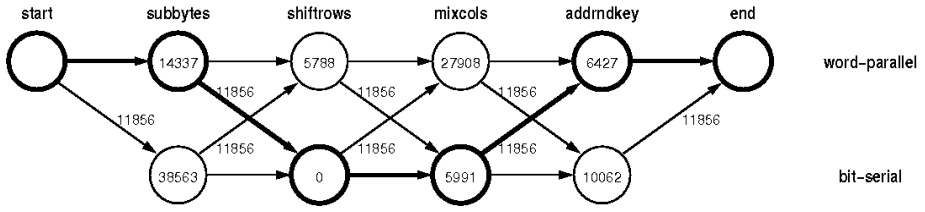


Figure 3: Flow graph of AES variants. Nodes and edges without mark have weight 0.

the fastest implementations for each part, but we strived to have a comparable level of code quality, so that the comparison is fair.

We clearly see that for the first and fourth steps (subbytes and addroundkey) the word-parallel variant is faster, while for the second and third steps (shiftrow and mixcolumns), the bit-serial variant is faster. Figure 3 depicts the flow graph of the variants with the shortest path highlighted[2]. Therefore, a mixed implementation looks like in the following pseudo-code (wp=word-parallel, bs=bit-serial).

```
AES(uint8 w[4][4]){ // input byte matrix
for(rnd=0..9){ // 10 rounds for 128-bit key
for(i,j=0..3) w[i][j] = tab[w[i][j]]; // byte subst wp
bittranspose(w); // convert representation to bs
shiftrow+mixcolumnbitserial(w); // do next two steps bs
bittranspose(w); // convert representation back to wp
for(i,j=0..3) w[i][j]=w[i][j]^rndkey[rnd][i][j]; // add rndk. wp
}
return w;  }
```

We need two conversions per round. We pack steps subbytes and addroundkey as part 1, and shiftrow and mixcolumns as part 2, and see that for part 1, variant A (word-parallel) is faster, while for part 2, variant B (bit-serial) is faster. We get

[2]Note that to get a complete picture, one also has to do the same with start and end in bit-serial representation. This however leads not to a shorter path in this case.

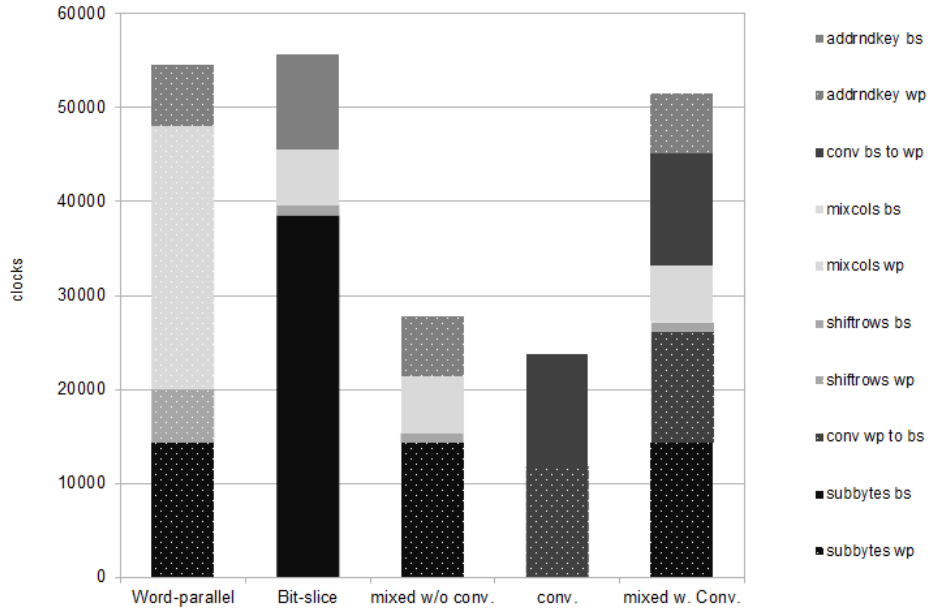


Figure 4: Runtimes of different code variants. Corresponding word-parallel (wp) and bit-serial (bs) code parts have same color but word-parallel variants are dotted. Bit-serial shiftrows really has runtime 0, but has been denoted by small rectangle for clarity.

Variant	Runtime
a_1	20764
a_2	33696
a_{tot}	54460
b_1	48625
b_2	5991
b_{tot}	54616
c	11856

Thus, the bit-serial and word-parallel variants almost have the same runtimes a_{tot} and b_{tot}. The optimal path (without conversion cost) is a_1 and b_2, the add-on for a pure variant is $d = 27861$ (or 27705, i.e. more or less the same for both variants). The double conversion cost is $2c = 23712$ which is lower, so that the runtime advantage of the mixed implementation is about 4000 or 7.4%. Figure 4 illustrates the different runtimes.

While this improvement seems not to be very large, it illustrates that in some applications,

it might be worthwhile to consider this choice. Please note that if conversion would be free (or almost free by providing hardware instructions for bit matrix conversion), then speed could almost be doubled.

5 Conclusions

We have presented a novel application of global composition of program variants: using both bit-serial and word-parallel variants of algorithmic parts in symmetric encryption. The bit-serial variants allow SIMD parallelism even within a single core, while the word-parallel variants serialize the threads of computation but avoid the overhead of bit-serial computation on data items when all bits of a word are needed.

Our case study demonstrates that the AES encryption algorithm can be accelerated compared to purely bit-serial and word-parallel implementations by combining the best parts of both. The massively parallel execution of AES frequently occurs for good and bad: in high-performance environments where multiple communications are encrypted simultaneously, and in dictionary attacks where attackers try to find the password by decrypting a known piece of text with all keys from a dictionary, exploiting the fact that many users still employ existing expressions as a key or password.

Future work will comprise investigation of other use cases, as well as more advanced uses: between computing on single bits (like in boolean function evaluation) and computing on full words (like in ordinary arithmetic computation), there are lots of in-betweens, like e.g. computations on bytes, that still could profit from SIMD parallelism. Such an approach has already been investigated in the frame of multiple executions for fault-tolerance (cf. [EFK09]), but not for performance improvement from parallelism.

Acknowledgements

Our sincerest thanks go to Erik Hansson and Christoph Kessler for long discussions about global composition, that inspired the present work.

References

[Bih97] Eli Biham. A fast new DES implementation in software. In *Proc. 4th International Workshop on Fast Software Encryption (FSE '97)*, pages 260–272. Springer LNCS, 1997.

[DR00] Joan Daemen and Vincent Rijmen. Rijndael for AES. In *AES Candidate Conference*, pages 343–348, 2000.

[EFK09] Klaus Echtle, Bernhard Fechner, and Jörg Keller. PAMOS and PAROS — Parallel Addition of Multiple or Redundant Operands in a Single Word. In *Proc. ESREL 2009*

Conference, 2009.

[HK14] Erik Hansson and Christoph Kessler. Global Optimization of Execution Mode Selection for the Reconfigurable PRAM-NUMA Multicore Architecture REPLICA. In *Proc. 2nd International Symposium on Computing and Networking (CANDAR 2014)*, pages 322–328, 2014.

[HK16] Erik Hansson and Christoph Kessler. Optimized Variant-Selection Code Generation for Loops on Heterogeneous Multicore Systems. In *Proc. International Conference on Parallel Computing (ParCo 2015)*, pages 103–112. IOS Press, 2016.

[KH89] B. A. Kahle and W. Daniel Hillis. The Connection Machine model CM1 architecture. *IEEE Transactions on Systems, Man, and Cybernetics*, 19:707–713, 1989.

[KS09] Emilia Käsper and Peter Schwabe. Faster and Timing-Attack Resistant AES-GCM. In *Proc. 11th International Workshop on Cryptographic Hardware and Embedded Systems (CHES 2009)*, pages 1–17. Springer LNCS, 2009.

[MvOV97] Alfred J. Menezes, Paul C. van Oorschot, and Scott A. Vanstone. *Handbook of Applied Cryptography*. CRC Press, 1997.

[Nat01] National Institute of Standards and Technology (NIST). *Federal Information Processing Standards Publication 197: Announcing the Advanced Encryption Standard (AES)*. NIST, 2001.

[RSD06] Chester Rebeiro, David Selvakumar, and A.S.L. Devi. Bitslice Implementation of AES. In *Proc. 5th International Conference on Cryptology and Network Security (CANS 2006)*, pages 203–212. Springer LNCS, 2006.

1. Aktuelle und zukünftige Aktivitäten (Bericht des Sprechers)

Die 33. Ausgabe der PARS-Mitteilungen enthält die Beiträge des 12. PASA-Workshops, der die wesentliche Aktivität der Fachgruppe im Jahr 2016 darstellt.

Der 12. PASA-Workshop fand am 4. April 2016 in Nürnberg im Rahmen der ARCS-Tagung statt. Der Workshop, der aus 4 Vorträgen bestand, war mit ca. 30 Teilnehmern gut besucht. Den Herren Professoren Wanka (PASA) und Fey (ARCS) sei für die Organisation des Workshops gedankt. Der PARS-Nachwuchspreis wurde in diesem Jahr nicht vergeben.

Unser nächster Workshop ist der

27. PARS-Workshop voraussichtlich am 4. und 5. Mai 2016 in Hagen.

Wegen Erkrankung des Sprechers der Fachgruppe muss auf diesem Workshop im Rahmen einer Mitgliederversammlung ein neuer Sprecher gewählt werden. Bereits seit Herbst 2015 führt Professor Karl als stellvertretender Sprecher die Geschäfte der Fachgruppe.

Aktuelle Informationen finden Sie auch auf der PARS-Webpage

http://fg-pars.gi.de/

Anregungen und Beiträge für die Mitteilungen können an den stellv. Sprecher (wolfgang.karl@kit.edu) gesendet werden.

Wir wünschen Ihnen einen guten Start ins Wintersemester und schon jetzt ein gesundes und erfolgreiches Jahr 2017.

Karlsruhe und Hagen, im September 2016
Wolfgang Karl und Jörg Keller

2. Zur Historie von PARS

Bereits am Rande der Tagung CONPAR81 vom 10. bis 12. Juni 1981 in Nürnberg wurde von Teilnehmern dieser ersten CONPAR-Veranstaltung die Gründung eines Arbeitskreises im Rahmen der GI: Parallel-Algorithmen und -Rechnerstrukturen angeregt. Daraufhin erfolgte im Heft 2, 1982 der GI-Mitteilungen ein Aufruf zur Mitarbeit. Dort wurden auch die Themen und Schwerpunkte genannt:

1) **Entwurf von Algorithmen für**
   - verschiedene Strukturen (z. B. für Vektorprozessoren, systolische Arrays oder Zellprozessoren)
   - Verifikation
   - Komplexitätsfragen
2) **Strukturen und Funktionen**
   - Klassifikationen
   - dynamische/rekonfigurierbare Systeme
   - Vektor/Pipeline-Prozessoren und Multiprozessoren
   - Assoziative Prozessoren
   - Datenflussrechner
   - Reduktionsrechner (demand driven)
   - Zellulare und systolische Systeme
   - Spezialrechner, z. B. Baumrechner und Datenbank-Prozessoren
3) **Intra-Kommunikation**
   - Speicherorganisation
   - Verbindungsnetzwerke
4) **Wechselwirkung zwischen paralleler Struktur und Systemsoftware**
   - Betriebssysteme
   - Compiler
5) **Sprachen**
   - Erweiterungen (z. B. für Vektor/Pipeline-Prozessoren)
   - (automatische) Parallelisierung sequentieller Algorithmen
   - originär parallele Sprachen
   - Compiler
6) **Modellierung, Leistungsanalyse und Bewertung**
   - theoretische Basis (z. B. Q-Theorie)
   - Methodik
   - Kriterien (bezüglich Strukturen)
   - Analytik

In der Sitzung des Fachbereichs 3 ‚Architektur und Betrieb von Rechensystemen' der Gesellschaft für Informatik am 22. Februar 1983 wurde der Arbeitskreis offiziell gegründet. Nachdem die Mitgliederzahl schnell anwuchs, wurde in der Sitzung des Fachausschusses 3.1 ‚Systemarchitektur' am 20. September 1985 in Wien der ursprüngliche Arbeitskreis in die Fachgruppe FG 3.1.2 ‚Parallel- Algorithmen und -Rechnerstrukturen' umgewandelt.

Während eines Workshops vom 12. bis 16. Juni 1989 in Rurberg (Aachen) - veranstaltet von den Herren Ecker (TU Clausthal) und Lange (TU Hamburg-Harburg) - wurde vereinbart, Folgeveranstaltungen hierzu künftig im Rahmen von PARS durchzuführen.

Beim Workshop in Arnoldshain sprachen sich die PARS-Mitglieder und die ITG-Vertreter dafür aus, die Zusammenarbeit fortzusetzen und zu verstärken. Am Dienstag, dem 20. März 1990 fand deshalb in

München eine Vorbesprechung zur Gründung einer gemeinsamen Fachgruppe PARS statt.
Am 6. Mai 1991 wurde in einer weiteren Besprechung eine Vereinbarung zwischen GI und ITG sowie eine Vereinbarung und eine Ordnung für die gemeinsame Fachgruppe PARS formuliert und den beiden Gesellschaften zugeleitet. Die GI hat dem bereits 1991 und die ITG am 26. Februar 1992 zugestimmt.

3. Bisherige Aktivitäten

Die PARS-Gruppe hat in den vergangenen Jahren mehr als 20 Workshops durchgeführt mit Berichten und Diskussionen zum genannten Themenkreis aus den Hochschulinstituten, Großforschungseinrichtungen und der einschlägigen Industrie. Die Industrie - sowohl die Anbieter von Systemen wie auch die Anwender mit speziellen Problemen - in die wissenschaftliche Erörterung einzubeziehen war von Anfang an ein besonderes Anliegen. Durch die immer schneller wachsende Zahl von Anbietern paralleler Systeme wird sich die Mitgliederzahl auch aus diesem Kreis weiter vergrößern.

Neben diesen Workshops hat die PARS-Gruppe die örtlichen Tagungsleitungen der CONPAR-Veranstaltungen:

CONPAR 86 in Aachen,
CONPAR 88 in Manchester,
CONPAR 90 / VAPP IV in Zürich und
CONPAR 92 / VAPP V in Lyon
CONPAR 94/VAPP VI in Linz

wesentlich unterstützt. In einer Sitzung am 15. Juni 1993 in München wurde eine Zusammenlegung der Parallelrechner-Tagungen von CONPAR/VAPP und PARLE zur neuen Tagungsserie EURO-PAR vereinbart, die vom 29. bis 31. August 1995 erstmals stattfand:

Euro-Par'95 in Stockholm

Zu diesem Zweck wurde ein „Steering Committee" ernannt, das europaweit in Koordination mit ähnlichen Aktivitäten anderer Gruppierungen Parallelrechner-Tagungen planen und durchführen wird. Dem Steering Committee steht ein „Advisory Board" mit Personen zur Seite, die sich in diesem Bereich besonders engagieren. Die offizielle Homepage von Euro-Par ist **http://www.europar.org/**.
Weitere bisher durchgeführte Veranstaltungen:

Euro-Par'96 in Lyon
Euro-Par'97 in Passau
Euro-Par'98 in Southampton
Euro-Par'99 in Toulouse
Euro-Par 2000 in München
Euro-Par 2001 in Manchester
Euro-Par 2002 in Paderborn
Euro-Par 2003 in Klagenfurt
Euro-Par 2004 in Pisa
Euro-Par 2005 in Lissabon
Euro-Par 2006 in Dresden
Euro-Par 2007 in Rennes
Euro-Par 2008 in Gran Canaria
Euro-Par 2009 in Delft
Euro-Par 2010 in Ischia
Euro-Par 2011 in Bordeaux
Euro-Par 2012 in Rhodos
Euro-Par 2013 in Aachen
Euro-Par 2014 in Porto
Euro-Par 2015 in Wien
Euro-Par 2016 in Grenoble

Außerdem war die Fachgruppe bemüht, mit anderen Fachgruppen der Gesellschaft für Informatik übergreifende Themen gemeinsam zu behandeln: Workshops in Bad Honnef 1988, Dagstuhl 1992 und Bad Honnef 1996 (je zusammen mit der FG 2.1.4 der GI), in Stuttgart (zusammen mit dem Institut für Mikroelektronik) und die PASA-Workshop-Reihe 1991 in Paderborn, 1993 in Bonn, 1996 in Jülich, 1999 in Jena, 2002 in Karlsruhe, 2004 in Augsburg, 2006 in Frankfurt a. Main und 2008 in Dresden (jeweils gemeinsam mit der GI-Fachgruppe 0.1.3 ‚Parallele und verteilte Algorithmen (PARVA)') sowie 2012 in München, 2014 in Lübeck und 2016 in Nürnberg (gemeinsam mit der GI-Fachgruppe ALGO, die Nachfolgegruppe von PARVA).

PARS-Mitteilungen/Workshops:

Aufruf zur Mitarbeit, April 1983 (Mitteilungen Nr. 1)

Erlangen, 12./13. April 1984 (Mitteilungen Nr. 2)

Braunschweig, 21./22. März 1985 (Mitteilungen Nr. 3)

Jülich, 2./3. April 1987 (Mitteilungen Nr. 4)

Bad Honnef, 16.-18. Mai 1988 (Mitteilungen Nr. 5, gemeinsam mit der GI-Fachgruppe 2.1.4 'Alternative Konzepte für Sprachen und Rechner')

München Neu-Perlach, 10.-12. April 1989 (Mitteilungen Nr. 6)

Arnoldshain (Taunus), 25./26. Januar 1990 (Mitteilungen Nr. 7)

Stuttgart, 23./24. September 1991, "Verbindungsnetzwerke für Parallelrechner und Breitband-Übermittlungssysteme" (Als Mitteilungen Nr. 8 geplant, gem. mit ITG-FA 4.1, 4.4 und GI/ITG FG Rechnernetze, wg. Kosten nicht erschienen. siehe Tagungsband Inst. für Mikroelektronik Stuttgart.)

Paderborn, 7./8. Oktober 1991, "Parallele Systeme und Algorithmen" (Mitteilungen Nr. 9, 2. PASA-Workshop)

Dagstuhl, 26.-28. Februar 1992, "Parallelrechner und Programmiersprachen" (Mitteilungen Nr. 10, gemeinsam mit der GI-Fachgruppe 2.1.4 'Alternative Konzepte für Sprachen und Rechner')

Bonn, 1./2. April 1993, "Parallele Systeme und Algorithmen" (Mitteilungen Nr. 11, 3. PASA-Workshop)

Dresden, 6.-8. April 1993, "Feinkörnige und Massive Parallelität" (Mitteilungen Nr. 12, zusammen mit PARCELLA)

Potsdam, 19./20. September 1994 (Mitteilungen Nr. 13, Parcella fand dort anschließend statt)

Stuttgart, 9.-11. Oktober 1995 (Mitteilungen Nr. 14)

Jülich, 10.-12. April 1996, "Parallel Systems and Algorithms" (4. PASA-Workshop), Tagungsband erschienen bei World Scientific 1997)

Bad Honnef, 13.-15. Mai 1996, zusammen mit der GI-Fachgruppe 2.1.4 'Alternative Konzepte für Sprachen und Rechner' (Mitteilungen Nr. 15)

Rostock, (Warnemünde) 11. September 1997 (Mitteilungen Nr. 16, im Rahmen der ARCS'97 vom 8.-11. September 1997)

Karlsruhe, 16.-17. September 1998 (Mitteilungen Nr. 17)

Jena, 7. September 1999, "Parallele Systeme und Algorithmen" (5. PASA-Workshop im Rahmen der ARCS'99)

An Stelle eines Workshop-Bandes wurde den PARS-Mitgliedern im Januar 2000 das Buch 'SCI: Scalable Coherent Interface, Architecture and Software for High-Performance Compute Clusters', Hermann Hellwagner und Alexander Reinefeld (Eds.) zur Verfügung gestellt.

München, 8.-9. Oktober 2001 (Mitteilungen Nr. 18)

Karlsruhe, 11. April 2002, "Parallele Systeme und Algorithmen" (Mitteilungen Nr. 19, 6. PASA-Workshop im Rahmen der ARCS 2002)

Travemünde, 5./6. Juli 2002, Brainstorming Workshop "Future Trends" (Thesen in Mitteilungen Nr. 19)

Basel, 20./21. März 2003 (Mitteilungen Nr. 20)

Augsburg, 26. März 2004 (Mitteilungen Nr. 21)

Lübeck, 23./24. Juni 2005 (Mitteilungen Nr. 22)

Frankfurt/Main, 16. März 2006 (Mitteilungen Nr. 23)

Hamburg, 31. Mai / 1. Juni 2007 (Mitteilungen Nr. 24)

Dresden, 26. Februar 2008 (Mitteilungen Nr. 25)

Parsberg, 4./5. Juni 2009 (Mitteilungen Nr. 26)

Hannover, 23. Februar 2010 (Mitteilungen Nr. 27)

Rüschlikon, 26./27. Mai 2011 (Mitteilungen Nr. 28)

München, 29. Februar 2012 (Mitteilungen Nr. 29)

Erlangen, 11.+12. April 2013 (Mitteilungen Nr. 30)

Lübeck, 25. Februar 2014 (Mitteilungen Nr. 31)

Potsdam, 7.+8. Mai 2015 (Mitteilungen Nr. 32)

Nürnberg, 4.+5. April 2016 (Mitteilungen Nr. 33)

4. Mitteilungen (ISSN 0177-0454)

Bisher sind 33 Mitteilungen zur Veröffentlichung der PARS-Aktivitäten und verschiedener Workshops erschienen. Darüberhinaus enthalten die Mitteilungen Kurzberichte der Mitglieder und Hinweise von allgemeinem Interesse, die dem Sprecher zugetragen werden.

Teilen Sie - soweit das nicht schon geschehen ist - Tel., Fax und E-Mail-Adresse der GI-Geschäftsstelle mitgliederservice@gi-ev.de mit für die zentrale Datenerfassung und die regelmäßige Übernahme in die PARS-Mitgliederliste. Das verbessert unsere Kommunikationsmöglichkeiten untereinander wesentlich.

5. Vereinbarung

Die Gesellschaft für Informatik (GI) und die Informationstechnische Gesellschaft im VDE (ITG) vereinbaren die Gründung einer gemeinsamen Fachgruppe

Parallel-Algorithmen, -Rechnerstrukturen und -Systemsoftware,

die den GI-Fachausschüssen bzw. Fachbereichen:

FA 0.1	Theorie der Parallelverarbeitung
FA 3.1	Systemarchitektur
FB 4	Informationstechnik und technische Nutzung der Informatik

und den ITG-Fachausschüssen:

FA 4.1	Rechner- und Systemarchitektur
FA 4.2/3	System- und Anwendungssoftware

zugeordnet ist.

Die Gründung der gemeinsamen Fachgruppe hat das Ziel,

- die Kräfte beider Gesellschaften auf dem genannten Fachgebiet zusammenzulegen,
- interessierte Fachleute möglichst unmittelbar die Arbeit der Gesellschaften auf diesem Gebiet gestalten zu lassen,
- für die internationale Zusammenarbeit eine deutsche Partnergruppe zu haben.

Die fachliche Zielsetzung der Fachgruppe umfasst alle Formen der Parallelität wie

- Nebenläufigkeit
- Pipelining
- Assoziativität
- Systolik
- Datenfluss
- Reduktion

  etc.

und wird durch die untenstehenden Aspekte und deren vielschichtige Wechselwirkungen umrissen. Dabei wird davon ausgegangen, dass in jedem der angegebenen Bereiche die theoretische Fundierung und Betrachtung der Wechselwirkungen in der Systemarchitektur eingeschlossen ist, so dass ein gesonderter Punkt „Theorie der Parallelverarbeitung“ entfällt.

1. Parallelrechner-Algorithmen und -Anwendungen

   - architekturabhängig, architekturunabhängig
   - numerische und nichtnumerische Algorithmen
   - Spezifikation
   - Verifikation
   - Komplexität
   - Implementierung

2. Parallelrechner-Software

   - Programmiersprachen und ihre Compiler
   - Programmierwerkzeuge
   - Betriebssysteme

3. Parallelrechner-Architekturen

   - Ausführungsmodelle
   - Verbindungsstrukturen
   - Verarbeitungselemente
   - Speicherstrukturen
   - Peripheriestrukturen

4. Parallelrechner-Modellierung, -Leistungsanalyse und -Bewertung

5. Parallelrechner-Klassifikation, Taxonomien

Als Gründungsmitglieder werden bestellt:

von der GI: Prof. Dr. A. Bode, Prof. Dr. W. Gentzsch, R. Kober, Prof. Dr. E. Mayr, Dr. K. D. Reinartz, Prof. Dr. P. P. Spies, Prof. Dr. W. Händler

von der ITG: Prof. Dr. R. Hoffmann, Prof. Dr. P. Müller-Stoy, Dr. T. Schwederski, Prof. Dr. Swoboda, G. Valdorf

Ordnung der Fachgruppe
Parallel-Algorithmen, -Rechnerstrukturen und -Systemsoftware

1. Die Fachgruppe wird gemeinsam von den Fachausschüssen 0.1, 3.1 sowie dem Fachbereich 4 der Gesellschaft für Informatik (GI) und von den Fachausschüssen 4.1 und 4.2/3 der Informationstechnischen Gesellschaft (ITG) geführt.

2. Der Fachgruppe kann jedes interessierte Mitglied der beteiligten Gesellschaften beitreten. Die Fachgruppe kann in Ausnahmefällen auch fachlich Interessierte aufnehmen, die nicht Mitglied einer der beteiligten Gesellschaften sind. Mitglieder der FG 3.1.2 der GI und der ITG-Fachgruppe 6.1.2 werden automatisch Mitglieder der gemeinsamen Fachgruppe PARS.

3. Die Fachgruppe wird von einem ca. zehnköpfigen Leitungsgremium geleitet, das sich paritätisch aus Mitgliedern der beteiligten Gesellschaften zusammensetzen soll. Für jede Gesellschaft bestimmt deren Fachbereich (FB 3 der GI und FB 4 der ITG) drei Mitglieder des Leitungsgremiums: die übrigen werden durch die Mitglieder der Fachgruppe gewählt. Die Wahl- und die Berufungsvorschläge macht das Leitungsgremium der Fachgruppe. Die Amtszeit der Mitglieder des Leitungsgremiums beträgt vier Jahre. Wiederwahl ist zulässig.

4. Das Leitungsgremium wählt aus seiner Mitte einen Sprecher und dessen Stellvertreter für die Dauer von zwei Jahren; dabei sollen beide Gesellschaften vertreten sein. Wiederwahl ist zulässig. Der Sprecher führt die Geschäfte der Fachgruppe, wobei er an Beschlüsse des Leitungsgremiums gebunden ist. Der Sprecher besorgt die erforderlichen Wahlen und amtiert bis zur Wahl eines neuen Sprechers.

5. Die Fachgruppe handelt im gegenseitigen Einvernehmen mit den genannten Fachausschüssen. Die Fachgruppe informiert die genannten Fachausschüsse rechtzeitig über ihre geplanten Aktivitäten. Ebenso informieren die Fachausschüsse die Fachgruppe und die anderen beteiligten Fachausschüsse über Planungen, die das genannte Fachgebiet betreffen. Die Fachausschüsse unterstützen die Fachgruppe beim Aufbau einer internationalen Zusammenarbeit und stellen ihr in angemessenem Umfang ihre Publikationsmöglichkeiten zur Verfügung. Die Fachgruppe kann keine die Trägergesellschaften verpflichtenden Erklärungen abgeben.

6. Veranstaltungen (Tagungen/Workshops usw.) sollten abwechselnd von den Gesellschaften organisiert werden. Kostengesichtspunkte sind dabei zu berücksichtigen.

7. Veröffentlichungen, die über die Fachgruppenmitteilungen hinausgehen, z. B. Tagungsberichte, sollten in Abstimmung mit den den Gesellschaften verbundenen Verlagen herausgegeben werden. Bei den Veröffentlichungen soll ein durchgehend einheitliches Erscheinungsbild angestrebt werden.

8. Die gemeinsame Fachgruppe kann durch einseitige Erklärung einer der beteiligten Gesellschaften aufgelöst werden. Die Ordnung tritt mit dem Datum der Unterschrift unter die Vereinbarung über die gemeinsame Fachgruppe in Kraft.

ITG

ARCS 2017

30TH GI/ITG INTERNATIONAL CONFERENCE ON ARCHITECTURE OF COMPUTING SYSTEMS
THIS YEAR'S FOCUS: HETEROGENEOUS NODE ARCHITECTURES WITH DEEP MEMORY SYSTEMS

Vienna, Austria
April 03 - 06, 2017
http://arcs2017.itec.kit.edu/

CALL FOR PAPERS

Submission Deadline: October 28, 2016

The ARCS conferences series has over 30 years of tradition reporting leading edge research in computer architecture and operating systems. The focus of the 2017 conference will be on Heterogeneous Node Architectures with Deep Memory Systems.

ARCS 2017 will be organized by the Complang Group at the Vienna University of Technology and the CAPP group at the Karlsruhe Institute of Technology (KIT).

The proceedings of ARCS 2017 will be published in the Springer Lecture Notes on Computer Science (LNCS) series. After the conference, it is planned that authors of selected papers will be invited to submit an extended version of their contribution for publication in a special issue of the Journal of Systems Architecture. Further, a best paper and best presentation award will be presented at the conference.

Paper submission: Authors are invited to submit original, unpublished research papers on one or more of the following topics:

- Multi-/many-core architectures, memory systems, and interconnect networks.
- Programming models, runtime systems, and middleware support for many-core and/or heterogeneous computing platforms.
- Tool support for performance optimization, debugging, and verification.
- Generic and application-specific architectures such as reconfigurable systems in hardware and software.
- Robust and fault-tolerant systems structures.
- Architectures and design methods/tools for real-time embedded systems.
- Cyber-physical systems and distributed computing architectures.
- Organic and autonomic computing including both theoretical and practical results on self-organization, self-configuration, self-optimization, self-healing, and self-protection techniques.
- Operating Systems, including but not limited to scheduling, memory management, power management, and real-time OS (RTOS) concepts.
- Energy and power-aware computing, including green computing topics.
- System aspects of ubiquitous and pervasive computing such as sensor nodes, novel input/output devices, novel computing platforms, architecture modeling, and middleware.
- Architectures for robotics and automation systems.
- Applications of embedded and cyber-physical systems.
- High-performance and large scale parallel computing.
- Approximate computing.
- Post-Moore Architectures, including but not limited to quantum and neuromorphic computing.

Submissions should be done through the link that is provided on the conference website https://easychair.org/conferences/?conf=arcs2017. Papers must be submitted in PDF format.

They should be formatted according to Springer LNCS style (see: http://www.springer.de/comp/lncs/authors.html) and must not exceed 12 pages, including references and figures.

Workshop and Tutorial Proposals: Proposals for workshops and tutorials within the technical scope of the conference are solicited. Submissions should be done through email directly to the corresponding chair: Carsten Trinitis, (Carsten.Trinitis@tum.de)

Important Dates:

Paper submission deadline:	October 28, 2016
Workshop and tutorial proposals:	November 30, 2016
Notification of acceptance:	December 21, 2016
Camera-ready papers:	January 11, 2017

Organizing Committee:

General Co-Chairs
Jens Knoop, Vienna University of Technology, Austria
Wolfgang Karl, Karlsruhe Institute of Technology, Germany

Program Co-Chairs
Martin Schulz, Lawrence Livermore National Laboratory, USA
Koji Inoue, Kyushu University, Japan

Workshop and Tutorial Co-Chairs
Carsten Trinitis, Technische Universität München, Germany

Publicity Chair:
Miquel Pericàs, Chalmers University of Technology, Sweden

Publication Chair
Thilo Piontek, Magdeburg University, Germany

Local Organization
TPD

Program Committee (to be completed):
Michael Beigl, Karlsruhe Institute of Technology, Germany
Mladen Berekovic, TU Braunschweig, Germany
Jürgen Brehm, Leibniz University Hannover, Germany
Uwe Brinkschulte, University of Frankfurt/Main, Germany
João Cardoso, FEUP/University of Porto, Portugal
Laura Carrington, San Diego Supercomputing Center, USA
Albert Cohen, INRIA, France
Martin Daněk, TU Darmstadt, Germany
Ahmed El-Mahdy, Alexandria University, Egypt
Dietmar Fey, University of Erlangen-Nuremberg, Germany
William Fornaciari, Politecnico di Milano, Italy
Roberto Giorgi, University of Siena, Italy
Daniel Gracia-Pérez, Thales Research & Technology, France
Jan Haase, UniversitätLübeck, Germany
Andreas Herkersdorf, TUMünchen, Germany
Christian Hochberger, TU Darmstadt, Germany
Gert Jervan, Tallinn University of Technology, Estland
Jörg Keller, Fernuniversität Hagen, Germany
Andreas Koch, TU Darmstadt, Germany
Hana Kubátová, FIT CTU, Prague, Czech Republic

Olaf Landsiedel, Chalmers University of Technology, Sweden
Dong Li, UC Merced, USA
Erik Maehle, Universität zu Lübeck, Germany
Christian Müller-Schloer, Leibniz University Hannover, Germany
Luis Pinho, CISTER, ISEP, Portugal
Thilo Pionteck, Universität zu Lübeck, Germany
Pascal Sainrat, IRIT - Université de Toulouse, France
Luca Santinelli, Onera, France
Toshinori Sato, Fukuoka University, Japan
Wolfgang Schröder-Preikschat, FAU, Germany
Muhammad Shafique, Karlsruhe Institute of Technology, Germany
Cristina Silvano, Politecnico di Milano, Italy
Leonel Sousa, IST/INESC-ID, Portugal
Rainer G. Spallek, TU Dresden , Germany
Olaf Spinczyk, TU Dortmund, Germany
Benno Stabernack, Fraunhofer HHI, Germany
Walter Stechele, TU Munich, Germany
Jürgen Teich, University of Erlangen-Nuremberg, Germany
Sven Tomforde, University of Kassel, Germany
Carsten Trinitis, TU Munich, Germany
Hans Vandierendonck, Queen's University Belfast, Great Britain
Stephane Vialle, SUPELEC, France
Lucian Vintan**,** "Lucian Blaga" University of Sibiu, Romania
Klaus Waldschmidt ,University of Frankfurt, Germany
Stephan Wong, Delft University of Technology, The Netherlands
Sungjoo Yoo, Seoul National University, Korea

GESELLSCHAFT FÜR INFORMATIK E.V.

PARALLEL-ALGORITHMEN, -RECHNERSTRUKTUREN UND -SYSTEMSOFTWARE

PARS

ITG INFORMATIONSTECHNISCHE GESELLSCHAFT IM VDE

CALL FOR PAPERS

27th PARS Workshop on May 4+5, 2017

Hagen

http://fg-pars.gi.de/workshops/pars-workshop-2017/

PARS is the special interest group on parallel algorithms, parallel computer structures and parallel system software within the German Informatics Societies (GI/ITG). The goal of the bi-annual PARS Workshop is the presentation of important research within the scope of PARS and an exchange of ideas between the participants. Topics of interest are:

- **Parallel Algorithms (Presentation, Complexity, Applications)**
- **Parallel Models of Computation and Parallel Architectures**
- **Parallel Programming Languages and Libraries**
- **Tools for Parallelization (Compilers, Performance Analysis, Auto-Tuner)**
- **Parallel Embedded Systems / Cyber-Physical Systems**
- **Software Engineering for Parallel and Distributed Systems**
- **Multicore-, Manycore-, GPGPU-Computing and Heterogeneous Architectures**
- **Cluster Computing, Grid Computing, Cloud Computing**
- **Interconnect and Hardware Structures (e.g. reconfigurable systems)**
- **Future Technologies and new Computational Paradigms for Architectures (SoC, PIM, STM, Memristor, DNA-Computing, Quantum Computing)**
- **Teaching Parallel Computation (Experiences, E-Learning)**
- **Parallel and Distributed Computing in Life Sciences (e.g. Bio-Informatics, Medical Informatics)**

Workshop languages are German and Englisch. Papers are limited to 10 pages. The workshop proceedings will be published in the yearly newsletter of the PARS special interest group (PARS-Mitteilungen, ISSN 0177-0454). The Workshop fee will be about 100 €.

Important Dates: Papers of at most 10 pages (Format: GI Lecture Notes in Informatics, previously unpublished) are to be submitted electronically until **March 1, 2017** via:
http://www.easychair.org/conferences/?conf=pars2017

Author notification: **April 1, 2017**
Submission of camera-ready papers: **August 31, 2017** (after Workshop)

Program Committee: *A. Döring, Zürich • N. Eicker, Jülich • T. Fahringer, Innsbruck • D. Fey, Erlangen V. Heuveline, Heidelberg • R. Hoffmann, Darmstadt • B. Juurlink, Berlin • W. Karl, Karlsruhe J. Keller, Hagen • C. Lengauer, Passau • E. Maehle, Lübeck • E. W. Mayr, München W. E. Nagel, Dresden • M. Philippsen, Erlangen • K. D. Reinartz, Höchstadt • B. Schnor, Potsdam P. Sobe, Dresden • C. Trinitis, München • T. Ungerer, Augsburg • R. Wanka, Erlangen*

Student Paper Award: The best paper based on a master or PhD thesis, and presented by the thesis author, will receive the student paper award of the PARS special interest group (endowed with 500 €). Co-authors are allowed, the PhD degree should not yet be awarded at the time of paper submission. Application for the award by e-mail to the organizers at the time of paper submission.

Sponsor: GI/ITG special interest group PARS, **http://fg-pars.gi.de**

Organizers:

Prof. Dr. Wolfgang Karl, Chair for Computer Architecture and Parallel Processing
Institute of Technology, 76131 Karlsruhe, Germany
Tel.: +49-721-608-43771, Fax: +49-721-608-43962, E-Mail: karl@kit.edu

Prof. Dr. Jörg Keller, Faculty of Mathematics and Computer Science, Parallelism and VLSI Group
FernUniversität in Hagen, 58084 Hagen, Germany
Tel.: +49-2331-987-376, Fax: +49-2331-987-308, E-Mail: joerg.keller@fernuni-hagen.de

PARS-Beiträge

Studenten	5,00 €
GI-Mitglieder	7,50 €
studentische Nichtmitglieder	5,00 €
Nichtmitglieder	15,00 €
Nichtmitglieder mit Doppel-Mitgliedschaften (Beitrag wie GI-Mitglieder)	--,-- €

Leitungsgremium von GI/ITG-PARS

Dr. Andreas Döring, IBM Zürich
Prof. Dr. Norbert Eicker, FZ Jülich
Prof. Dr. Thomas Fahringer, Univ. Innsbruck
Prof. Dr. Dietmar Fey, Univ. Erlangen
Prof. Dr. Vincent Heuveline, Univ. Heidelberg
Prof. Dr. Ben Juurlink, TU Berlin
Prof. Dr. Wolfgang Karl, stellv. Sprecher, KIT
Prof. Dr. Jörg Keller, Sprecher, FernUniversität in Hagen
Prof. Dr. Christian Lengauer, Univ. Passau
Prof. Dr.-Ing. Erik Maehle, Universität zu Lübeck
Prof. Dr. Ernst W. Mayr, TU München
Prof. Dr. Wolfgang E. Nagel, TU Dresden
Dr. Karl Dieter Reinartz, Ehrenvorsitzender, Univ. Erlangen-Nürnberg
Prof. Dr. Bettina Schnor, Univ. Potsdam
Prof. Dr. Peter Sobe, HTW Dresden
Prof. Dr. Theo Ungerer, Univ. Augsburg
Prof. Dr. Rolf Wanka, Univ. Erlangen-Nürnberg

Sprecher

Prof. Dr. Jörg Keller
FernUniversität in Hagen
Fakultät für Mathematik und Informatik
Lehrgebiet Parallelität und VLSI
Universitätsstraße 1
58084 Hagen
Tel.: (02331) 987-376
Fax: (02331) 987-308
E-Mail: joerg.keller@fernuni-hagen.de
URL: http://fg-pars.gi.de/