GESELLSCHAFT FÜR INFORMATIK E.V.

PARALLEL-ALGORITHMEN, -RECHNERSTRUKTUREN UND -SYSTEMSOFTWARE

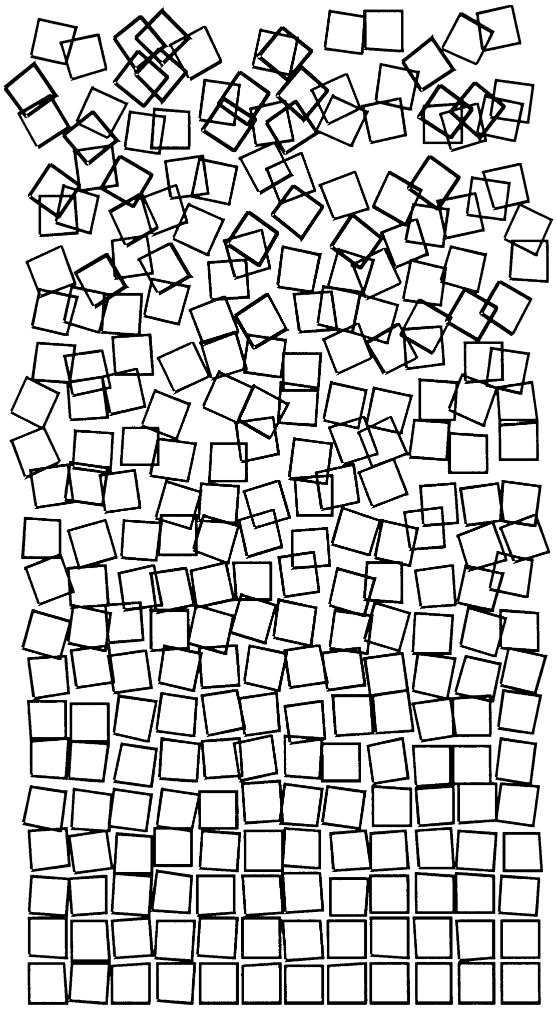
PARS

INFORMATIONSTECHNISCHE GESELLSCHAFT IM VDE

I n h a l t

Aktuelle PARS-Aktivitäten unter:

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



Computergraphik von: Georg Nees, Generative Computergraphik

MITTEILUNGEN Nr. 32, September 2015 (Workshop 2015)

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.

GESELLSCHAFT FÜR INFORMATIK E.V.

PARALLEL-ALGORITHMEN, -RECHNERSTRUKTUREN UND -SYSTEMSOFTWARE

PARS

INFORMATIONSTECHNISCHE GESELLSCHAFT IM VDE

CALL FOR PAPERS

26. PARS - Workshop am 7.-8. Mai 2015

Potsdam

http://fg-pars.gi.de/workshops/pars-workshop-2015/

Ziel des PARS-Workshops ist die Vorstellung wesentlicher Aktivitäten im Arbeitsbereich von PARS und ein damit verbundener Gedankenaustausch. Mögliche Themenbereiche sind:

- **Parallele Algorithmen (Beschreibung, Komplexität, Anwendungen)**
- **Parallele Rechenmodelle und parallele Architekturen**
- **Parallele Programmiersprachen und Bibliotheken**
- **Werkzeuge der Parallelisierung (Compiler, Leistungsanalyse, Auto-Tuner)**
- **Parallele eingebettete Systeme / Cyber-Physical Systems**
- **Software Engineering für parallele und verteilte Systeme**
- **Multicore-, Manycore-, GPGPU-Computing und Heterogene Architekturen**
- **Cluster Computing, Grid Computing, Cloud Computing**
- **Verbindungsstrukturen und Hardwareaspekte (z. B. rekonfigurierbare Systeme)**
- **Zukünftige Technologien und neue Berechnungsparadigma für Architekturen (SoC, PIM, STM, Memristor, DNA-Computing, Quantencomputing)**
- **Parallelverarbeitung im Unterricht (Erfahrungen, E-Learning)**
- **Methoden des parallelen und verteilten Rechnens in den Life Sciences (z.B.Bio-, Medizininformatik)**

Die Sprache des Workshops ist Deutsch und Englisch. Für jeden Beitrag sind maximal 10 Seiten vorgesehen. Die Workshop-Beiträge werden als PARS-Mitteilungen (ISSN 0177-0454) publiziert. Es ist eine Workshopgebühr von ca. 100 € geplant.

Termine: Beiträge im Umfang von 10 Seiten (Format: GI Lecture Notes in Informatics, nicht vor-veröffentlicht) sind bis zum **16. März 2015** in elektronischer Form unter folgendem Link einzureichen:
http://www.easychair.org/conferences/?conf=pars2015

Benachrichtigung der Autoren bis **13. April 2015**
Druckfertige Ausarbeitungen bis **31. August 2015** (nach dem Workshop)

Programmkomitee: *H. Burkhart, Basel • S. Christgau, Potsdam • A. Döring, Zürich • T. Fahringer, Innsbruck • D. Fey, Erlangen W. Heenes, Darmstadt • V. Heuveline, Heidelberg • R. Hoffmann, Darmstadt • W. Karl, Karlsruhe J. Keller, Hagen • C. Lengauer, Passau • E. Maehle, Lübeck • E. W. Mayr, München • F. Meyer auf der Heide, Paderborn • W. E. Nagel, Dresden • M. Philippsen, Erlangen • K. D. Reinartz, Höchstadt H. Schmeck, Karlsruhe • B. Schnor, Potsdam • P. Sobe, Dresden • T. Ungerer, Augsburg • R. Wanka, Erlangen • H. Weberpals, Hamburg*

Nachwuchspreis: Der beste Beitrag, der auf einer Diplom-/Masterarbeit oder Dissertation basiert, und von dem Autor/der Autorin selbst vorgetragen wird, wird auf dem Workshop von der Fachgruppe PARS mit einem Preis (dotiert mit 500 €) ausgezeichnet. Co-Autoren sind erlaubt, der Doktorgrad sollte zum Zeitpunkt der Einreichung noch nicht verliehen sein. Die Bewerbung um den Preis erfolgt durch E-Mail an die Organisatoren bei Einreichung des Beitrages.

Veranstalter: GI/ITG-Fachgruppe PARS, **http://fg-pars.gi.de**

Organisation:

Prof. Dr. Bettina Schnor, Institut für Informatik
Universität Potsdam, 14482 Potsdam, Germany
Tel.: +49-331-977-3120, Fax: +49-331-977-3122, E-Mail: schnor@cs.uni-potsdam.de

Prof. Dr. Jörg Keller (PARS-Sprecher), Fakultät für Mathematik und Informatik, LG Parallelität und VLSI
FernUniversität in Hagen, 58084 Hagen, Germany
Tel.: +49-2331-987-376, Fax: +49-2331-987-308, E-Mail: joerg.keller@fernuni-hagen.de

26. PARS-Workshop (Full Papers)

Novel Image Processing Architecture for 3D Integrated Circuits

Benjamin Pfundt[1], Marc Reichenbach[1], Christopher Söll[2], Dietmar Fey[1]

[1]Chair of Computer Architecture
Department of Computer Science
[2]Institute for Electronics Engineering
Department of Electrical, Electronic and Communication Engineering
Friedrich-Alexander-University Erlangen-Nürnberg (FAU), Germany
{benjamin.pfundt, marc.reichenbach, christopher.soell, dietmar.fey}@fau.de

Abstract: Utilizing highly parallel processors for high speed embedded image processing is a well known approach. However, the question of how to provide a sufficiently fast data rate from image sensor to processing unit is still not solved. As Trough-Silicon-Vias (TSV), a new technology for chip stacking, become available, parallel image transmission from the image sensor to processing unit is enabled. Nevertheless, the usage of a new technology requires architectural changes in the processing units. With this technology at hand, we present a novel image preprocessing architecture suitable for image processing in 3D chips stacks. The architecture was developed in parallel with a customized image sensor to make a real assembly possible. It is fully functionally verified and layouted for a 150 nm process. Our performance estimation shows a processing speed of 770 up to 14.400 fps (frames per second) for 5×5 filters.

1 Introduction

Due to the continuously rising performance requirements in image processing systems, novel approaches in architecture design are desperately needed. One solution to fulfill these requirements is the processing or at least preprocessing of the captured image near to the sensor. For that reason, image sensor and processing unit will be connected together, which is the idea behind *smart cameras*. Due to the fact that image processing algorithms are generally easily parallelizable, a high performance can be achieved in the domain of *smart cameras* with a well designed parallel processing architecture. Nevertheless, a common problem with high speed data acquisition frequently occurs: while capturing and processing of the image can be executed in parallel, the data link in between is mostly designed using serial links. This slows down the processing and limits the possible degree of parallelism and therefore performance in the processing architecture.

To overcome this issue, a paradigm shift from smart cameras to *smart sensors* is needed. This can be achieved by integrating processing structures in or very close to the sensor. A straightforward implementation is to construct a SIMD array of processing elements

(PEs) and assign it to one or more pixel cells. Especially local processing algorithms profit from these fine grained processor arrays because data exchange to and from neighboring elements only requires additional wires in the simplest case. Also, specialized high speed and resource consuming transmission logic for high volume raw sensor data can be dropped. Still, a low latency is achieved as sensor data is directly read by the processing elements. Due to the massively parallel transmission, a high bandwidth is possible if all processing elements are considered while the elements themselves could have a low processing frequency. Furthermore, a large on-chip storage can be omitted as processing elements only operate on few pixels.

Though these apparent advantages, major drawbacks arise at the IC design level. If an array of elements consisting of photo diodes and processing logic is created, only a very low fill factor can be achieved. Due to the extra size of the processing logic and analog to digital converter, the pixel size strongly increases. This results in a low sensor resolution and limits the practical use. Solving this problem by splitting pixel and processing leads to other drawbacks, e.g. a massive increase in wiring complexity or a large footprint.

A promising approach to bypass planar layout problems is vertical chip stacking. Several chips with different functions are stacked upon each other and are connected by a multitude of through silicon vias (TSVs). Figure 1 illustrates an example stack: photo diodes, ADCs, processing logic and memory could be placed on separate layers. The result is a smart image sensor chip stack with a much smaller footprint compared to a planar design, yet offering the possibility to increase the bandwidth between pixel and processing array. The interconnect length decreases while a large number of connections can be implemented as the diameter of TSVs can be as small as $1\mu m$ [Tor13]. Furthermore, chips from heterogeneous technologies can be stacked and troublesome mixed signal designs can be avoided. The possibilities of 3D chip stacking were recognized early on. First concrete ideas for processing schemes [Tan85] and also simple stacked IC designs [NIS$^+$87] are nearly as old as monolithic chip designs of sensor and processing logic.

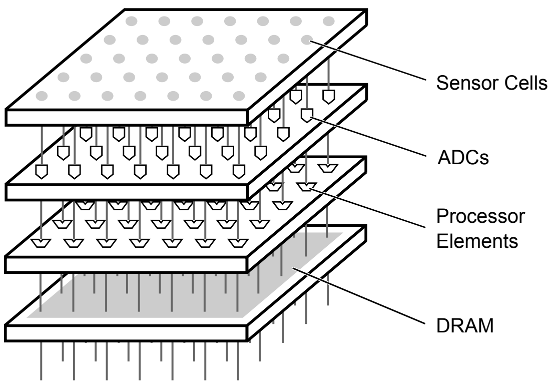


Figure 1: Heterogenous Chip Stack with Massive Parallel Transmission

Although, the benefits of 3D ICs for processing raw image data close to the sensor are at hand, a digital image processing architecture which harnesses this potential is not straightforward. We therefore propose a parallel image preprocessing architecture in this paper which can be connected via TSVs to an image sensor. This image sensor is currently developed in close cooperation with our partners at LTE [SSB$^+$15]. The architecture is functionally verified and completely layouted for productive use.

Over the years, different architectures have been proposed. Hence, we will first provide a short overview about recent designs in the next section. Based on the shortcomings, we will then develop a novel processing scheme for image preprocessing in Section 3. Our goal is to outweigh performance and resource usage to design an efficient low power smart sensor for embedded purposes. The actual realization as ASIC is presented in Section 4 and the results are discussed afterwards. Finally, Section 6 offers a conclusion and addresses further enhancements.

2 Related Work

The topic of integrating photo sensor and processing capabilities into a single chip has been investigated for decades and many designs have been proposed. Among the early monolithic chips are designs based on cellular automata. In [GZD85] an 8×8 array of elementary processors easily extendable to 256×256 elements was proposed. Each elementary processor was assigned to one photodiode and could perform combinatorial operations on its and the neighboring pixel's values. Also in optoelectronic processing close to the sensor has been put forward. In this domain the term *smart pixel* was coined to describe a hybrid design of optical devices, e.g. photo diodes, and electronics for processing [Hin88].

The designs of image processors for vertical integration generally split into three categories. One approach is similar to earlier smart pixels or cellular automata designs and uses pixel parallel processing arrays [RVCGFB$^+$14, LD11]. In addition to this, there exist designs with larger processing units. These units are assigned to a large portion of pixels or can even work on the whole image [DFJAM14, CHF$^+$12]. Additionally, a combination of both approaches has been suggested in [SBP$^+$12].

With respect to architectural complexity, the easiest way to design 3D image processors is to use a SIMD array with a processing element for each pixel. In-pixel ADCs on a different layer than the photo diodes have been successfully manufactured in [GHI$^+$14], however the resolution of 64×64 was rather low. A multiple layer chip stack is proposed in [LD11] offering cellular automata operation for 128×96 pixels. Although the digital processing part has been split on two layers of $25mm^2$ each, the achievable fill factor is still low.

If a large portion of pixels is to be processed by a single or a couple of processing units, one main advantage of processing close to the photo diodes diminishes: the exchange of values between neighboring pixel areas cannot be achieved without storing the whole or portions of the image. In [DFJAM14] a 48×32 sensor array is introduced where a column-wise computation takes place. No data exchange between array elements is possible and the operations are very limited. A much more complex design has been proposed in [CHF$^+$12] where a large multi-core chip of $63mm^2$ accommodates eight RISC processors. The digital layer should be connected to a partitioned layer of ADCs which in turn should be connected to a tiled image sensor with a resolution of 2048×1536. The introduced digital layer has two SDRAM controllers for external memory. A large memory is needed to calculate even simple neighborhood operations like 2D filters or stencil codes. Most probably, only a fraction of the actual computational power can be obtained for bandwidth bound problems.

Both extremes, pixel-parallel and large scale computation, have their disadvantages. Therefore, we pursue an image preprocessing architecture for 3D stacking which provides a balanced mixture of parallel computation and chip utilization as well as resource utilization.

3 Architectural Conception

In this section we introduce a fine grained parallel architecture which provides data exchange between sensor elements at a minimum of additional resources in form of specialized buffer structures.

3.1 Overall Layout

Vertical interconnections between different IC layers influence the coupling and also the architectural layout of each layer. The smart pixel and pixel parallel approaches had one ADC per photo diode. If more photo diodes shared one ADC, a homogeneous distribution would not be possible. For the application domain of cellular automata and image filters, a one-to-one ratio between ADCs and PEs leads to a simple logic layer. For every pixel and its neighbors the respective operation has to be carried out. If more pixel are feed into one PE, the exchange and storage becomes more complex. Although a one-to-one ratio is straightforward, the main drawback is the space consuming ADC which eventually causes low fill factors. Therefore, the goal has to be to reduce the number of ADCs and use a more traditional approach where photo cells are read out row by row and column by column. The number of ADCs can be increased if a couple of pixels per row are read out simultaneously. This can be achieved if the output of the column multiplexer of an off-the-shelf CMOS image sensor is enlarged as Figure 2 illustrates. The fill factor is not changed, as the ADCs are not located inside the pixel cells.

3.2 Partitioning

High sensor resolutions require many ADCs to achieve decent frame rates. For parallel mask operations, pixels of coherent image regions have to be converted. This has two main disadvantages for high resolutions. First of all, neighboring pixel cells have to be converted simultaneously and therefore connected to different ADCs. This increases the

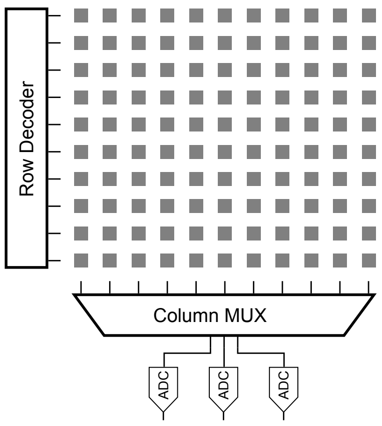


Figure 2: Image Sensor with Parallel ADCs

wiring complexity dramatically for a large number of ADCs. Secondly, the number of PEs has to be adjusted to the number of ADCs. This leads to many stores and loads as previous pixel cells have to be temporarily stored if they should be reused again. To cope with this problem, we propose a partitioning scheme where the sensor is split into rectangular tiles. Each tile has its ADCs and is connected to an array of PEs. The number of ADCs and PEs is decreased with the lower resolution per partition. This greatly relieves the wiring complexity and limits the local memory traffic, while the overall frame rate remains constant.

As 2D filter operations also include neighboring pixels, communication has to take place across partition borders. Due to local masks requiring only a small image region, just a portion of the partition's pixel data has to be held in memory. Therefore, we propose a partitioning sequence which is depicted in Figure 3. The pixel cell read out starts in the middle of the image sensor and proceeds to the opposite end of the partition either meandering or line by line. The starting point could also be at the corners of the image sensor as long as the read out proceeds similarly in each partition. The current pixel values are held inside local buffers. Due to the processing order, it is ensured that PEs at partition border can access data elements from other partition as they are currently held in the buffers of an other PE array. All PE arrays can be directly exchange the appropriate data.

A further advantage results from the partitioning scheme. Besides the configuration possibilities, the partitions can be reused. The system becomes easily scalable if the constraints for a new design are changed. Thus, our architecture is highly configurable and can be exactly adjusted to application and image sensor constraints. The result is a light-weight and balanced system which efficiently employs the resources used.

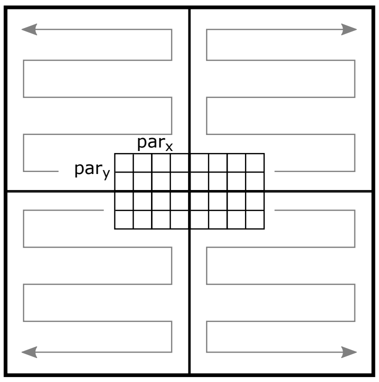


Figure 3: Processing Scheme for Synchronous Data Exchange Between Partitions

3.3 Processing Scheme

As the rows of the sensor are read out successively, pixel values have to be stored to allow operations which need the neighboring pixel values. If only a CPU is available, the whole image is commonly stored in a RAM utilizing double buffering. This results in a high power consumption due to a large RAM and access is slowed down by additional latencies. A more resource efficient approach is to process the image data on the fly while it is streamed out of the image sensor. With a CPU this could be achieved utilizing circular buffer structures. For our target applications, e.g. 2D filters, an even more light-weight custom implementation is possible which will be presented in the next paragraphs.

On-the-fly processing of 2D filters and other mask operators can be efficiently realized utilizing line buffers in a full buffering scheme. A scalable full buffering architecture for FPGAs was presented in [SRF12]. A processing scheme for 3D chip stacking can be devised similarly. The basic structure for 3×3 masks is illustrated in Figure 4. Pixel value transmission from the ADCs goes directly into registers. The array of registers holds all data elements which are needed to carry out the operations by PEs in parallel. Larger storage elements, e.g. SRAM blocks, are used as line buffers to store exactly the number of previous pixel values which are needed for further calculations. After the PEs have finished their calculation, a line buffer behaves like a FIFO. Newly received pixel values will replace older elements which are in turn feed into the PE registers.

Parallelism for full buffering structures can be increased in two ways. On the one hand, the number of PEs can be increased which is is limited by the number of possible ADCs. As solution a demultiplexer could be introduced after the ADCs to serve more PEs. Then, the one-to-one relation between ADCs and PEs had to be modified according to the respective operation frequencies. On the other hand, parallelism can be implemented by building several stages of full buffering structures. The parallel output of one structure will then be feed into the next one. Depending on the application, the number of PEs per stage can vary as long as a synchronization mechanism between the stages is applied.

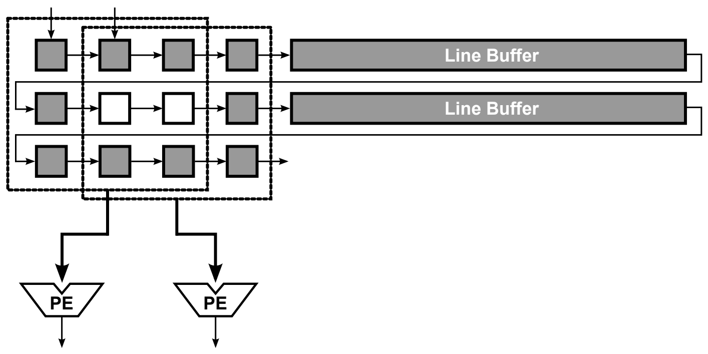


Figure 4: Full Buffering Structure with Dual PE

4 Implementation

The architectural concept has been implemented in VHDL to produce a real chip stack. In cooperation with our university partner, we developed the constraints for a two layered smart sensor. The final sensor chip will consist of a partitioned photo sensor with a resolution of 216×216. Nine pixels per partition can be accessed in parallel and are feed into an analog processing unit for basic 3×3 filter masks [SSB$^+$15]. The uneven sensor resolution is a multiple of the parallel accessible pixels. In the final design either 9 parallel 16 bit ADCs will transform the raw pixel data or only one will convert the calculated value.

The transmission via TSVs to the digital processing part is done bit-parallel. As up to nine pixel values can be transmitted, the same number of PEs has been implemented. For the digital part a window size of 5×5 was implemented. This leads to a minimum of 144 bit of parallel in- and outputs per partition. If available, the output could be stored in a DRAM layer. Otherwise a deserializer is placed at the outputs to limit the number of pins.

4.1 Layout

For VHDL synthesis we used *Design Compiler*® from Synopsys. The IC layout has been generated with *Encounter*® from Cadence. As the design tool support for 3D ICs is still not mature, the tool support for ball grid arrays (BGAs) is used as workaround [Tor13]. The BGAs is placed and assigned in at the place and route step. In a further layout step the actual TSV cells are placed. We passed through the design flow with a 150 *nm* from LFoundry. The diameter of the TSV cells used is $1.2\ \mu m$ at a $10\ \mu m$ pitch. The complete design flow including the TSV assignment has been scripted and can therefore be easily rerun with different parameters.

4.2 Structure

The implemented architectural structure can be seen in Figure 5. Besides the partition unit, the architecture consists of five major building blocks which will be described in the next paragraphs.

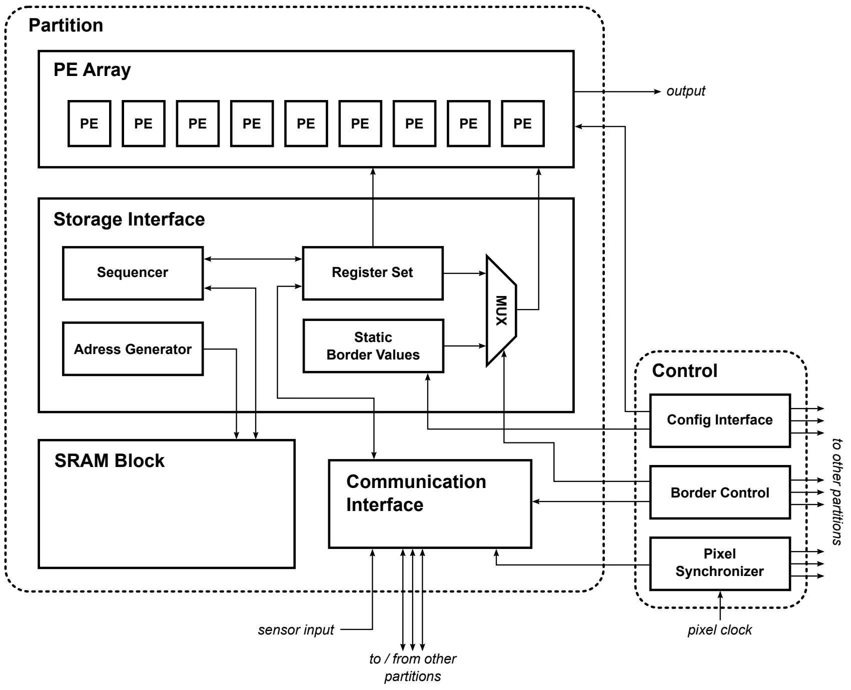


Figure 5: Final Architecture of Digital Processing Layer

Control The partitions have to be configured and controlled to work together, this is done by the separate unit *Control* which is connected to each partition. The component *Config Interface* includes the setting of the static border values needed for mask operations at the image sensor edges. Another task is to configure operation modes for the PEs. The second component *Border Control* uses an internal counter to indicate if a communication with an other partitions has to take place or if the static border values have to be used. Finally, the *Pixel Synchronizer* monitors the *pixel clock* from the sensor to indicate if new pixel values have been converted by the ADCs.

Communication Interface One main unit which finally joins the partitions is the *Communication Interface*. This interface picks up new sensor data as well as reroutes and controls the data flow from and to other partitions.

PE Array The main work is done by an array of PEs. In the current design, a single cycle mean filter is implemented for performance and comparability reasons. Thus, it is possible to compare the analog and digital implementation in size, processing speed and accuracy. Other filter or local operator can be implemented easily, as the *PE Array* interface is rather generic and provides all inputs in parallel. Similar to the pixel clock, a signal can be activated to indicate if a potential multi cycle PE calculation is finished.

SRAM Block For ASIC designs the line buffers have to be placed in SRAM blocks. To save resources we placed all line buffers of a partition in one *SRAM Block* with a single port interface. The storage requirement in bits resulting from the full buffering structure of Figure 4 can be directly calculated with Equation 1. For our design, this results in a minimum size of 6080 bits.

$$\begin{aligned} mem \geq\; & (partition\_width - \#PE - mask\_size + 1)\ \ldots \\ & \times (mask\_size - 1) \times resolution \end{aligned} \tag{1}$$

Storage Interface A more complex unit which controls the storage and provides the appropriate data for partition exchange is the *Storage Interface*. The static border values and the current registers are implemented and appropriately connected to the *PE array*. Particular registers have to be substituted, if pixels at partition edges are processed and the mask reaches across the border. A *Sequencer* and *Adress Generator* map the four line buffers of the current design to the single ported SRAM.

5 Results

The final layout of the digital IC in 150 *nm* technology is displayed in Figure 6. The SRAM cells were placed close to the power rings to provide sufficient power supply. The possible positions of the TSVs can be recognized by the uniform grid, though not every position is actually assigned. Approximately 2.5 % of the over 47000 positions were used for data pins. Further chip characteristics are shown in Table 1. The rectangular chip dimensions which does not perfectly fit to a rather quadratic image sensor is owed to the SRAM blocks. As the layer dimensions of a 3D IC do not have to fit exactly, this is no real problem. The size of the digital chip might be enlarged without affecting the functionality.

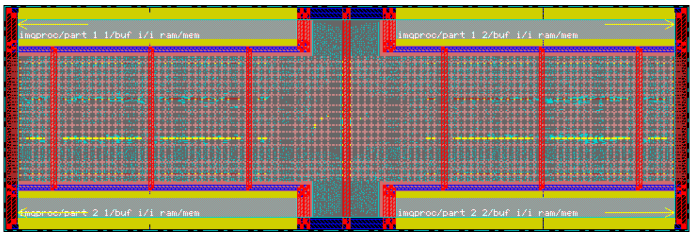

imgproc/part 1 1/buf i/i ram/mem
imgproc/part 1 2/buf i/i ram/mem
imgproc/part 2 1/buf i/i ram/mem
imgproc/part 2 2/buf i/i ram/mem

Figure 6: Layout of Chip Stacking Enabled Digital Processing ASIC

The IC runs at 40 MHz which is nearly at the limit of the SRAM blocks. If the analog part could operate at half the speed to respect an additional cycle for the *Sequencer*, a frame rate of $9\,pixels \times 20\ MHz/(108\ \times\ 108\,pixels/frame) \approx 15.430\,$*fps* would be possible. This would result in an overall bandwidth of approximately 1.44 *GByte/s*. But even for a very pessimistic pixel clock of 1 *MHz*, $770\,$*fps* without analog processing are possible. These values clearly demonstrate the practical advantages of 3D ICs for smart sensor application. With a low power consumption and moderate clock speeds, high frame rates can be achieved.

Table 1: Digital IC Properties

Property	Value
Chip Area	$1.5 \times 5.0\ mm^2$
Density	61%
Voltage	$1.8\ V$
Estimated Core Power	$60\ mW$

6 Conclusion and Outlook

Based on the technological possibilities of TSVs, chip stacking is at hand. Especially image processing architectures can benefit from these new developments to overcome the problem with serial transmissions between image sensor and processing unit. Therefore, we presented in this paper a novel image processing architecture for 3D chip stacks. The proposed design exhibits a high degree of parallelism. Firstly, subsequent pixels are processed in parallel. Secondly, due to a distribution of ADCs at the image sensor, the image is divided in four partitions for parallel processing. To avoid external memory, line buffering is used. Data exchange between partitions is implemented which enables a high flexibility for possible extensions to more partitions.

Although the chip is fully layouted and functionally verified, it still has to be manufactured and field tested. Due to rare 3D design kits, we will create a 3D chip stack prototype together with Lfoundry Srl. for a new generation of image acquisition and processing systems. In the near future, we want to connect our processing chip with the image sensor, which is developed by our colleagues.

Acknowledgements This work was financially supported by the Research Training Group 1773 “Heterogeneous Image Systems” funded by the German Research Foundation (DFG). Additionally, the authors would like to thank Andreas Bänisch for his valuable hints and comments.

References

[CHF$^+$12] C. Cheng, H. Hsieh, T. Fan, W. Tang, C. Liu, and P. Huang. High Resolution and Frame Rate Image Signal Processor Array Design for 3-D Imager. In *Interna-*

tional Symposium on Intelligent Signal Processing and Communications Systems (ISPACS), 2012, pages 735–739, Nov 2012.

[DFJAM14] M. Di Federico, P. Julian, A.G. Andreou, and P.S. Mandolesi. Fully Functional Fine-grain Vertically Integrated 3D Focal Plane Neuromorphic Processor. In *SOI-3D-Subthreshold Microelectronics Technology Unified Conference (S3S), 2014*, pages 1–2, Oct 2014.

[GHI+14] M. Goto, K. Hagiwara, Y. Iguchi, H. Ohtake, T. Saraya, M. Kobayashi, E. Higurashi, H. Toshiyoshi, and T. Hiramoto. Three-Dimensional Integrated CMOS Image Sensors with Pixel-Parallel A/D Converters Fabricated by Direct Bonding of SOI Layers. In *International Electron Devices Meeting (IEDM), 2014*, pages 4.2.1–4.2.4, Dec 2014.

[GZD85] P. Garda, B. Zavidovique, and F. Devos. Integrated Cellular Array Performing Neighborhood Combinatorial Logic on Binary Pictures. In *11th European Solid-State Circuits Conference, 1985. ESSCIRC '85.*, pages 58–63, Sept 1985.

[Hin88] H.S. Hinton. Architectural Considerations for Photonic Switching Networks. *IEEE Journal on Selected Areas in Communications*, 6(7), Aug 1988.

[LD11] A. Lopich and P. Dudek. Architecture and Design of a Programmable 3D-Integrated Cellular Processor Array for Image Processing. In *19th International Conference onVLSI and System-on-Chip (VLSI-SoC), 2011*, pages 349–353, Oct 2011.

[NIS+87] T. Nishimura, Y. Inoue, K. Sugahara, S. Kusunoki, T. Kumamoto, S. Nakagawa, M. Nakaya, Y. Horiba, and Y. Akasaka. Three Dimensional IC for High Performance Image Signal Processor. In *International Electron Devices Meeting, 1987*, volume 33, pages 111–114, 1987.

[RVCGFB+14] A. Rodriguez-Vazquez, R. Carmona-Galan, J. Fernandez Berni, S. Vargas, J.A. Lenero, M. Suarez, V. Brea, and B. Perez-Verdu. Form Factor Improvement of Smart-Pixels for Vision Sensors through 3-D Vertically-Integrated Technologies. In *Circuits and Systems (LASCAS), 2014 IEEE 5th Latin American Symposium on*, pages 1–4, Feb 2014.

[SBP+12] M. Suarez, V.M. Brea, F. Pardo, R. Carmona-Galan, and A. Rodriguez-Vazquez. A CMOS-3D Reconfigurable Architecture with In-pixel Processing for Feature Detectors. In *International 3D Systems Integration Conference (3DIC), 2011*, pages 1–8, Jan 2012.

[SRF12] M. Schmidt, M. Reichenbach, and D. Fey. A Generic VHDL Template for 2D Stencil Code Applications on FPGAs. In *15th IEEE International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing Workshops (ISORCW), 2012*, pages 180–187, April 2012.

[SSB+15] L. Shi, C. Soell, R. Baenisch, A. Weigel, J. Seiler, and T. Ussmueller. Concept for a CMOS Image Sensor Suited for Analog Image Pre-Processing. In *DATE Friday Workshop on Heterogeneous Architectures and Design Methods for Embedded Image Systems (HIS), 2015*, pages 16–21, March 2015.

[Tan85] K. Taniguchi. Three Dimensional IC's and an application to High Speed Image Processor. In *7th Symposium on Computer Arithmetic (ARITH), 1985*, pages 216–222, June 1985.

[Tor13] K. Torki. 3D-IC Integration. In *CMP annual users meeting*, Paris, Jan 2013.

A run-time reconfigurable NoC Monitoring System for performance analysis and debugging support

Erol Koser[*] and Benno Stabernack[**]

[*] Insitute for Integrated Systems, Technische Universität München
[**] Embedded Systems Group, Fraunhofer Heinrich-Hertz-Institute Berlin

Abstract:
Recently Network-on-Chip based architectures become more and more important due to their advantages in respect to design flexibility and systems bandwidth scalability since nowadays systems consists typically of a huge number of processing elements (e.g. heterogeneous multi processor systems). In contrast to typical shared memory based systems, predicting and monitoring the runtime behaviour of the system e.g. data throughput, link utilization and contention becomes more complex and requires special architectural features. Besides the traditional approach of using simulation based approaches at design time, runtime usable features promise to have a number of advantages. In this paper we present a flexible, reusable and run-time reconfigurable NoC monitoring system for performance analysis and debugging purposes. The evaluation of the monitoring data enables the system designer to achieve better resource utilization by adjusting the system architecture and the programming model.

1 Introduction

Since traditional bus systems are the critical bottleneck if more than a few master modules are attached, Networks-on-Chip (NoC) are a promising approach to solve this issue [BDM02]. Its advantages are high scalability and massive parallel communication capabilities. Even though NoCs as a communication infrastructure are well established their usage implies a number of new problems which need to be addressed to gain their full advantage. Simulations, especially HW/SW-Co-simulations, are very time consuming. A more time efficient option is to run the complete system on a prototype and trace its activity. Since NoCs do not have a central point of communication, monitoring mechanisms have to be deployed across the system. The mechanisms can be utilized at design-time and at run-time. During design-time they can be used for rapid prototyping. In case of malicious system behaviour the monitoring data can be utilized for communication debugging. Performance analyses can be executed to adjust the programming model and the system architecture. Monitoring systems are also used at run-time to support resource management functionalities of the operating system. The main components of the proposed monitoring system are probes, that are attached to communication links, and a Central Monitoring Unit (CMU). Different modes of abstraction are offered in order to collect as much data as possible. Additionally a data compression scheme is introduced, that decreases the overall amount of monitoring data without information loss.

2 Related Work

NoC monitoring is recently getting more and more attention. The majority of the systems deal with system reliability as in [MVBT09], [PVS+10] and [ZMV+11]. Sensors are placed across the chip and measure different properties like power dissipation and soft errors. Other systems focus on the communication. In [CBR+04], [CGB+06] and [VG09] the monitoring system is used to provide transaction data for debugging purposes. In [DF14] a tree-based debugging infrastructure is introduced that debugs end-to-end-transactions and recovers the system online in case of malicious behaviour.
Little effort has been done so far on run-time management and performance analysis. In both cases similar information is required and the goal of both areas is to achieve a better system performance and resource utilization. The main difference is that for run-time management the evaluation of monitoring data is performed on chip and for performance analysis on an external host. In [FPS09] and [FPS10] the monitoring data is evaluated to adjust adaptable and reconfigurable systems. External performance analysis is executed in [ASNH10] and [HAS+08] to provide the system designer with appropriate run-time data.
The ESG (Embedded Systems Group) Monitoring System is primarily designed to deliver information for performance analyses. In opposite to [CBR+04] and [CGB+06] the ESG Monitoring System is non-intrusive and offers higher reusability. It is not utilizing the monitored NoC itself for its internal communication and therefore just the front-end of the probes are NoC specific. Furthermore it is run-time reconfigurable, which enables the system designer to change the data granularity without resynthesizing the system. The probes are monitoring one link exclusively. This property allows to monitor parallel communication on all links at the same time.
The probes in [ASNH10], [FPS09], [FPS10] and [HAS+08] mainly consist of counters, which measure different events and qualities. Even this data might be sufficient for a lot of analyses the ESG Monitoring System provides more information, which enables more precise analyses like reconstructing the paths of packets and examining delays within the system. Additionally it is the only approach that includes a data compression scheme to decrease the overall amount of monitoring data. Simulation-based approaches like [RCM14] are not considered in this section since the focus lies on monitoring real-time behaviour on-chip.

3 ESG Monitoring System

NoC traffic analyses require concurrent transaction tracing. In order to achieve concurrency, probes are attached to communication links. The **probe placement** is an important design time choice. It determines both, the obtainable information and its granularity. The scheme offers high flexibility and maximum design freedom. One option is to place the probes at the links between the network interfaces (NI) and the routers. This option is only capable of end-to-end analyses (total delay, communication between entities, etc.) but also results in the lowest complexity. The internal run-time behaviour of the NoC (contention, traversed paths, etc.) can be monitored, if the probes are placed at links between routers.

Using exclusive probes for individual links enables concurrent monitoring. The application of the ESG Monitoring System requires (1) packet based transactions, (2) flit based packet composition and (3) wormhole switching.

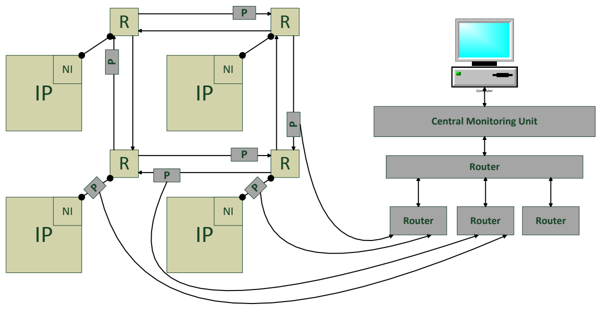


Figure 1: In the ESG Monitoring System the probes are placed directly at the NoC links. The communication infrastructure is organised in a tree topology where routers forward the monitoring packets from the probes (P) to the CMU. The IP cores (IP), NoC routers (R) and network interfaces (NI) are part of the monitored NoC and do not belong to the monitoring system.

Communication Infrastructure. A common approach is to use NoCs as the internal communication medium of monitoring systems. The first option is to use the monitored NoC itself. Since extra load would distort performance analyses it is not suitable for our purposes. The second option is to use a dedicated NoC. Even though it is non-intrusive it is questionable if NoCs are the best choice. The reason is that in monitoring systems all communication is concentrated on a central module. Therefore the parallel communication abilities of NoCs are not required. If the mostly unidirectional communication behaviour of monitoring systems is taken into account, a tree based topology (see Fig. 1) appears to be a good option [PVS$^+$10]. Packets are routed from the probes (leaves) to the Central Monitoring Unit (the root) by routers. Functions like address translation, complex routing algorithms or ID management are not required which means that low cost routers are employed. The main advantage is high scalability. If the probe number increases, new routers and levels in the tree architecture can be easily introduced. Another advantage is that the buffers in the routers are logically shared between the probes.

Probe Architecture. The probes monitor the transactions on the NoC links, process the information and send it to the Central Monitoring Unit (CMU). Additional statistics are taken over a specified time window similar to [FPS09]. The monitored NoC (ESG NoC) has a larger bus width (144 bit) than the ESG Monitoring System (32 bit). Additionally extra information (header, timestamp) has to be attached to the raw data. This properties make it impossible to process a link utilization of 100 %. Usually not all information of a transaction are relevant for an analysis. Depending on the required information and to provide as much data as possible different modes are available. The modes mainly differ in their abstraction level similar to the modes presented in [CGB$^+$06]. The probe architecture is illustrated in Fig. 2. It consists of 4 main parts. The timer is used to generate timestamps and to trigger time-outs whereas the configuration register stores configuration bits that can be adapted during run-time. The configuration options are: (1) switching the probe on or off and (2) programming its mode. The FIFO is used to buffer the monitoring packets until they are read by a router.

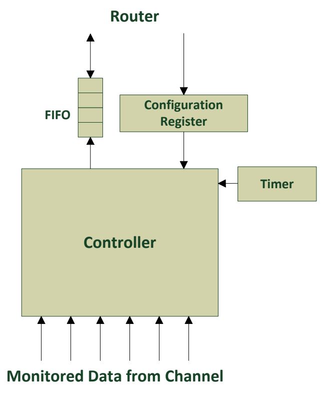


Figure 2: Probe Architecture

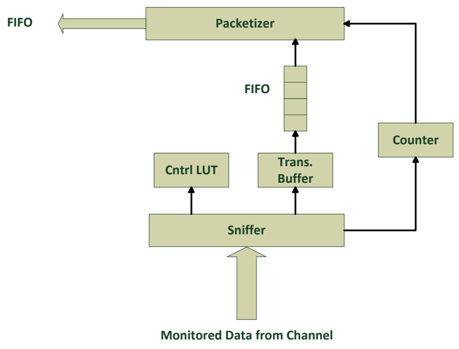


Figure 3: Probe Controller

The controller consists of two modules (see Fig. 3). The **Sniffer** analyses the data streams on the link utilizing a look-up-table (LUT). The **Control-LUT** is used to check if a packet completely passed the link. If a header flit is monitored, the packet-ID and the packet length are stored in the Control-LUT. For each subsequent payload flit the length will be decreased by one. Once the packet length reached zero the packet fully passed the link. The **Transaction Buffer** stores the characteristical parameters of the packet. If a packet passed the link, the parameters are taken from it and saved together with the timestamps and delays in the **Packetizer-FIFO**. The **Packetizer** takes the transaction information from the Packetizer-FIFO and stores it together with some static information (probe number, transaction type, etc.) in the required packet format in the FIFO on Fig. 2.

In **raw mode** the monitored raw data will be forwarded to a host. No abstraction or analysis is performed. The monitoring packets consist of 6 flits (32 bit each). The first flit is the header, which contains general and routing information. Since one NoC flit contains 4 data words (32 bits), 4 payload flits are added. The packet is completed by a timestamp. For each **NoC flit** a subsequent **monitoring packet** is sent. The header and the timestamp cause additional 33% overhead to the monitored payload. Since 6 clock cycles are required to send a monitoring packet it is not able to process a higher link utilization than 16.6 %.

The **packet mode** introduces the first abstraction level. Transactions are analysed from the packet perspective. The monitoring packet in this mode consists of 4 to 5 flits (see Fig. 5). The first flit is the header similar to the raw mode. The three subsequent flits contain the source address of the monitored NoC packet, the destination address and a timestamp. Depending on the burst length of the NoC packet, a fifth flit (the delay flit) will be attached. The delay flit contains the delays of all NoC flits to their predecessor[1]. For every **NoC packet** (not flit) a subsequent **monitoring packet** is sent. Compared to a NoC

[1]The ESG NoC operates with a guaranteed throughput and the maximal packet length is 8 flits. Therefore one monitoring flit is able to carry all delays, which might not be sufficient for other NoCs or other routing algorithms.

packet and depending on its burst length the data is comprised by 11 % to 86 % since the payload is removed. In this mode the probes are able to process a link utilization between 25 % an 100 %. 25 % in case the monitored NoC operates with single flit packets only, which can be theoretically monitored at each clock cycle. The subsequent monitoring packet would would consist of four flits and require four clock cycles to be transferred. Once the NoC packet consists at least of five flits the subsequent monitoring packet would have the same or a lower number of flits and the processable link utilization would increase to 100%.

The **transaction mode** provides the highest abstraction level. Instead of flit delays the whole transaction is analysed from the channel view. The header flit has an additional field. The field contains the delay between the header flit and the last payload flit (packet delay) in clock cycles. The monitoring packet length is four flits. The subsequent three flits (source address, destination address, timestamp) have the same function as in the packet mode. Just as in the packet mode for each NoC packet a subsequent monitoring packet is generated and the probe is able to process a link utilization of 25 % up to 100 % without information loss. The difference is that the probe is already able to process a link utilization of 100 % when the NoC packets consist at least of four flits. Table 1 presents an overview of the characteristical values of the modes.

Table 1: Comparison of the available Modes

	Raw Mode	Packet Mode	Transaction Mode
Abstraction Level	raw data	flit delays	packet delay
Monitored entities	flits	packets	packets
Packet Length	6 flits	4-5 flits	4 flits
Data reduction	+33.3 %	-11 % to -86 %	-11 % to -89 %
Max. link utilization	16.6 %	25 % to 100 %	25 % to 100 %

Central Monitoring Unit. The CMU collects the monitoring packets and manages the communication with the host. Since the CMU needs to forward concurrently collected data (monitoring packets) in a sequential manner it presents the critical bottleneck in the ESG Monitoring System. In addition the on-chip data rate usually exceeds the external data rate which worsens the problem. To reduce the overall amount of monitoring data a compression scheme is introduced. Fig. 4 illustrates the CMU architecture. The CMU consists of three main modules: the **Packet Analyzer** which analyses the incoming stream of monitoring packets, the **Packetizer** which performs the data compression and data packing and the **Sender** which transmits the packets to the host. The Packet Management RAM (**PM-RAM**) saves the characteristics of a NoC transaction as a parameter set. The parameters are static information of the monitoring packets that belong to the same monitored NoC transaction. The **Payload-RAM** stores the non-redundant information.

CMU Work Flow. To manage incoming monitoring packets and to determine which monitoring packets belong together the PM-RAM and Payload-RAM are used. The PM-RAM is managed by the PM-LUT (Packet Management LUT). The PM-LUT is a register file that keeps track about the status (occupied, available) of PM-RAM rows. Timers are used

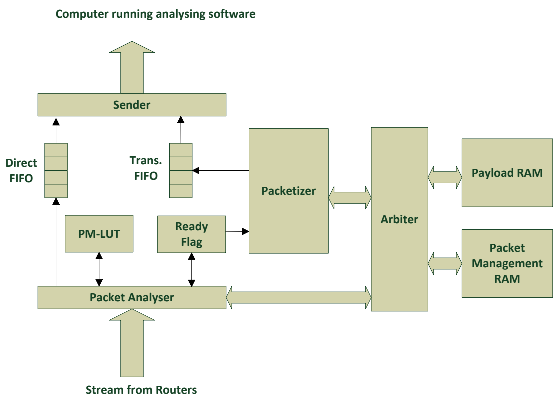


Figure 4: CMU Architecture

to determine if all monitoring packets of a particular NoC transaction are received. If for a specific time window no monitoring packets belonging to a particular NoC transaction are received, it is assumed that all information is received. With this scheme it is not necessary that probes are placed over the entire path. The Packetizer takes the parameters of the NoC transaction from the PM-RAM and the payload information from the Payload-RAM. Afterwards the information is packed into flits and saved in the Transaction-FIFO. The Sender takes the flits from the Transaction-FIFO and sends them to the host. The direct FIFO is used for statistical packets.

Data Compression. As mentioned earlier one of the main tasks of the CMU is to reduce the amount of monitoring data. With a close look on the monitoring packets in packet mode and transaction mode it becomes obvious that the monitoring packets contain a lot of redundant data. This redundancies are exploited in order to relax the time pressure at the interface between CMU and external host. The redundant parts of the packets are highlighted on Fig. 5.

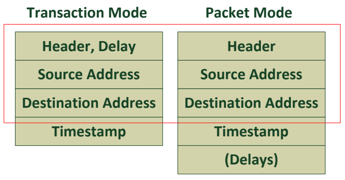


Figure 5: Redundancies within monitoring packets that belong to the same NoC transaction

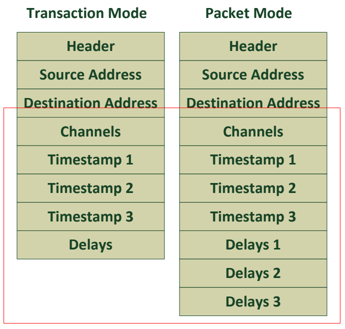


Figure 6: CMU Packet Structure

The header information, the source address and the destination address are directly taken from the original NoC packet. The only non-redundant information within the monitoring packets (the parts which differ between the monitoring packets that belong to the same NoC transaction) is the probe number, the timestamps and the delays. The probe number (a field in the header flit) identifies on which link the packet was monitored and the timestamp shows the time the header flit passed the channel. The delay flit contains the delays between the NoC flits (packet mode) or the delay field in the header flit the overall packet delay (transaction mode). In order to save bandwidth the redundant information is removed. The same principle is utilized by the Stream Analyzer. The redundant information of monitoring packets is stored once in the PM-RAM and the variable information in the Payload-RAM. In order to reduce the amount of data a new packet format for the communication between the CMU and a host is defined (see Fig. 6). The first three flits are similar to the monitoring packets and are only transmitted once. For both modes a new flit type is introduced. The channel flit contains one to three probe numbers (depending on the number of links the NoC packets passed), which were initially part of the variable information of the monitoring packets. The next flits are one to three timestamps (depending on how many probe numbers the channel flit contains). The timestamps are followed by delay flits. The number of delay flits differs between the modes. In the packet mode for each channel in the channel flit a corresponding delay flit has to be sent. In the transaction mode the delays for each channel in the channel flit can be sent within one flit (one delay flit belongs to one channel flit). If a NoC packet passed more than 3 channels, channel flits, timestamp flits and delay flits are added in the same order.

The number of required channel flits (cfn) for the number of received monitoring packets (x) can be calculated as follows:

$$cfn(x) = \begin{cases} cfn(x+1), & \text{if x mod 3 } > 0 \\ x/3, & \text{if x mod 3 = 0} \end{cases} \tag{1}$$

The CMU packet length (cpl) in the transaction mode:

$$cpl(x) = 3 + 2 * cfn(x) + x \tag{2}$$

and in the packet mode:

$$cpl(x) = 3 + cfn(x) + 2 * x \tag{3}$$

Example: To demonstrate the efficiency of the compression scheme we take an example in which a NoC packet consisting of 8 flits passes 9 channels. The system is assumed to run in the transaction mode. Initially: $9 * 4 = 36$ flits are received. After data compression $3 + 2 * (\frac{9}{3}) + 9 = 18$ flits are left. In this small example the compression scheme reduces the amount of data by 50%. The gain increases the more channels are passed and decreases if the distances in the NoC are relatively small.

4 Results

All modules were tested and evaluated individually to prove their functionality. Afterwards a simulation of a 2x2 NoC, based on ESG NoC components, has been established to test the ESG Monitoring System in a realistic configuration by connecting four processor models with four memory instances. To evaluate the NoC functionality port models of processing elements are used instead of full functional processor cores. To verify the functional and non-functional parameters of the monitoring system the architecture has been implemented on the basis of an FPGA validation system.

Resource Utilization. The whole design has been implemented in synthesiseable VHDL code. The resource costs have been verified by synthesizing the modules individually with corresponding EDA tools for the used FPGA technology. Results are shown in Tab 2. The results for combinatorial ALUTS are given as absolute numbers. The numbers in brackets show the size in relation to the monitored NoC.

Table 2: Resource utilization and clock frequency

	Monitoring Router	Probe	CMU
Combinatorial ALUTS	433 (<1%)	2515 (<4%)	4940(<8%)
Total registers	228	1433	2955
Total block memory bits	0	5632	92928
FMAX	330 MHz	254 MHz	159 MHz

Tracing Performance. The first investigation considers the ratio between the real and the theoretical probe performance. A channel workload of 100 % with different NoC packet lengths has been simulated. The results show a discrepancy between the simulation results and the theoretical values described in Sec. 3. Table 3 shows the overview of the results.

Table 3: Probe Performance Analysis

Mode	Raw	Packet	Transaction
Required NoC packet length to monitor 100% link utilization (theo./real)	/	5 flits/ 7 flits	4 flits/ 5 flits
Max. monitorable channel load (theo/real.)	16.6%/ 14.3%	25% to 100%/ 20% to 100%	25% to 100%/ 20% to 100%

The differences are based on delays, which are caused by implementation details. An example is that the finite state machine (FSM) of the packetizer needs to reach its idle state between the processing of subsequent monitoring packets. Therefore it takes 5 clock cycles to send a transaction packet instead of 4. Also each router in the monitoring system introduces an additional delay of one clock cycle. The simulated workload of 100 % presents the worst case. For real applications and NoCs with smaller bus widths the probes would achieve better results. The analysis of the CMU shows that the compression scheme introduces a noticeable delay. The data compression algorithm eliminates redundant information. Therefore the delay is balanced and a positive yield is achieved if a specific amount of monitoring packets belonging to the same NoC transaction are received. The following formula shows when the break even point is reached, considering the monitoring packet length (mpl) and the number of received monitoring packets (x):

$$\text{Break Even Point} = 100 - \frac{x * (\text{mpl+1})}{7 * x} \tag{4}$$

If we take the example from section 3 the break even point is reached if the external data rate is $(100 - \frac{9*(4+1)}{7*9}) = 28\,\%$ lower than the internal one. Since the CMU represents the critical bottleneck in the monitoring system it is creating backpressure for the probes. Based on simulations and calculations it is estimated that a monitoring system with 20 probes would be able to process 5.5% average link utilization. The assumptions that are made are that the compression scheme is enabled and the NoC monitoring systems operates with the same clock frequency as the monitored NoC. The external data rate is neglected.

5 Conclusion and Future Work

In this paper we presented a universal and run-time reconfigurable monitoring system that enables performance analysis and debugging support for SoCs. Probes are tracing communication activities within the SoC and provide different abstraction modes to provide as much and as accurate data as possible. The probes are placed at links that potentially carry relevant information. The probe placement offers high flexibility which enables a good trade off between resource cost and obtainable information. The probes send packets to a central monitoring unit which is responsible for the communication to an external host. Since it is not always known beforehand which information is required during an analysis, all components within the ESG Monitoring System can be reconfigured at run-time. This enables to regulate the amount of data and the data granularity without resynthesizing the system. To offer high reusability and to not interfere the monitored system, a dedicated tree based communication infrastructure is established. Also a compression scheme, which removes redundant data, is presented. The evaluation showed that the system benefits from the compression scheme in case the NoC packets pass a certain average number of links and if the external bandwidth is slower than the internal one. The provided data can be used by the system designer to align the SoC architecture or the programming model to achieve a better system performance and resource utilization. Communication debugging in case of malicious behaviour is also supported. The resource utilization of the monitoring system shows reasonable costs in comparison to the cost of the monitored NoC itself. The future work will include the adaptation and usage of the monitoring system for run-time resource management. As mentioned earlier run-time management and performance analysis require similar information. In this sense the CMU will be upgraded with run-time management capabilities to adjust reconfigurable and adaptive parts of an SoC.

References

[ASNH10] A. Alhonen, E. Salminen, J. Nieminen, and T. D. Hamalainen. A scalable, non-interfering, synthesizable Network-on-chip monitor. In *Proc. NORCHIP*, pages 1–6, 2010.

[BDM02] L. Benini and G. De Micheli. Networks on chip: a new paradigm for systems on chip design. In *Design, Automation and Test in Europe Conference and Exhibition, 2002. Proceedings*, pages 418–419, 2002.

[CBR+04] C. Ciordas, T. Basten, A. Radulescu, K. Goossens, and J. Meerbergen. An event-based network-on-chip monitoring service. In *Proc. Ninth IEEE Int. High-Level Design Validation and Test Workshop*, pages 149–154, 2004.

[CGB+06] C. Ciordas, K. Goossens, T. Basten, A. Radulescu, and A. Boon. Transaction Monitoring in Networks on Chip: The On-Chip Run-Time Perspective. In *Proc. Int. Symp. Industrial Embedded Systems IES '06*, pages 1–10, 2006.

[DF14] M. Dehbashi and G. Fey. Transaction-Based Online Debug for NoC-Based Multiprocessor SoCs. In *Parallel, Distributed and Network-Based Processing (PDP), 2014 22nd Euromicro International Conference on*, pages 400–404, 2014.

[FPS09] L. Fiorin, G. Palermo, and C. Silvano. MPSoCs run-time monitoring through Networks-on-Chip. In *Proc. DATE '09. Design, Automation & Test in Europe Conf. & Exhibition*, pages 558–561, 2009.

[FPS10] L. Fiorin, G. Palermo, and C. Silvano. A Monitoring System for NoCs. In *Third International Workshop on Network on Chip Architectures. New York, USA*, 2010.

[HAS+08] K. Holma, T. Arpinen, E. Salminen, M. Hannikainen, and T. D. Hamalainen. Real-time execution monitoring on multi-processor system-on-chip. In *Proc. Int. Symp. System-on-Chip SOC 2008*, pages 1–6, 2008.

[MVBT09] S. Madduri, R. Vadlamani, W. Burleson, and R. Tessier. A monitor interconnect and support subsystem for multicore processors. In *Proc. DATE '09. Design, Automation & Test in Europe Conf. & Exhibition*, pages 761–766, 2009.

[PVS+10] B. Phanibhushana, P. Vijayakumar, P. Shabadi, G. Prabhu, and S. Kundu. Towards efficient on-chip sensor interconnect architecture for multi-core processors. In *Proc. Int. SoC Design Conf. (ISOCC)*, pages 307–310, 2010.

[RCM14] M. Ruaro, E.A. Carara, and F.G. Moraes. Tool-set for NoC-based MPSoC debugging — A protocol view perspective. In *Circuits and Systems (ISCAS), 2014 IEEE International Symposium on*, pages 2531–2534, 2014.

[VG09] B. Vermeulen and K. Goossens. A Network-on-Chip monitoring infrastructure for communication-centric debug of embedded multi-processor SoCs. In *Proc. Int. Symp. VLSI Design, Automation and Test VLSI-DAT '09*, pages 183–186, 2009.

[ZMV+11] Jia Zhao, S. Madduri, R. Vadlamani, W. Burleson, and R. Tessier. A Dedicated Monitoring Infrastructure for Multicore Processors. 19(6):1011–1022, 2011.

Proximity Scheme for Instruction Caches in Tiled CMP Architectures

Tareq Alawneh, Chi Ching Chi, Ahmed Elhossini, and Ben Juurlink

Embedded Systems Architectur (AES)
Technical University of Berlin
Einsteinufer 17
D-10587 Berlin, Germany
{tareq.alawneh, cchi, ahmed.elhossini, b.juurlink}@tu-berlin.de

Abstract: Recent research results show that there is a high degree of code sharing between cores in multi-core architectures. In this paper we propose a proximity scheme for the instruction caches, a scheme in which the shared code blocks among the neighbouring L2 caches in tiled multi-core architectures are exploited to reduce the average cache miss penalty and the on-chip network traffic. We evaluate the proposed proximity scheme for instruction caches using a full-system simulator running an n-core tiled CMP. The experimental results reveal a significant execution time improvement of up to 91.4% for microbenchmarks whose instruction footprint does not fit in the private L2 cache. For real applications from the PARSEC benchmarks suite, the proposed scheme results in speedups of up to 8%.

1 Introduction

Recently, Chip Multiprocessor (CMP) architectures have become very common. They became the focus of the leading CPU manufacturers (Intel, AMD, and IBM). The power wall and memory wall are the two main issues motivating the move from uniprocessor to CMP architectures to achieve high performance. With the continuous increase in the number of integrated processors (cores) on a single die and the size of the caches in CMP architectures, the high access latency of the large cache available on the chip is becoming critical part of future CMP architectures [ZA05]. Tiled CMP architectures are introduced as an efficient solution for the future scalable CMPs. These architectures are designed as arrays of identical tiles connected over a switched network on-chip (NoC). In tiled CMP architectures that employ the directory-based protocol to maintain cache coherence, when an instruction cache miss occurs in the Last Level Cache (LLC), a request is issued to the directory to obtain the code block, although the desired code block may be present in one of the neighbouring cores. Increasing the number of integrated cores in a single chip will increase the average number of network hops traversed to satisfy the request. As a result, the cache miss penalty and the on-chip network traffic will increase. In this paper, we present a proximity scheme for instruction caches in tiled CMP architectures. When an L1 instruction cache miss occurs, the desired code block is requested in parallel from the neighbouring cores L2 caches as well as the private L2 cache. This makes it possible to resolve most L1 instruction misses in just few cycles by accessing the neighbouring L2 caches via dedicated paths instead of requesting the directory. This reduces the average cache miss penalty in instruction cache. Moreover, the available on-chip cache capacity is utilized effectively if the forwarded code blocks from the neighbouring cores are just copied in the local L1 instruction cache. This also reduces the on-chip network traffic by eliminating unnecessary message to the directory. The contributions of this paper can be summarized in the following points:

- Introducing a proximity technique for the instruction caches in the tiled CMP architectures, a scheme in which the desired code blocks, when a miss occurs in the L1-I cache, are serviced, in parallel, by the private L2 cache or the neighbouring caches before contacting the directory structure. Therefore, our proposed scheme reduces the cache latency, which in turn improves execution time.
- Furthermore, our proposed scheme reduces the on-chip network traffic as a result of reducing the number of required hops to reach the requested code block.
- Finally, we show that our scheme scales extremely well with the number of cores. In other words, it is particularly well-suited for large-scale CMP architectures.

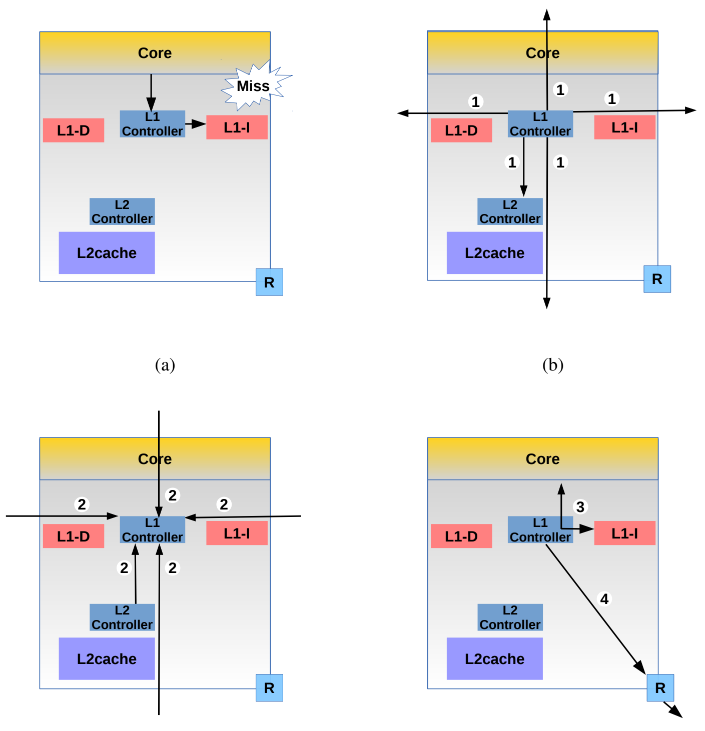
2 Related Work

Several recent studies have proposed schemes to reduce the average cache miss penalty in the tiled CMP architectures. Brown et al. [BKT07] presented an algorithm which is proximity aware. Their proposed algorithm is based on the following observation: although the desired data is not present in the L2 cache of the home node, it might be still resided in other nodes. Therefore, the home node can issue a message to the closest sharer, requesting it to forward the desired data. This reduces the number of requests to the off-chip memory. Requests to the directory still introduce a major load on the on-chip network traffic to locate a node in proximity of the home node.

Hossain et al. [BKT08] introduced a scheme where a direct access is performed to the predicted remote L1 cache, which is likely to contain the desired data before requesting the directory. In their work, the desired data is requested from the close-by cache instead of neighbouring caches. Furthermore, the forwarded data are not usable before receiving an acknowledgement from the directory. Unlike our work, the provided code blocks from the neighbouring caches are usable immediately by the requested tile upon receiving them.

Williams et al. [WFM10] presented a scheme, in which a request is sent to the neighbouring caches before contacting the directory when a load miss occurs. Point-to-point links are used to transfer cache block between neighbouring nodes. Directory is contacted only when none of the neighbours have a copy of the requested data. This approach is introduced only for data caches.

Previous studies focused on improving the performance of the data caches. In contrast, our work aims at improving the instruction caches with a proximity scheme. Investigating the proximity scheme allows us to exploit the read-only property of instruction cache blocks with a less hardware overhead.



(a) (b) (c) (d)

Figure 1: The detailed behaviour of proximity coherence for the instruction caches in tiled CMP architectures

3 The Proposed Proximity Scheme for Instruction Caches

In this paper we propose a proximity scheme for instruction cache. Dedicated links between each core and its four neighbouring cores are used to transfer the required code block from the neighbours L2 cache in the case of an instruction cache miss. The state machine of the conventional MOESI protocol is also modified to enable the use of these dedicated link. Figure 1 shows the detailed behaviour of the proximity mechanism for the instruction caches. When an instruction fetch operation results in an instruction miss (as shown in Figure 1a), the L1 cache controller sends out parallel requests to the private and neighbouring L2 caches (message 1 in Figure 1b) instead of sending a direct request to the directory.

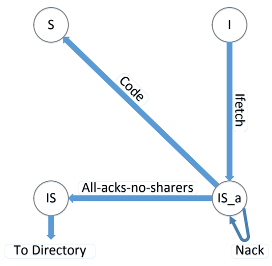


Figure 2: The additional transient states in our proximity scheme for instruction caches when an L1 instruction cache miss occurs

In the meantime, the state of the missed code block is changed to a transient state. If the private or neighbouring L2 caches have it, they reply with a hit, otherwise they return a miss (message 2 in Figure 1c). Once the L1 cache controller receives a hit from the private or the neighbouring L2 caches, the forwarded code block is copied in the private L1-I cache and forwarded to the core simultaneously (message 3 in Figure 1d) and its state is changed to the shared state. On the other hand, when the L1 controller receives all the responses and all of them replay with a miss, a request message for the directory will be issued (message 4 in Figure 1d).

Figure 2 shows the additional transient states that are added to the state machine of the conventional MOESI protocol. When an L1 instruction cache miss occurs, messages are issued to the private L2 and neighbouring caches at the same time. The code block state is moved to the *IS_a* state, which indicates that a request message is issued, waiting the responses from the private L2 cache and all neighbouring cores. If all of the them return a NACK message, which means that none of them contains a valid copy of the required code block, the code block state is moved to *IS* which indicates that a request message is issued to the directory. On the other hand, if one of the requested caches responds by sending a valid copy of the required code block, after forwarding the requested code block to the core, the code block is directly copied in the L1-I cache and its state is changed to the shared state (*S*).

4 Experimental Setup

By using full-system simulations based on the GEM5 [BBB$^+$11], we evaluate the performance of our scheme against the baseline system which employs the MOESI directory-based coherence protocol. To implement a detailed simulation model for the memory subsystem, the Ruby memory model in GEM5 is used. Using a higher-level language in the GEM5 (a.k.a. SLICC), we specify our extension by modifying the state machine of the conventional MOESI protocol with all new transient states. The existing Gem5 network model is augmented with dedicated links between the neighbouring cores. Table 1 describes the values of the main parameters of the evaluated baseline system in this study. We study the tiled CMP architecture which consists of n replicated tiles interconnected

Table 1: System parameters for full-system simulation

Tiled CMP size	4, 8, 16, 32, 64 cores
L1-I and L1-D Cache Size	32 KB per core
L1-I Cache Hit Latency	3 cycles
L1-D Cache Hit Latency	3 cycles
Private L2 Cache Size	256 KB per core
L2 Cache Hit Latency	15 cycles
L1 and L2 Block Size	64 B
Network Configuration	2D Mesh Topology
Memory Size	8 GB
Memory Latency	250 cycles
Dedicated Links Latency	1 cycle

with a 2D mesh switched network as shown in Figure 3. Each tile consists of a processor core, a private split first-level instruction and data cache (L1-I/L1-D), a private second-level cache (L2), a network interface or router for on-chip data transfers and a directory to keep track of cores with copies for cached blocks.

For our evaluation, we first used microbenchmarks with the purpose of generating a high L1-I miss rate. These microbenchmarks are composed of a mixture of jump (JMP) and no operation (NOP) instructions. Each jump instruction is padded with NOP operations to fill a complete cache block of 64 bytes. The JMP instruction in every code block jumps to the JMP instruction in the next code block. In these microbenchmarks, the previous instruction pattern is replicated to create the various program sizes (i.e. its instruction footprint) from 4kB to 2MB. The entire program code is looped over by a specific count in order to have the same number of instruction cache block requests for all micro benchmarks (e.g., 8192 loops for 4 kB and 16 loops for 2MB). Each core executes the same microbenchmark. We employ this microbenchmark because it allows to precisely identify when the proposed technique is effective and when not.

Another set of experiments was performed to test the system with real applications. We evaluate some applications from PARSEC [BKSL08] parallel benchmark suite. Each run consists of n-threads of the application running on the n-core tiled CMP. Table 2 lists the applications which are simulated in this study. Due to the large number of benchmarks and the relatively long simulation time, we selected seven workloads from the PARSEC benchmark suite to cover the various application domains as shown in Table 2.

5 Evaluation Results and Analysis

In this section, we present the results and the analysis of the simulation results that have been obtained when running the microbenchmarks and the PARSEC suite benchmarks using our proposed proximity for the instruction caches compared to the baseline system which employs the traditional MOESI directory-based protocol.

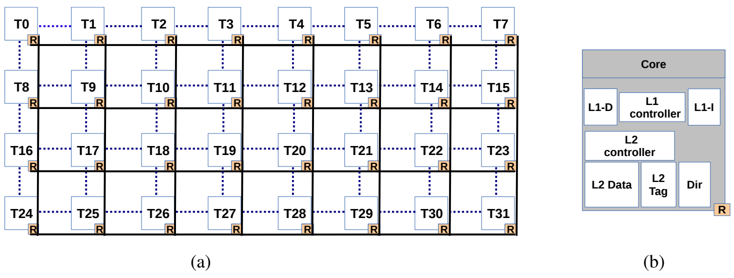


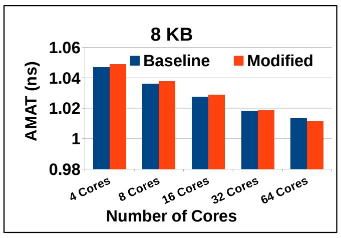
Figure 3: An example of the adopted tiled many-core processor. (a) A 4 X 8 tiled CMP (b) The architecture of a single tile. The continuous black lines show the global on-chip interconnect. The dashed black lines show the proximity links that connect L2 caches

Table 2: Simulated workloads

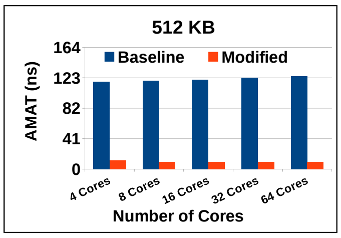
Benchmark Name	**Application Domain**
Blackscholes	Financial Analysis
Swaptions	Financial Analysis
X264	Media Processing
Dedup	Enterprise Storage
Canneal	Engineering
Fluidanimate	Animation
Freqmine	Data Mining

5.1 Evaluation using Micro-benchmark

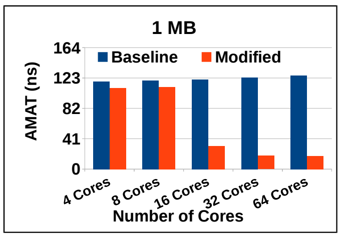
Figure 4 depicts the actual Average Memory Access Time (AMAT) achieved by our proximity scheme for different instruction footprints of the micro-benchmarks and for different numbers of cores. It can be seen that when the instruction footprint is smaller than or equal to 256KB, the improvements are small and in some cases even negative. The reason for this is that these instruction footprints fit in the L2 cache (Table 1). Some microbenchmarks do not provide any improvement in the AMAT as might be expected (some of them achieve a slight reduction in the AMAT due to the reduction in the cold misses). However, other microbenchmarks show an insignificant increase in the AMAT, less than 2% compared to the baseline system. The latency to access the neighbour cache is few cycles longer than that of the private L2, and combined with the fact that no local L2 copy is created in case of a hit in a neighbouring cache, the average proximity hit latency is higher than the average private L2 hit latency in the baseline approach. When the code size is 512 KB, the proposed scheme provides huge benefits, ranging from 90.3% for 4 cores to 92.1% for 64 cores. In this case the instruction footprint does not fit in the private L2 cache of a single core anymore, but a core and its neighbors together provide sufficient cache capacity to hold the entire instruction footprint. When the code size is again doubled to 1MB, interesting behavior can once more be observed. In that case significant benefits are obtained for 16, 32, and 64 cores, but for 4 and 8 cores the improvements are rather small. The reason is that when the number of cores increases, so does the average number



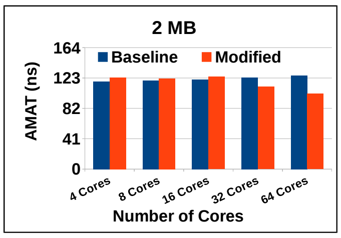
(a)



(b)



(c)



(d)

Figure 4: The actual Average Memory Access Time (AMAT) in the baseline and our proposed scheme

of neighbors. For example, in a 4-core (2×2) CMP, each core has exactly 2 neighbors, and so the aggregate L2 cache size of a core and its neighbors is $3 \times 256 = 768$KB, which is smaller than the instruction footprint. Similarly, in a 8-core (4×2) CMP, each core has 2 or 3 neighbors, and so the aggregate L2 cache size of a core and its neighbors is 896KB on average, which is still smaller than the instruction footprint. On the other hand, when the number of cores is 64 (8×8), the average number of neighbors is almost 4 (3.5), meaning that the instruction footprint fits in the aggregate L2 of a core and its immediate neighborhood. When the code size of the micro-benchmarks is again doubled, however, the instruction footprints no longer fits in the L2 caches of a core and its neighbors, which is why the improvements decrease substantially. For 32 and 64 cores, the proximity scheme still provides improvements, but for 16 and fewer cores, there is a small slowdown. In the 16 and fewer cores, the aggregate L2 cache size of a core and its neighbors is too small to hold the microbenchmark's footprint. Therefore, the proximity hit rate is low, which in turn increases the AMAT. The latency to access the neighbouring caches is few cycles higher than the access to the private L2 cache. Although, the aggregate L2 cache size of a core and its neighbors in the 32 and 64 cores is still small to hold microbenchmark's full footprint. The reduction in the L2 cache miss penalty is higher than the proximity overhead, which is why the proposed scheme still achieves improvements in the 32 and 64 cores.

Figure 5 shows the overall reduction in execution time presented by the proposed approach compared to the baseline for the same set of microbenchmarks. Microbenchmarks with small instruction footprints, as expected, do not provide any significant improvement in the execution time. On the other hand, the aforementioned improvements in the AMAT translated into reduction on the overall execution time in microbenchmarks with a footprint that does not fit in the private L2 cache (512KB and 1MB). Our proposal achieves execution time reduction of up to 91.4% compared to the the baseline. Similar behaviour can be observed for the 2MB benchmark. The achieved reduction in execution time corresponds to the reduction in the AMAT for the same benchmark.

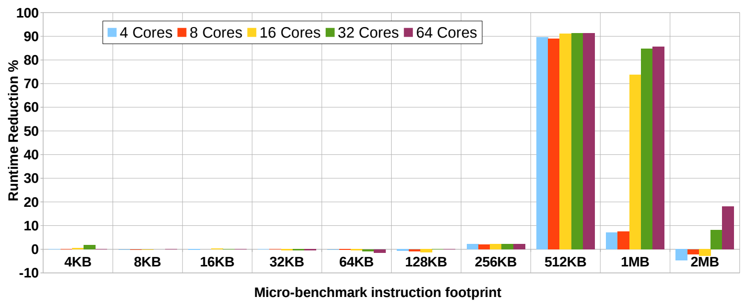
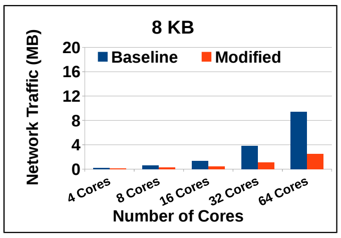


Figure 5: Runtime reduction in our proposed approach compared to the baseline system

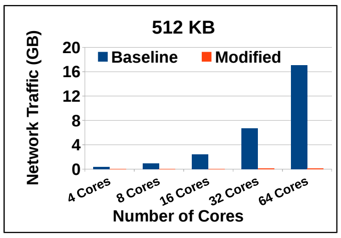
Figure 6 shows the aggregate number of bytes transferred by the on-chip network. The proposed approach introduces a significant reduction in the on-chip network traffic for all microbenchmarks compared to the baseline system which employs the traditional MOESI directory-based protocol. Contacting the neighbouring cores via the dedicated links to obtain the required cache blocks significantly reduced the on-chip network traffic. The microbenchmarks with instruction footprints smaller than 32KB, provide on average 38.7% reduction in the on-chip network traffic. For these benchmarks most of the traffic is due to the cold misses and they are served mainly by the dedicated links from neighbouring caches. As a result, the total on-chip network traffic is reduced. In all CMP configurations based on Table 1, we can observe that the proposed scheme introduces more reduction in the on-chip network traffic compared to the baseline system as the number of core count grows. When the data and control messages travel using the on-chip network, they may take several hops to reach the destination. The average number of hops increases as the number of cores increases, which is significant for the baseline system. Larger microbenchmarks that do not fit in the L1 cache but still fit in the L2 cache present similar reduction in the on-chip network traffic for the same aforementioned reasons. However, microbenchmarks with foot-prints 512KB and 1MB benefit more because their foot-print does not fit in the L2 cache. In the baseline system, misses to the L2 cache are serviced by contacting the directory which increases the network traffic. In our approach these requests are serviced by the neighbouring cores eliminating the network traffic. On average, our proposed approach achieves a reduction in the network traffic by 45%.

5.2 Evaluation using Real Benchmarks

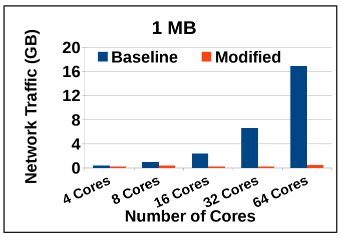
Evaluating the proposed scheme using the microbenchmarks shows that the proposed approach can be used to reduce the execution time of application by reducing the average memory access time. This approach can be used to reduce the on-chip network traffic as



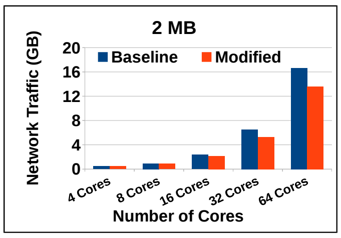
(a)



(b)



(c)

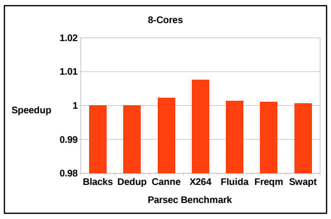


(d)

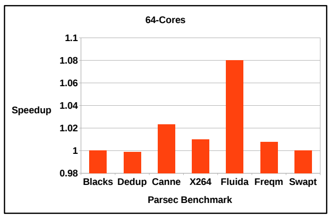
Figure 6: Total number of bytes (in GB or MB) transferred by on-chip global network in our proposed approach compared to the baseline system

well. In this section, we present the evaluation results using real applications from PARSEC benchmark suite. All simulations are performed using GEM5 as explained in Section 4. The benefit of our proposed proximity scheme for the instruction caches is limited by the high L1-I cache hit rates which are observed in the simulated PARSEC workloads. Therefore, we reduce the L1-I cache size to 4KB to show this benefit. Figure 7 shows the execution time speedups that have been obtained for PARSEC benchmarks when shrinking the L1-I cache to 4KB for 8 and 64 cores. The proposed approach achieved speedups for all benchmarks up to 8% compared to the baseline system. The blackscholes has a small workload that fits in the L1-I cache. Therefore, it achieves a slight speedup due to the cold misses which are serviced by the neighbouring caches. On the contrary, the speedup in the fluidanimate workload increases as the number of cores grows. This comes from the increase in the average aggregated cache capacity of the neighbouring cores. The improvements in the other benchmarks are small and some cases even negative. The reason for this is that their instruction footprints fit in the private L2 cache. The slight speedups, which achieved in some benchmarks, result from the reduction in the cold misses. On the other hand, the small slow down in other benchmarks comes from the low proximity hit rate. Therefore, the AMAT increases due to the proximity overhead.

Figure 8 shows speedup for reduced L1-I cache size of 4KB and the L2 cache size of 64KB for 8 and 32 cores. In the case of the canneal and x264 benchmarks, the proposed scheme provides speedup up to 33% and 7.5% respectively. The instruction footprints of these two benchmarks do not fit in the private L2 cache of a single core anymore, but a core and its neighbors together provide sufficient cache capacity to hold the entire instruction footprint. On the contrary, other benchmarks still fit in the L2 cache and provide slight speedups due to the reduction in the cold misses.

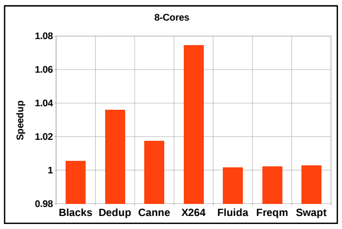


(a)

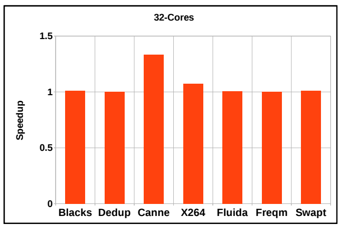


(b)

Figure 7: The achieved runtime speedup in our proposed approach compared to the baseline system. When L1-I cache size is 4KB, L1-D cache size is 32KB, and L2 cache size is 256KB

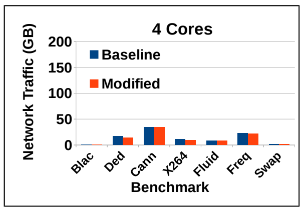


(a)

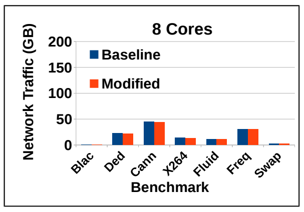


(b)

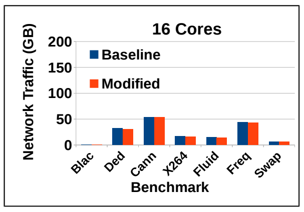
Figure 8: The achieved runtime speedup in our proposed approach compared to the baseline system in small and large CMP configurations. When L1-I cache size is 4KB, L1-D cache size is 32KB, and L2 cache size is 64KB



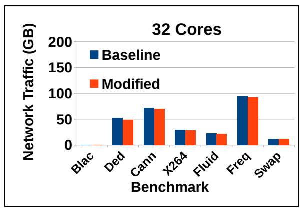
(a)



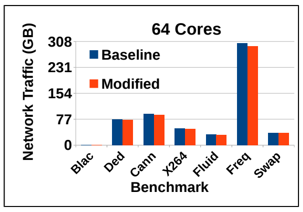
(b)



(c)



(d)



(e)

Figure 9: Total number of bytes (in GB) transferred by on-chip global network in our proposed approach compared to the baseline system. When L1-I cache size is 4KB, L1-D cache size is 32KB, and L2 cache size is 256KB

Figure 9 shows the aggregate number of bytes transferred by the global on-chip network when the L1-I cache size is 4KB. As shown in the Figures from 9a to 9e, the total bytes transferred by the on-chip network in the blackscholes is negligible for both the traditional and proposed systems, because most of the L1-I requests are serviced by the L1-I cache. We can also observe that all the simulated PARSEC benchmarks achieve a reduction in the on-chip network traffic (9.4% on average) when our proposed mechanism is employed. This is due to requests which are serviced via the dedicated links, which in turn reduce the number of the messages transferred by the on-chip network.

6 Conclusions and Future work

Tiled CMP architectures are introduced as the best choice for the future scalable CMPs. As the number of the cores grows, the average cache miss penalty will have a significant impact on the overall performance. Several approaches have been proposed to reduce the average cache miss penalties in the tiled CMP architectures which employ a directory-based coherence protocol. These approaches focused on improving the performance of data caches. In this work, we propose a proximity scheme for the instruction caches to provide execution time improvements as well as reduction in the on-chip network traffic. Our results reveal a significant reduction in the overall execution time and global network traffic of up to 91.4% and 99%, respectively, for the microbenchmarks whose instruction footprint exceeding the private L2 cache size. Moreover, the proposed policy led to improvement in the PARSEC workloads execution time of up to 8% compared to the baseline system.

Applications with large footprints that do not fit in the modern L1-I cache, such as Online Transaction Processing (OLTP) workloads [LBE$^+$98, ATAM12, HA04, KADS03, IATAM13] are most likely to benefit from the proposed approach. They exhibit a high degree of instruction reuse over multiple cores. Several prior studies observed this extensive sharing of instruction blocks among the multiple processors in different workloads [ATAM12, KADS03, CWS06, HA06]. While we do not simulate these workloads in this study, they will be considered in future work. Moreover, a detailed analysis of the power consumption for our proposed scheme is left to future work.

Acknowledgment

This work is funded by the Erasmus Mundus program of the European Union, Erasmus Mundus EPIC project.

References

[ATAM12] I. Atta, P. Tozun, A. Ailamaki, and A. Moshovos. SLICC: Self-Assembly of Instruction Cache Collectives for OLTP Workloads. In *Proceedings of the 45th Annual IEEE/ACM International Symposium on Microarchitecture, MICRO*, pages 188–198, 2012.

[BBB+11] N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, T. Krishna, S. Sardashti, R. Sen, K. Sewell, M. Shoaib, N. Vaish, M. D. Hill, and D. A. Wood. The Gem5 Simulator. *ACM SIGARCH Computer Architecture News*, 39:1–7, May 2011.

[BKSL08] Christian Bienia, Sanjeev Kumar, J. P. Singh, and K. Li. The PARSEC Benchmark Suite: Characterization and Architectural Implications. In *Proceedings of the 17th International Conference on Parallel Architectures and Compilation Techniques, PACT*, pages 72–81, 2008.

[BKT07] J. A. Brown, R. Kumar, and D. Tullsen. Proximity-Aware Directory-based Coherence for Multi-core Processor Architectures. In *Proceedings of the 19th Annual ACM Symposium on Parallel Algorithms and Architectures, SPAA*, pages 126–134, 2007.

[BKT08] J. A. Brown, R. Kumar, and D. Tullsen. Improving Support for Locality and Fine-Grain Sharing in Chip Multiprocessors. In *Proceedings of the 17th International Conference on Parallel Architectures and Compilation Techniques, PACT*, pages 155–165, 2008.

[CWS06] K. Chakraborty, P. M. Wells, and G. S. Sohi. Computation Spreading: Employing Hardware Migration to Specialize CMP Cores on-the-fly. In *Proceedings of the 12th International Conference on Architectural Support for Programming Languages and Operating Systems, ASPLOS*, pages 283–292, 2006.

[HA04] S. Harizopoulos and A. Ailamaki. Steps Towards Cache-Resident Transaction Processing. In *Proceedings of of the 30th International Conference on Very Large Data Bases, VLDB*, pages 660–671, 2004.

[HA06] S. Harizopoulos and A. Ailamaki. Improving Instruction Cache Performance in OLTP. *ACM Transactions on Database Systems*, 31:887–920, September 2006.

[IATAM13] P. Tözün I. Atta, X. Tong, A. Ailamaki, and A. Moshovos. STREX: Boosting Instruction Cache Reuse in OLTP Workloads Through Stratified Transaction Execution. In *Proceedings of the 40th Annual International Symposium on Computer Architecture, ISCA*, pages 273–284, 2013.

[KADS03] P. Kundu, M. Annavaram, T. Diep, and J. Shen. A Case for Shared Instruction Cache on Chip Multiprocessors Running OLTP. In *Proceedings of the 2003 Workshop on MEmory Performance: DEaling with Applications, Systems and Architecture, MEDEA*, pages 11–18, 2003.

[LBE+98] J. L. Lo, L. A. Barroso, S. J. Eggers, K. Gharachorloo, H. M. Levy, and S. S. Parekh. An Analysis of Database Workloads Performance on Simultaneous Multithreaded Processors. In *Proceedings of the 25th Annual International Symposium on Computer Architecture, ISCA*, pages 39–50, 1998.

[WFM10] N. B. Williams, C. Fensch, and S. Moore. Proximity Coherence for Chip Multiprocessors. In *Proceedings of the 19th International Conference on Parallel Architectures and Compilation Techniques, PACT*, pages 123–134, 2010.

[ZA05] M. Zhang and K. Asanovic. Victim replication: Maximizing capacity while hiding wire delay in tiled chip multiprocessors. In *Proceedings of the 32nd Annual International Symposium on Computer Architecture, ISCA*, pages 336–345, 2005.

Particle-in-Cell algorithms on DEEP: The iPiC3D case study

Anna Jakobs[1], Anke Zitz[1], Norbert Eicker[1], Giovanni Lapenta[2]

[1]Forschungszentrum Jülich
Jülich Supercomputing Centre
D-52425 Jülich
a.jakobs@fz-juelich.de
a.zitz@fz-juelich.de
n.eicker@fz-juelich.de
[2]Katholieke Universiteit Leuven
BE-3001 Heverlee
giovanni.lapenta@wis.kuleuven.be

Abstract: The DEEP (Dynamical Exascale Entry Platform) project aims to provide a first implementation of a novel architecture for heterogeneous high-performance computing. This architecture consists of a standard HPC Cluster and – tightly coupled – a cluster of many-core processors called Booster. This concept offers application developers the opportunity to run different parts of their program on the best fitting part of the machine striving for an optimal overall performance. In order to take advantage of this architecture applications require some adaption. To provide optimal support to the application developers the DEEP concept includes a high-level programming model that helps to separate a given program to the Cluster and Booster parts of the DEEP System. This paper presents the adaption work required for a Particle-in-Cell space weather application developed by KULeuven (Katholieke Universiteit Leuven) done in the course of the DEEP project. It discusses all crucial steps of the work starting with a scalability analysis of the different parts of the program, their performance projections for the Cluster and the Booster leading to the separation decisions for the application and finally the actual implementation work. In addition to that some performance results are presented.

1 Introduction

Even though today's supercomputer systems reach multi-Petaflop compute power (examples are: Tianhe-2 at Guangzhou, Titan at Oak Ridge National Lab, IBM Sequoia at LLNL, or JSC's JUQUEEN) the HPC community prepares for the next step, i.e. having Exascale systems (10^{18} floating-point operations per second) by the end of the decade. The DEEP (Dynamical Exascale Entry Platform) project[1] aims to develop a novel, Exascale-enabling supercomputing platform, the Cluster-Booster architecture. On this platform application developers can map their code onto two diverse parts of the system supporting the different demands of scalability of their programs in an optimized way. Those two parts are: (1) a Cluster of multi-core Xeon processors interconnected by an InfiniBand network with a fat-tree topology; and (2) a second cluster of self-hosted Xeon Phi many-core processors called Booster. The latter utilizes the EXTOLL network [2] supporting a 3D torus topology. A comprehensive software environment including low-level communication libraries, programming environments and run-time systems complete the system.

As an assessment of this novel supercomputer architecture six scientific pilot applications were chosen within DEEP. In the course of the project these applications are ported to the platform acting as the yard-stick to evaluate the software environment and to act as benchmarks of the overall architecture. They have been selected with regard to their high scientific, industrial and social relevance and the urgent need for Exascale compute power in their research fields. Furthermore, the existence of highly scalable parts in their code that shall profit from the Booster was crucial. This paper will focus on the iPiC3D application from the Katholieke Universiteit Leuven. iPiC3D is a Particle-in-Cell space weather simulation predicting conditions of the magnetized plasma that permeates the space between Sun and Earth as well as the whole solar system.

The actual steps performed in the first three years of the DEEP project are explained in detail in the course of this paper. As a first step this includes a detailed analysis of the application with different performance tools. Next a plan to separate the code in a Booster and a Cluster part based on this analysis was created. It serves as a basis to finally implement this separation. To help the application developers to perform the separation the OmpSs runtime[3] – developed by BSC (Barcelona Supercomputing Center) and extended in the DEEP project – was used.

The paper is organized as follows: As a first step we motivate the division of applications by the description of the Cluster-Booster architecture in section 2. Next we give a short overview of the iPiC3D application including the analysis results from the beginning of the DEEP project in section 3. After a brief summary of Xeon Phi optimization strategies in section 4 we focus on the actual work done for adapting the code to the DEEP system and the resulting problems in section 5. Finally we present some performance results in section 6. The last section gives a short conclusion and an outlook on the next steps.

2 Cluster-Booster architecture

On the way to an Exascale supercomputer the DEEP project pursues the strategy of a new architecture consisting of a Cluster part and a Booster part. This architecture represents an alternative approach to organize heterogeneity in high-performance computing. As sketched in figure 1 the Cluster nodes utilize Intel Xeon multi-core processors and utilize an InfiniBand fabric for communication. In contrast to that, the Booster is based on Xeon Phi's many-core processors interconnected by an EXTOLL fabric developed at University of Heidelberg. A so called Booster Interface bridges between the two different interconnects of Cluster and Booster and allows for a most efficient communication between the two parts of the system.

The general idea of the Cluster-Booster concept is based on the observation that complex

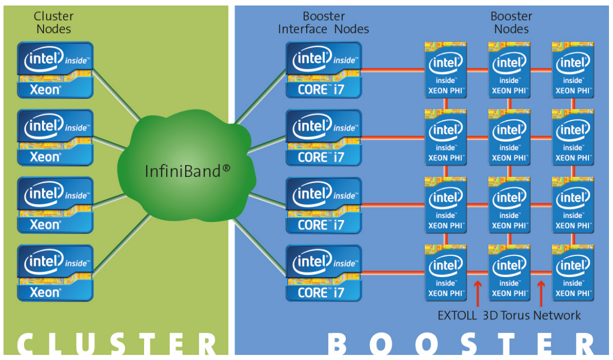


Figure 1: DEEP architecture scheme

applications in HPC often have code-parts with differing ability to scale on parallel machines. In order to utilize the DEEP system in a most efficient way, the less scalable code parts and the I/O operations are run on the Cluster while highly scalable parts are offloaded from the main application to the Booster. The DEEP system, in contrast to today's GPU-based heterogeneous systems, allows for communication between the processes offloaded to the Booster Nodes. This gives the application developers the chance to offload more complex kernels; code parts with intensive collective communications should be executed on the Cluster instead of the Booster anyway.

In order to support application developers to port their applications to this unconventional heterogeneous architecture the DEEP project develops a rich software infrastructure. While its basic mechanisms rely on MPI and its `MPI_Comm_spawn` functionality to start additional processes within the system, special emphasis was taken to relief the application programmer from having to re-organize the code manually. For this OmpSs, an OpenMP based data-flow programming model with directives to support asynchronous parallelism and heterogeneity, is extended allowing the user to just annotate the code leaving the main work to the Mercurium source-to-source compiler and the Nanos++ runtime system.

Of course, separating applications into two parts and distributing them to the Cluster and Booster parts of the DEEP system might introduce new bottle-necks if the parts have to exchange data. Therefore, it is crucial to split the application in a way that the amount of data to be exchanged is minimized. In addition to that a highly optimized Cluster-Booster protocol was introduced into the DEEP software stack reducing the overhead of the necessary bridging between the two fabrics as much as possible.

Since this paper will not explain neither the overall concept nor the hardware or software architecture in more detail, the interested reader is referred to [4] for more information.

3 iPiC3D application

The iPiC3D application of KULeuven is a massively parallel code to simulate the evolution of a magnetized plasma traveling from the Sun to the Earth. This topic is highly relevant in order to forecast space weather related events which may lead to severe problems like damage to spacecraft electronics, GPS signal scintillation or even disturbance of wide-area power grids. There are different approaches to model the plasma evolution; the iPiC3D application implements a kinetic approach wherein both ions and electrons are simulated as particles. iPiC3D is written in C++ and was parallelized using MPI before the beginning of the DEEP project. At the current state of work also OpenMP is used allowing for an hybrid parallelization. An application run consists of multiple time steps. For each of them particles are moved under the effect of the electric and magnetic fields defined on a discrete mesh in physical space.

Particle information is deposited on this spatial grid through interpolation procedures in the form of moments, i.e. densities, currents and pressures defined on the mesh, which act as sources for the equation solved for obtaining the fields at the next time step. More details on the actual algorithm are provided in [5].

The main part of iPiC3D is to calculate the evolution of the electric and magnetic fields and the positions and velocities of the computational particles in time. The particle information is averaged and collected as moments; these also have their part in evolving the electric field. During the analysis phase at the beginning of the DEEP project figure 2 was created, representing the logical structure of the application.

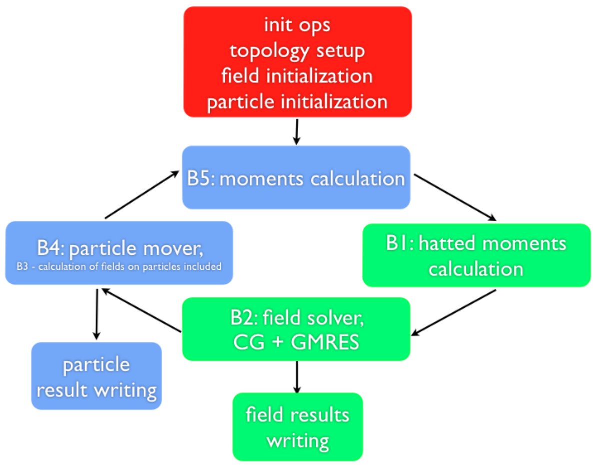


Figure 2: Logical structure of iPiC3D

The different parts can be shortly explained as follows:

B5: moments calculation: density, currents and pressures are calculated starting from particles.

B1: hatted moments calculations: the hatted moments, i.e the effective moments of interaction with the electric and magnetic fields, are calculated from the moments.

B2: field solver: the electric and magnetic fields are calculated for the new time step.

B4: particle mover: particles are moved under the influence of the newly calculated fields.

The green boxes of figure 2 are related to the fields and therefor the grid, the blue ones refer to the particles.

When describing the different phases we will concentrate on the moments calculation, the field solver and the particle mover, as they are the most time consuming or communication intensive parts of the application. This was shown by Scalasca profiling runs performed on the JUDGE cluster and the BlueGene/Q system JUQUEEN at Jülich Supercomputing Centre; the results are presented in figure 3.

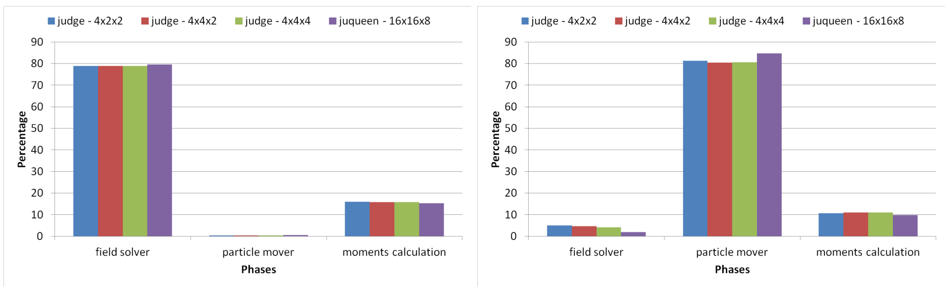


Figure 3: Percentage of execution time (left) and communication time (right) of the relevant phases for the JUDGE tests with $4 \times 2 \times 2$, $4 \times 4 \times 2$ and $4 \times 4 \times 4$ cores (small test case) and the JUQUEEN test with $16 \times 16 \times 8$ cores (big test case)

During the moments calculation, the particle information is gathered and accumulated in grid moments. After the collection the total moments of quantities like density or pressure are calculated. The nearest neighbors then exchange ghost node information using `MPI_Sendrecv_replace`. This phase is the second most relevant phase with regard to percentage of execution time, communication calls and bytes exchanged. Especially the collective communication of the ghost node information should be noted here.

In the field solver phase a Poisson correction of the electric field is performed and both the electric and the magnetic field are updated. This phase is not very time consuming, but is the most important one related to communication. As in the moment calculation the ghost node information is exchanged using point to point communication. Additional collective communication is required during each iteration of the solver in order to determine the stopping criterion.

Finally, the particle mover updates the particle positions and velocities. Since the particles are moved independently, this phase is highly scalable. After the movement all particles that are now located on a part of the grid hosted by a different processor have to be identified and exchanged; this is done with a series of point to point communications between nearest neighbors. In addition to that, the number of particles to be moved is assessed grid-wide through an `MPI_Allreduce`. These are the only few collective communications not done in the solver phase. This phase is essentially compute intensive.

The decision on how to divide the application into a Cluster and a Booster part was made on the basis of the following aspects:

- There are two different kind of phases, grid related (fields and moments) and particle related ones. They operate mostly on different data, so they should be kept on one part of the DEEP system each.
- The phases which have the most intensive collective communication should be kept on the Cluster, as these are implemented more efficiently on this part of the system. In the case of iPiC3D these are the grid related ones, i.e. moment and field calculations.
- The particle related phases can be vectorized more easily and have better scalability, as particles are processed independently; therefore, they fit better on the Booster part.

4 Xeon Phi optimization

For a most beneficial use of the DEEP Booster the application parts that will be launched there shall be optimized for the Xeon Phi processor. Different code changes were done to achieve this goal. They will be briefly explained within this section.

The most important step in the course of the optimization process is the introduction of OpenMP thread parallelization of the loops that process particles. Tests unveiled that using one MPI process for each hardware thread – 240 in total on a 60 core MIC – would introduce a significant communication overhead. On the other hand, with only 60 MPI processes of 4 OpenMP threads each the performance was about 2 to 4 times better.

A second approach was to introduce an improved localization of field data by using an array of `structs` (AoS) instead of a `struct` of arrays (SoA). Typically an SoA is preferred over an AoS when vectorizing code. Although, in this application using an AoS has considerable advantages like a faster sorting of particles (needed to eliminate random access) or a superior cache performance. As transpositions are rather cheap in this case it was decided to use an AoS for the basic particle representation and convert it to an SoA in blocks when it is beneficial for vectorization.

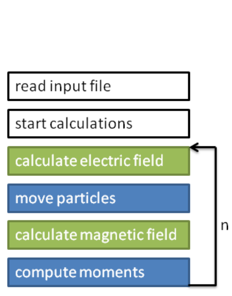
The next optimization step would be the vectorization of particle processing; for that step a sorting of the particle is needed, which has not yet been implemented for the code.

5 Offload adaption work

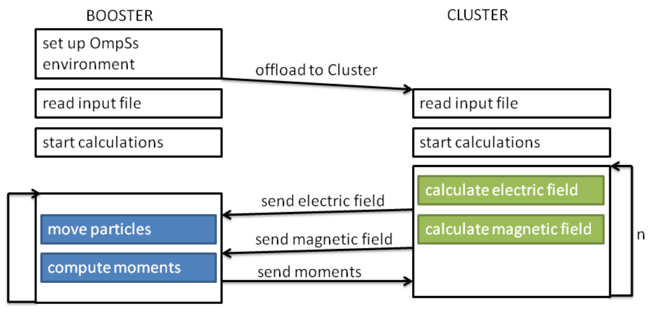
The application was logically divided into a part to run on the Booster and one to be offloaded to the Cluster[1] as can be seen in figure 2. This section explains the necessary steps performed to achieve a successful division.

5.1 Code division

The changes in the application workflow with the offload can be shown best along the flow charts in figure 4.



(a)



(b)

Figure 4: Workflow without (a) and with offload (b)

The original version of the application contains a single `run` function to start the whole simulation. In the offload version two corresponding functions are called, `run_Cluster` and `run_Booster`. For a clearer logical division the different calculation steps were divided between host and offload part as sketched in figure 4b and separated into the two functions. The necessary communication of the moments, the particles and the magnetic field between Cluster and Booster were added after the corresponding calculations. For the sending and receiving of data between host and offload the proper MPI communicators have to be used. In the offload part the parent communicator can be fetched via `MPI_Comm_get_parent`. This function returns an inter-communicator to the processes on the host part. For sending to or receiving from the offload on the host part the inter-communicator is used that was created and returned by the call of `deep_booster_alloc` (see section 5.2). To have access to this communicator in the relevant classes it is given as a parameter to the `run_Booster` function and from there forwarded to the communication functions.

[1]In fact, this is contrary to the basic concept described in section 2. Nevertheless, DEEP's software stack is flexible enough to support the reverse offloading used in iPiC3D, too.

5.2 OmpSs

As mentioned before iPiC3D was parallelized in a hybrid way using MPI and OpenMP. Utilizing the OpenMP offload was no option in the case of the DEEP project for two reasons: On the one hand this technology was not yet introduced in the OpenMP standard until almost two years in the project. On the other hand the OpenMP offload assumes a local target and does not allow for MPI-communication between offloaded processes. The latter is crucial when the offloaded tasks are computationally heavy as in the DEEP examples. Although OmpSs and OpenMP are similar in many aspects OmpSs does not support all OpenMP pragmas. All pragmas aside from `#pragma omp task` and `#pragma omp for` were commented in the OmpSs version of the application.

For using OmpSs the application has to be compiled with the Mercurium compiler developed by BSC[6]. Some minor changes in the code were necessary for that but will not be discussed further in this paper.

The next step was to integrate the call of the OmpSs function `deep_booster_alloc` and the offload pragma into the code. We are implementing a reverse offload here where we start the application on the Booster part and offload to the Cluster. This is due to the fact, that logically the application is build around the particle calculation, offloading the fields calculations fits better into the overall concept of the code (see figure 5).

```
MPI_Comm clustercomm;
MPI_Comm_rank(MPI_COMM_WORLD,&rank);
MPI_Comm_size(MPI_COMM_WORLD,&size);

deep_booster_alloc(MPI_COMM_WORLD, 1, size, &clustercomm);

//create offload task
#pragma omp task device(mpi) onto(clustercomm,rank) \
     in([...]) copy_deps
{
   solver.run_Cluster();
}

solver.run_Booster(clustercomm);
```

Figure 5: Integration of OmpSs in main part of the code

In the original code, the input parameters were read from the input file, the necessary objects were created and the `run` function was called to start the calculations. It was planned to give the objects essential for the calculations to the offload as input parameters, but this was not feasible due to OmpSs not supporting the offload of C++ objects. A serialization and deserialization of all necessary objects seemed to be a solution, but with some testing this approach turned out to be way to complicated, as too many objects were needed.

Therefore it was decided to recreate the whole environment in the offload part. For this approach, only `argc` and `argv` are crucial, as `argv` contains the name of the input file, which is sufficient for establishing the whole setup in the offload part. `argv` is now serialized and given as an input parameter to the offload pragma. Generally speaking, the whole calculation is set up and started both on the host and in the offload part.

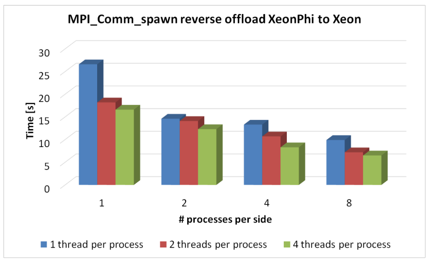
5.3 MPI_Comm_spawn

It has to be taken into account that the OmpSs runtime and the Mercurium compiler are still under development as part of the DEEP project; during the months of work shown in this paper they were constantly enhanced but issues like a significantly decrease of application performance or compiling problems appeared from time to time that needed investigation and hindered us to get detailed and meaningful performance results. To avoid these issues for the moment and to have a possibility for later comparison (mostly to see if using OmpSs creates overhead) yet another version of the application was implemented. This variant uses `MPI_Comm_spawn` explicitly for offloading. In contrast to that OmpSs uses this function, too; nevertheless the actual calls are hidden from the application developers. It required only minor changes in the code to create this version; mainly the OmpSs offload pragma was replaced by the call of `MPI_Comm_spawn`. Additionally the application was compiled directly with the Intel compiler without performing a source to source compilation with Mercurium beforehand. The main benchmarks were performed with this version of iPiC3D to analyze the general behavior of the application when offloading from Xeon Phi to Xeon. Lately, some tests revealed that with the newest versions of OmpSs the issues of a decrease of performance were solved and the overhead compared to the `MPI_Comm_spawn` has mostly vanished; future experiments will give specific numbers.

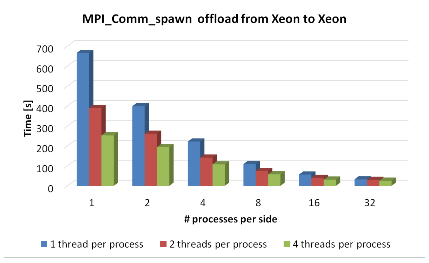
6 First results

During the adaption work the application was mainly tested on a KNC system installed at JSC with two compute nodes. This was done due to lack of the actual Booster hardware. The node that was used consists of two Xeon ES-2670 processors with 8 cores clocked at 2.6 GHz and four Xeon Phis 7120 co-processors with 61 cores each running at 1.23 GHz. Each Xeon Phi is equipped with 16 GB of memory. As it was decided to use a reverse offload model, the application was started on the Xeon Phis and offloaded to the Xeons. In addition to that tests were performed on the DEEP Cluster, offloading from Xeon to Xeon (Intel Xeon E5-2680 at 2.70 GHz).

Two different test cases were simulated. The smaller one contains 460,800 particles and was used for performance runs on the smaller KNC system. The larger test case of about 93 million particles was performed on the DEEP Cluster, taking advantage of the bigger system. In both cases 30 cycles were simulated.



(a)



(b)

Figure 6: Execution time on the KNC system with small test case (a) and on the DEEP Cluster with larger test case (b)

Figure 6a shows the performance results generated on the KNC system with the small test case, in figure 6b the execution times from the DEEP Cluster simulating the larger test case are shown.

Especially for the larger test case the application scales quite well with an increasing number of MPI processes. For the smaller example, the gain of the faster calculation is likely too small to hide the communication for more than 2 processes. A larger test case with more intense calculations should lead to a better scalability but couldn't be tested on the KNC system due to memory restrictions. One should to take into account that these performance numbers were created shortly after integrating the offload in iPiC3D; therefore they should be seen as first evaluation of the application behavior.

7 Conclusion and outlook

At the current point of the project, both the OmpSs and the `MPI_Comm_spawn` reverse offload are fully integrated in the iPiC3D application. The code was separated into different functions for the Cluster and the Booster part. Starting on the Booster, the parts of the code which operate on the moments or fields are offloaded to the Cluster. The minimal required communication between Cluster and Booster to exchange the fields and moments was set up.

Additional tests were performed on a bigger system, showing promising results. The next step will be to run a large set of benchmarks on the whole DEEP system as soon as possible.

As illustrated in this paper, working on a highly experimental project can lead to unforeseen challenges that might influence the outcome of the whole project. E.g. due to the late availability of hardware the applications could not be tested on the full DEEP system in time. Instead it had to be settled for a smaller test system for Xeon Phi performance evaluations and the DEEP Cluster for larger tests on Xeon cards; but these systems are able to give a prospect on what to expect from the final system and were used extensively for optimizing the applications for the two different kinds of architectures.

With hindsight it probably would have been more efficient to implement the offload in iPiC3D directly with `MPI_Comm_spawn` instead of trying OmpSs first, as the concept of OmpSs to offload multiple tasks with input and output dependencies does not fit the structure of the application in an optimal way. Nevertheless, it revealed some issues in the OmpSs runtime which could be solved this way, leading to more efficient and better performing versions in the course of the last months.

8 Acknowledgments

The authors would like to thank the people and partners involved in the DEEP consortium for their ongoing engagement and strong commitment towards the project, which led to several of the results described in this paper.

The research leading to these results has been conducted in the frame of the DEEP project, which receives funding from the European Commission's Seventh Framework Programme (FP7/2007-2013) under Grant Agreement n° 287530.

This article is dedicated to the memory of Alec Johnson who passed away in an accident right before Christmas 2014. He contributed significant parts of this work. He will remain in the memories of the authors and all collaborators of the DEEP project.

References

[1] `http://www.deep-project.eu`

[2] M. Nüssle, B. Geib, H. Fröning, and U. Brüning, "An FPGA-based custom high performance interconnection network", In "Proceedings of the 2009 International Conference on Reconfigurable Computing and FPGAs", 2009

[3] `http://pm.bsc.es/ompss`

[4] Norbert Eicker, Thomas Lippert, Thomas Moschny, and Estela Suarez, "The DEEP project - Pursuing cluster-computing in the many-core era", 42nd International Conference on Parallel Processing (ICPP), p. 885 - 892, 2013

[5] G. Lapenta, J. U. Brackbill und P. Ricci, "Kinetic approach to microscopic-macroscopic coupling in space and laboratory plasmas", Physics of Plasmas, Vol. 13, Nr. 5, pp. 055904-055912, 2006.

[6] `http://pm.bsc.es/mcxx`

[7] S. Markidis, G. Lapenta und R. Uddin, "Multi-scale simulations of plasma with iPiC3D", Mathematics and Computers in Simulation, Vol. 80, Nr. 7, p. 1509 - 1519, 2010.

High performance CCSDS image data compression using GPGPUs for space applications

Sunil Chokkanathapuram Ramanarayanan[a], Kristian Manthey[b], Ben Juurlink[a]

[a]Technische Universität Berlin
Embedded Systems Architecture
Einsteinufer 17
10587 Berlin, Germany

[b]German Aerospace Center (DLR)
Institute of Optical Sensor Systems
Optical Sensors and Electronics
Rutherfordstraße 2
12489 Berlin, Germany

Abstract: The usage of graphics processing units (GPUs) as computing architectures for inherently data parallel signal processing applications in this computing era is very popular. In principle, GPUs in comparison with central processing units (CPUs) could achieve significant speed-up over the latter, especially considering data parallel applications which expect high throughput. The paper investigates the usage of GPUs for running space borne image data compression algorithms, in particular the CCSDS 122.0-B-1 standard as a case study. The paper proposes an architecture to parallelize the Bit-Plane Encoder (BPE) stage of the CCSDS 122.0-B-1 in lossless mode using a GPU to achieve high throughput performance to facilitate real-time compression of satellite image data streams. Experimental results are furnished by comparing the performance in terms of compression time of the GPU implementation versus a state of the art single threaded CPU and an field-programmable gate array (FPGA) implementation. The GPU implementation on a NVIDIA® GeForce® GTX 670 achieves a peak throughput performance of 162.382 $^{\text{Mbyte}}/_{\text{s}}$ (932.288 $^{\text{Mbit}}/_{\text{s}}$) and an average speed-up of at least 15 compared to the software implementation running on a 3.47 GHz single core Intel® Xeon™ processor. The high throughput CUDA implementation using GPUs could potentially be suitable for air borne and space borne applications in the future, if the GPU technology evolves to become radiation-tolerant and space-qualified.

1 Introduction

The spatial as well as the spectral resolution of air borne and space borne image data increases steadily with new technologies and user requirements resulting in higher precision and new application scenarios. On the technical side, there is a tremendous increase in data rate that has to be handled by such remote sensing systems. While the memory capacity requirements can still be fulfilled, the transmission capability becomes increasingly

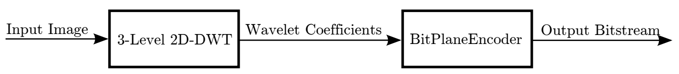


Figure 1: General Schematic of the CCSDS 122.0-B-1 Encoder [Con07].

problematic. The communication bandwidth is always an expensive entity and is limited by the radio channel and also the visibility of the ground control station (GCS) from the orbit. Moreover, the key requirement is to compress huge amounts of data in real-time and transmit them in as little time as possible to the GCS. These stringent requirements mandate a high performance real-time on-board compressor to attain high compression throughput of the orders of $^{\mathrm{Mbyte}}/_{\mathrm{s}}$. The generic commercial off-the-shelf (COTS) CPU technologies cannot be used for reliability reasons, space qualification criteria, and high throughput requirements. GPUs, by virtue of thread level parallelism (TLP) easily cater to the requirements of the massive data parallel image-processing applications. However, GPUs haven't yet been space-qualified as of now.

The Consultative Committee for Space Data Systems (CCSDS) compression standard exhibits data parallelism inherently in its encoding stages. It comprises of 2 major stages namely, discrete wavelet transform (DWT) and BPE as illustrated in the Fig. 1. The BPE stage is as computationally demanding as the DWT or more depending on the input images. This paper focuses on parallelizing the BPE stage of the CCSDS 122.0-B-1 standard using GPU. The CCSDS 122.0-B-1 compression standard operates in lossy/lossless modes based on the quality parameters. The lossless mode is expected to run through the entirety of the algorithm without exiting intermediately, to produce the worst case execution time in comparison to the lossy mode which could be configured to exit at different stages based on the quality parameters. This paper focuses on the lossless mode to analyze the worst case execution time of the encoder. Also, it could also be safe enough to assume lossless or near-lossless compression to be obligatory in several cutting-edge scientific missions which refuse the ideology of abiding by lossy compression.

The paper is organized as follows. Chapter 2 describes related work concerned to the image processing solutions for space applications, prior work pertaining to comparable compression standards on General-Purpose Computation on Graphics Processing Units (GPGPUs) to understand upfront about the state of the art technologies and research. Chapter 3 explains the fundamentals of CCSDS 122.0-B-1 image data compression standard. Chapter 4 proposes the design and development of a GPGPU-based BPE stage of CCSDS 122.0-B-1 compressor. It explains the various design decisions; porting, parallelization, thread mapping strategies and optimizations done for the GPGPU solution. Chapter 5 comprehensively analyzes the performance benchmarks of the GPGPU implementation against the FPGA and host counterparts. Chapter 6 summarizes the results and discusses about the pros and cons of the GPGPU solution. Finally, chapter 7 summarizes the findings of the paper.

2 Related work

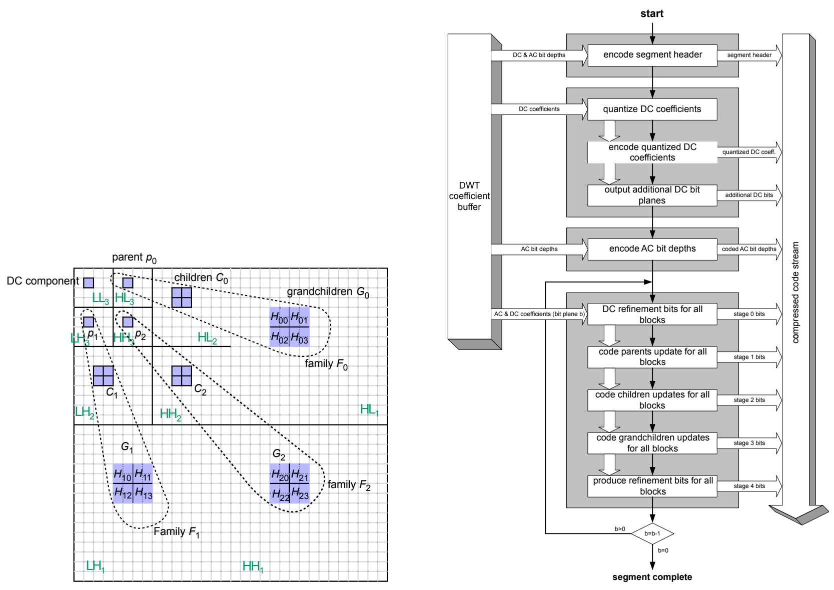
Many image compression standards such as JPEG2000 [Kur12], SPIHT [SLH11, LBM11] to name a few, have been tried on GPGPUs due to the presence of inherent data-parallelism. The DWT is an integral part of most of the image compression standards and is completely data parallel wherein every pixel could be independently processed. The following table 1 lists the prior DWT implementations on GPUs. Shifting focus to the CCSDS compression standards, [KAH+12] proposes a GPGPU-based Fast Lossless hyper-spectral image compressor which achieves a throughput of 583.08 Mbit/s and a speed-up of 6 times in comparison with 3.47 GHz single core Intel® Xeon™ processor. Having seen significant research in the DWT implementations on GPGPU, the paper focuses on parallelizing the BPE stage of the CCSDS 122.0-B-1 standard in lossless mode. This paper also comparatively analyzes the GPGPU implementation with the hardware FPGA implementation of the CCSDS 122.0-B-1 standard by [MKJ14]. The FPGA implementation achieves an average throughput of 238.274 Mbyte/s.

Related works	**Speedup** (X times w.r.t host)	**Reference CPU configuration**	**GPU card**
GPU-Based DWT Acceleration for JPEG2000[Mat09]	148	Intel Core i7, 3.2GHz, 3 × 2GB RAM	NVIDIA Geforce GTX295
A novel parallel Tier-1 coder for JPEG2000 using GPUs[LBM11]	100	Intel Core i7, 2.8GHz, 12GB RAM	NVIDIA Geforce GTX480
A GPU-Accelerated Wavelet Decompression System with SPIHT[SLH11]	158	2 Intel quad-core Xeon E5520, 2.27 GHz, 16GB RAM	NVIDIA Tesla C1060

Table 1: Wavelet transform implementations using GPGPUs

3 CCSDS 122.0-B-1 image data compression

CCSDS 122.0-B-1 image data compression standard [Con05, YAK+05] is a single-band compression technique and has been recently used for image data compression on-board spacecraft. It can compress 16 bit signed and unsigned integer images in lossless as well as in lossy mode. At first, the DWT module applies a 3-level 2D-DWT on the input image. The lossless "Integer DWT" implementation is considered in this paper to ensure that the original image is perfectly reconstructed. The DWT module forms a hierarchy of wavelet coefficients as shown in Fig. 2a. A *block* is a group of one DC coefficient and the 63 corresponding AC coefficients (3 parents, 12 children, 48 grandchildren). A block loosely represents a region in the input image. For the BPE, the blocks are further arranged into groups: A segment is a group of S consecutive blocks, where $16 \leq S \leq 2^{20}$. Segments are encoded independently and are further partitioned into *gaggles*, which is a group of $G = 16$ consecutive blocks. Once all the coefficients are grouped, the BPE starts to encode the image segment-wise (see Fig. 2b). Each segment starts with a *segment header* containing information about the current segment. After the segment header is written, the DC coefficients are quantized with a quantization factor q that depends on the wavelet



(a) A block consisting of a DC coefficient and 63 AC coefficients [Con05].

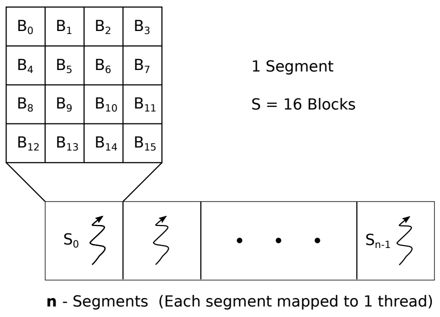
(b) Program and data flow of BPE [Con07]

Figure 2: Overview of the CCSDS 122.0-B-1 BPE

transform type and on the dynamic range of the wavelet coefficients. In a next step, differential pulse code modulation (DPCM) is applied on the quantized DC coefficients and is followed by Rice Coding. After all quantized DC coefficients are encoded, some additional DC bit-planes may be refined. The next step is to encode the bit-depth of the AC coefficients in each block with the same DPCM method. The BPE encodes the wavelet coefficients bit-plane-wise and in decreasing order. For each bit-plane, the encoding process is divided into stages 0–4. In stage 0, remaining bits of the DC coefficients are coded (DC refinement). Stage 1–3 encode the AC coefficients' sign and the position of the *significant bit*, which is the highest non-zero bit. Stage 1 refers to the refinement of the parents coefficients. The same procedure is applied to the children coefficients at stage 2 and to the grandchildren coefficients at stage 3. Stages 1–3 produce words which are first mapped to symbols which are then encoded with variable-length code (VLC). Once an AC coefficient is selected, Stage 4 encodes the AC coefficients' refinement.

4 Design

The goal of this paper is to parallelize the segment-wise BPE stage of the CCSDS 122.0-B-1 standard. Each segment could be executed mutually exclusive of the other. The BPE loop nest could be visualized wherein the DWT processed pixels are partitioned into NUM_SEGMENTS segments. The 1D loop is mapped onto a 1D thread grid. Each thread



B_0 B_1 B_2 B_3
B_4 B_5 B_6 B_7
B_8 B_9 B_{10} B_{11}
B_{12} B_{13} B_{14} B_{15}
1 Segment
S = 16 Blocks
S_0
S_{n-1}
n - Segments (Each segment mapped to 1 thread)

Figure 3: Thread mapping for the segment-wise BPE on GPU

processes one segment of the encoder as illustrated in Fig. 3. Each segment comprises of S blocks where $S = 16$ is chosen as a fixed parameter for this architecture. The reasons for this configuration shall be explained subsequently.

4.1 Thread mapping

Compute Unified Device Architecture (CUDA) allows to split the work in terms of threads, wherein a group of threads constitutes a thread block and further, a group of thread blocks forms a thread grid. The choice for these thread configurations is directly dependent on the loop index which is in this case the number of segments NUM_SEGMENTS. Also, the NVDIA architecture influences the potential size of the thread block. The threads within a thread block are further divided into groups of 32 threads called *warps* which are eventually scheduled by the *warp scheduler* onto the streaming multiprocessors (SMX). The NVIDIA GK104 SMX Kepler architecture has 4 warp schedulers to pick 4 active warps per clock cycle and dispatches them to the execution units. Hence it is always customary for each thread block to have at least 4 warps to ensure peak utilization of the SMX. Hence the number of threads within a thread block is chosen to be $32 \times 4 = 128$. Consequently, the number of thread blocks would be as follows.

```
int blocksPerGrid = (NUM_SEGMENTS/128) + 1;
```

4.2 Choice of encoder parameters

The CCSDS 122.0-B-1 standard is configured with quality parameters to operate in lossless mode in order to analyze worst case behavior and to facilitate perfect reconstruction of the input images. Fig. 4 shows the impact of the segment size S on the compression efficiency in lossless mode. Lesser the bits per pixel consumed by the compressor, better the compression efficiency. It can be noted that the value of S has minimal or no impact on the compression efficiency. Since the encoder is ported onto a GPGPU and the BPE

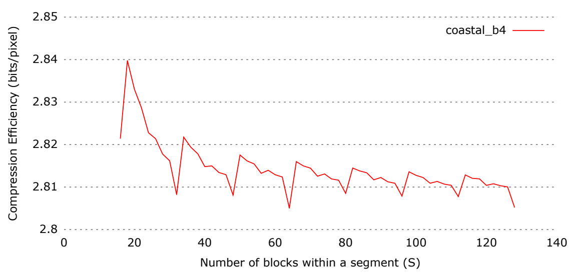


Figure 4: Impact of the segment size S on the compression efficiency

inherently has a lot of loop nests with loop index S, it is intuitive to reduce the value of S in order to reduce the branch latency considering the fact that GPUs perform poorly with branches due to branch divergence within the warps. Moreover, lesser the number of blocks within a segment, greater will be the total number of segments to process in the BPE stage. Therefore, minimizing the value of S results in achieving maximum possible value for NUM_SEGMENTS and thereby increasing the degree of parallelism. Hence, the segment size S is set to the least possible value of 16.

4.3 Concatenating the output bit-stream buffer

Since each segment is processed independently by different threads, the individual bit-streams generated by each thread have to be combined to produce the final output bit-stream buffer. Moreover, since dynamic allocation of the memory is impossible in GPU address space unlike the reference CPU implementation, the buffers have to be statically allocated up-front. In order to determine the size of the buffers, it is safe to assume that the compressed output bit-stream size shall not exceed the input barring the exceptions of high entropy images such as noise images. The input buffer size for each segment is $S \times 64 \times 4\,\mathrm{byte}$, as each segment contains S blocks and each block consists of 64 coefficients wherein the dynamic range of each wavelet coefficient does not exceed 20 bit. Hence the output bit-stream buffer size for each segment is also assumed to not exceed this limit for regular images. This concatenation process is performed on the host side after the CUDA kernel has completed its execution. Fig. 5 illustrates the process of the generation of the output bit-stream.

4.4 Optimizations

4.4.1 L1 Cache configuration

The NVIDIA GK104 SMX Kepler architecture offers a 64 kbyte unified memory subsystem useable as shared memory and also as L1 cache. In general, shared memory is

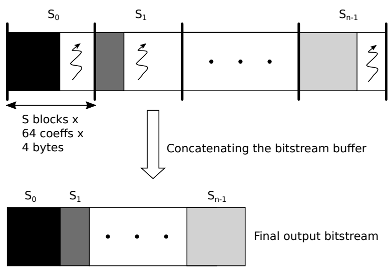


Figure 5: Output bitstream generation

configured for 48 kbyte and the remaining 16 kbyte behaves as L1 cache. By virtue of a lot of local variables in the BPE encoder, it is preferable to have a bigger L1 cache rather than the shared memory. Also, due to the fact that the input data set could not fit onto the shared memory, it was decided to use the bulk of the faster memory subsystem to act as L1 cache. The following CUDA API allows the programmer to set the preference for a 48 kbyte L1 cache.

```
// Prefer L1 cache of 48 kbyte instead of 16 kbyte
cudaFuncSetCacheConfig(encode, cudaFuncCachePreferL1);
```

4.4.2 Reduced Global memory accesses

The accesses to the global memory of the GPU is always expensive and keeping this in mind, repeated accesses to the same global memory variables were avoided by fetching them only once and efficiently rearranging the dependent computations.

5 Analysis

This section provides the results of the GPGPU implementation of the CCSDS 122.0-B-1 BPE and also compares it with the state of the art reference and FPGA implementations.

5.1 Methodology

5.1.1 Experimental setup

The reference implementation of the encoder shall be run on the state of the art 3.47 GHz single core Intel® Xeon™ for obtaining a host profile. The state of the art FPGA implementation from [MKJ14] is used for comparative analysis. The GPU used shall be NVIDIA® GeForce® GTX 670 launched in 2013 as a GPGPU platform to demonstrate the prototype. In a nutshell, the GPU card contains 7 NVIDIA Kepler GK104 SMX, each with 192 CUDA cores to sum up to a total of 1344 cores. Also, the presence of 2048 Mbyte of global GPU memory ensures a significant storage to fit the large input image data of the orders of $16\,\mathrm{k} \times 32\,\mathrm{k}$ pixel words.

5.1.2 Scope

The conformance single spectrum image set specified by the CCSDS 122.0-B-1 standard [Con07] is used to test the correctness of the encoder. Four images having different pixel bit depths namely $coastal\_b4$, ice_2kb4, foc and sar are chosen in this paper in order to have an extensive coverage of the encoder behavior. The encoded bit-streams are validated by decoding using a reference implementation of the CCSDS 122.0-B-1 decoder on the host machine. The input raw image data to the encoder and the output bit-stream from the decoder is checked for bit-wise match to ensure lossless reconstruction of the input image.

5.1.3 Profiling tools

The NVDIA command line profiler *nvprof* is used to measure the execution times of the CUDA GPGPU implementation running on NVIDIA GTX 670 GPU. The total execution time measured includes the CUDA kernel execution time as well as the host/GPU to/from memory transfers. The Linux clock/time API is used for obtaining the host profile of the CPU implementation. The FPGA profiling results are obtained using the cycle-accurate ModelSim™ simulator.

5.2 Impact of image sizes on performance

As already described in subsection 4.2, the parallelism is directly dependent on the value of the NUM_SEGMENTS. For normal sized images in the order 1024×1024 pixels, the NUM_SEGMENTS is not significantly high to utilize the GPU cores to good effect. Hence in order to improve the GPU utilization, the input image has to be big enough to ensure high occupancy of the cores. Therefore, for benchmark purposes, bigger images are generated by concatenating the entire original image as a linear buffer n times. Thus, the resultant image obtained is n times the original image size, thereby increasing the occupancy of the GPU cores by virtue of increased value of NUM_SEGMENTS. Four sets of images are created for $n = 1, 4, 16, 64$ for the performance analysis. Here, $n = 1$ represents the original image as is. Fig 6 shows that greater the image size is, better is the achieved throughput.

5.3 Performance Benchmarks

The throughput analysis is performed on the large sized images with $n = 64$ case on CPU, GPGPU and FPGA platforms. It could be observed in Fig. 7, that the GPGPU imple-

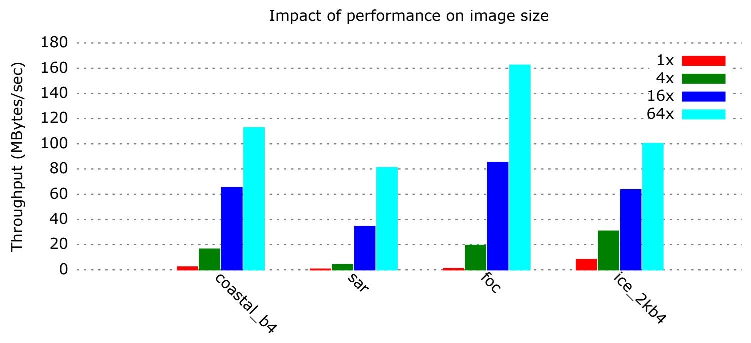


Figure 6: Comparison of Throughput based on Image sizes

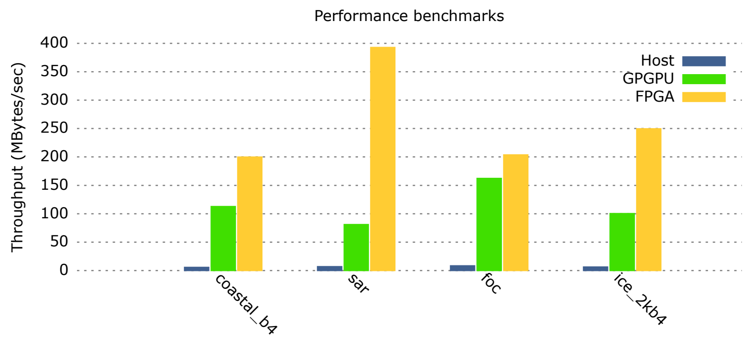


Figure 7: Throughput comparison between CPU, GPGPU and FPGA implementations

mentation achieves better throughput than the reference CPU implementation. However, the FPGA hardware implementation [MKJ14] achieves the best throughput in comparison with the CPU and GPGPU implementations.

6 Discussion

GPGPUs as the name suggests could be used for general purpose computation and isn't tightly bound to the application. GPUs could be used as a platform to run several different applications at different instants of time. The software programmability of the GPUs offers a high degree of flexibility in terms of application adaptation. GPUs also offer good backward compatibility. If an algorithm changes, the new software could run on older chip sets. The current GPU technologies have not proven to be space-qualified and radiation-tolerant. But, GPUs could easily be used in low Earth orbits and aeronautics.

7 Conclusion

The high-resolution satellite imaging systems require real-time image compressors on-board due to limited storage to store uncompressed raw data and/or also to conserve communication bandwidths. Hence it is mandatory for the image compressors to guarantee high compression throughput in the order of several $^{\mathrm{Mpx}}/_{\mathrm{s}}$. GPUs owing to their massive number of computing cores is investigated as a potential architecture platform to run the CCSDS 122.0-B-1 image data compression standard in lossless mode. The BPE stage of the standard was parallelized on GPU to satisfy the high compression throughput requirements. The parallelized BPE achieved an average speed-up of 16.718 times the host CPU implementation. The GPGPU solution is approximately 2.59 times slower in comparison to the state of the art hardware FPGA solution. This paper explores the possibilities of usage of GPU technologies for space applications provided they become radiation-tolerant and space-qualified in the future.

References

[Con05] Consultative Committee for Space Data Systems (CCSDS). Image Data Compression - Blue Book, 2005.

[Con07] Consultative Committee for Space Data Systems (CCSDS). Image Data Compression - Green Book, 2007.

[KAH$^+$12] D. Keymeulen, N. Aranki, B. Hopson, A. Kiely, M. Klimesh, and K. Benkrid. GPU lossless hyperspectral data compression system for space applications. In *2012 IEEE Aerospace Conference*, pages 1–9. IEEE, March 2012.

[Kur12] Krzysztof Kurowski. Graphics processing unit implementation of JPEG2000 for hyperspectral image compression. *Journal of Applied Remote Sensing*, 6(1):061507, June 2012.

[LBM11] Roto Le, Iris R. Bahar, and Joseph L. Mundy. A novel parallel Tier-1 coder for JPEG2000 using GPUs. In *2011 IEEE 9th Symposium on Application Specific Processors (SASP)*, pages 129–136. IEEE, June 2011.

[Mat09] J Matela. GPU-Based DWT Acceleration for JPEG2000. In *MEMICS 2009 Proceedings*, pages 136–143, Brno, 2009.

[MKJ14] Kristian Manthey, David Krutz, and Ben Juurlink. A new real-time system for image compression on-board satellites. *ESA/CNES On-Board Payload Data Compression (OBPDC), Venice, Italy*, 2014.

[SLH11] Changhe Song, Yunsong Li, and Bormin Huang. A GPU-Accelerated Wavelet Decompression System With SPIHT and Reed-Solomon Decoding for Satellite Images. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, 4(3):683–690, September 2011.

[YAK$^+$05] Pen-Shu Yeh, P. Armbruster, A. Kiely, B. Masschelein, G. Moury, C. Schaefer, and C. Thiebaut. The new CCSDS image compression recommendation. In *Aerospace Conference, 2005 IEEE*, pages 4138–4145, March 2005.

Parallelization of the Particle-in-cell-Code PATRIC with GPU-Programming

Jutta Fitzek

GSI Helmholtzzentrum für Schwerionenforschung
Planckstraße 1
64291 Darmstadt
j.fitzek@gsi.de

Abstract: The Particle-in-cell (PIC) code PATRIC (Particle Tracking Code) is used at the GSI Helmholtz Center for Heavy Ion Reasearch to simulate particles in circular particle accelerators. Parallelization of PIC codes is an open research field and solutions depend very much on the specific problem. The possibilities and limits of GPU integration are being evaluated. General GPU aspects and problems arising from collective particle effects are put into focus with an emphasis on code maintainability and reuse of existing modules. The studies have been performed using NVIDIA®'s Tesla C2075 GPU. This contribution summarizes the findings.

1 Introduction

Computer simulations play an important role in physics research to complement or replace experiments. This contribution focuses on simulations for circular particle accelerators at the GSI Helmholtz Center for Heavy Ion Reasearch, Darmstadt, Germany. Particle accelerators are used in physics to investigate the structure of matter. In this particular application, simulations are used to study the impact of parameter variations on the particle motion that are not easily measurable. The particle motion is defined by the accelerator layout as well as interactions between the particles, and is evaluated over space and time. The simulations are very computationally intensive. Message Passing Interface (MPI) has been successfully used in parts since 2001, but still long running simulations take hours.

Graphics processing units (GPUs) allow for mass-execution of algorithms on large amounts of data and are therefore more frequently used for parallelization. The beam physics department at GSI decided thus to evaluate the use of GPUs in their existing simulations. The test system contains a 2.67 GHz Intel Xeon X5650 processor and NVIDIA's Tesla C2075 GPU that is programmed using CUDA C. The preference for NVIDIA is mainly motivated by freely available libraries such as cuFFT and cuBLAS. In the studies, the possibilities and limits of the GPU integration are investigated. Several modifications to the present algorithms are discussed and evaluated. The result are maintainable parallelized algorithms that allow for up to six-times faster simulations and will be the basis for future developments.

The remainder of this article is structured as follows: Section 2 introduces the general parallelization aspects of GPUs, Section 3 describes the simulation methods, Section 4 and 5 present the realized modifications and discuss the findings, Section 6 concludes.

2 Parallelization with GPUs

GPUs are massively parallel accelerators originally introduced for graphic processing, but nowadays also used for general purpose computing. In contrast to CPUs, that execute different programs sequentially, GPUs execute one program with hundreds of parallel execution units. To the developer, the GPU is represented through a logical abstraction layer, NVIDIA's Compute Unified Device Architecture (CUDA). The PC is referred to as *host*, the graphics card as *device*. A *kernel* describes the procedure for a single execution unit on the GPU and is executed in many *threads* that each have their own index to access the data. Threads grouped in *blocks* are executed together and have a shared memory. Blocks are structured in *grids*.

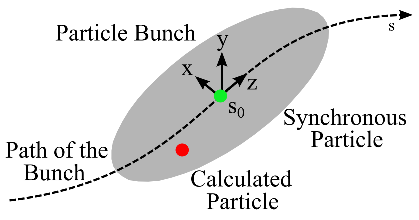
The Tesla C2075 GPU consists of 14 multiprocessors with 32 cores each [NVI09, p. 7]. At runtime, blocks get assigned to multiprocessors according to their resource usage [KH10, p. 84]. Blocks are independent to ensure scalability [Far11, p. 86]. Out of one block, warps of 32 threads get executed in parallel. Several warps are active in a time-sliced way [KH10, p. 88]. The GPU has different memory types: slow global memory for data exchange with the host system, fast shared memory for threads within a block, and very fast registers per thread. Optimal memory usage is essential for improving the speedup.

GPU algorithms can be analyzed treating each thread as a logical processor in the shared memory model [SK10, p. 66]. Threads in a warp act like a vector computer of the 1970s and fit in the SIMD category [Far11, p. 88] in the classification of Flynn [Fly72]. For the whole GPU, the author follows the view of [KH10] who suggests SPMD (single program, multiple data) known from MPI, where autonomous processors execute the same program on different data, which fits to independent blocks on the GPU. Since the exact block execution is not known here, only single warps are analyzed using the Parallel Random Access Machine (PRAM) model [JáJ92, p. 9 ff.] which is based on the well-known RAM model [AHU74, p. 5 ff.]. Further statements regarding the whole GPU can be derived, but must not resemble real measurements.

3 Simulation of the Particle Motion

The simulation model used is shortly described as basis. In accelerators, charged particles are guided, accelerated and interact with each other through electromagnetic forces. This Lorentz force can be written as $\vec{F}=q\cdot(\vec{v}\times\vec{B}+\vec{E})$, with q being the charge and $\vec{v}$ the velocity of the particle, and $\vec{B}$ and $\vec{E}$ the magnetic and electric fields surrounding the particle. The magnetic field bends and focuses the particles transversally. The electric field accelerates the particles longitudinally and bundles the beam into bunches [Wil05, pp. 3-4].

In simulation, particles or larger macro particles are described relative to the synchronous (ideal) particle s_0 (see Fig. 1) using a vector with x, x' as horizontal, y, y' as vertical positional and directional deviation, z as longitudinal positional deviation and v als momentum deviation [Wil05, p. 76 ff.]. x, y, v are measured in mm, x', y' in mrad, v in per-mil. While particles move through the accelerator, elements like magnets act on them. To track the particles in the simulation, their movement is realized as matrix-vector multiplication with magnets being represented as transport matrices.



$$\bar{p}' = M \cdot \bar{p} = \begin{pmatrix} M_{11} & M_{12} & 0 & 0 & 0 & M_{16} \\ M_{21} & M_{22} & 0 & 0 & 0 & M_{26} \\ 0 & 0 & M_{33} & M_{34} & 0 & 0 \\ 0 & 0 & M_{43} & M_{44} & 0 & 0 \\ M_{51} & M_{52} & 0 & 0 & 1 & M_{56} \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} x \\ x' \\ y \\ y' \\ z \\ v \end{pmatrix}$$

Figure 1: Simulation of the particle motion

To simulate the particle interaction, the forces between the particles are discretized on a grid and then the grid acts back on the particles. Doing so drastically reduces the complexity compared to a full n-body simulation with $\mathcal{O}(n^2)$. This technique is called Particle-in-cell (PIC) method and is used since the 1950s for plasma simulations, see [BL05, p. 3]. The calculation cycle for one discrete time step is shown in Fig. 2. It starts on the right side with a given particle distribution. Based on the particle density, in the first step the charge (ρ) and current distribution (J) are interpolated on the grid. In the second step (field solver) the electric and magnetic space charge field on the grid is calculated, which can be done e. g. with a forward and backward FFT. The electromagnetic field gets integrated and results in an electrostatic potential [Rei08, p. 173]. In the third step, forces are derived from the potential, that diffract or accelerate the particles [Rei08, p. 164]. In the last step (particle pusher) new particle coordinates are calculated based on these forces.

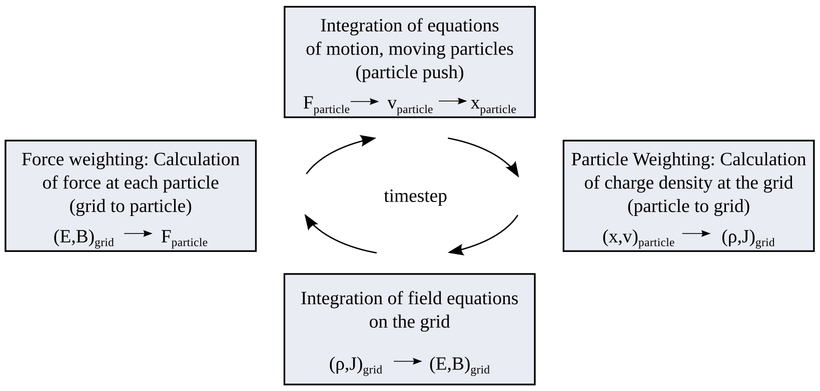

Integration of equations of motion, moving particles (particle push)
$F_{particle} \longrightarrow v_{particle} \longrightarrow x_{particle}$
Force weighting: Calculation of force at each particle (grid to particle)
$(E,B)_{grid} \longrightarrow F_{particle}$
timestep
Particle Weighting: Calculation of charge density at the grid (particle to grid)
$(x,v)_{particle} \longrightarrow (\rho,J)_{grid}$
Integration of field equations on the grid
$(\rho,J)_{grid} \longrightarrow (E,B)_{grid}$

Figure 2: PIC calculation cycle (see [BL05])

4 Particle Tracking and General GPU Aspects

The existing simulation code PATRIC (Particle TrackIng Code) [BFK06] is a C++ program developed by the beam physics department and is specific to the research emphasis at GSI. Here, a simplified version was used that focuses on particle tracking. Since the particle transport steps in PATRIC (matrix-vector multiplications) take 64% of the time, this step was ported to the GPU. Particle tracking is memory bound: the compute to global memory access (CGMA) ratio is only 3 instead of 30 for fully exploiting the GPU [KH10, p. 97]. Therefore memory usage was analyzed. For the up to a few hundred transport matrices, the faster constant memory is too small, so global memory is used with the keyword `const` to benefit from caching. In measurements, the difference to constant memory was negligible. Particle data is also kept in global memory due to its size. To have neighboring threads access neighboring data, the array of structures (AoS) was converted to a structure of arrays (SoA) [Far11, p. 6], with e. g. all 'x' coordinates in one array.

The hypothesis, whether it is beneficial to calculate a single coordinate or a whole particle per thread, was tested through measurements using a simplified linear optics with 16 transport matrices and 128 turns with $100,000$ particles. One particle per thread with its more favorable CGMA ratio was faster, resulting in a speedup of 1.18 compared to the CPU version, see Tab. 1. The versions in brackets highlight noteworthy aspects: host-device synchronization has hardly any impact, most of the GPU time is spent on calculations as expected, but – most importantly – data copy takes about 27% of the execution time.

For the number of threads per block NVIDIAs optimal block size calculator [NVI13b] suggests several possibilities. In measurements, 64 up to 256 threads showed marginal differences below 1.5%, above 256 threads the execution time slightly went up. So the number of 256 threads per block was chosen, following also a recommendation by [Far11].

The next goal was keeping particles between transport steps on the GPU to avoid data transfer, as suggested by [NVI13a]. Methods were added for data copy and for dealing with lost particles (additional `boolean` array). To investigate the behavior with varying problem size, typical number of particles up to $1,000,000$ were simulated. For the CPU version, linear scaling is expected, since the central loops over all particles are of complexity $\mathcal{O}(n)$. This is reflected well by the measurements, see Fig. 3. The complexity of the

Table 1: Single transport step on the GPU: speedup of 1.18 with one particle per thread

Version	CPU time	GPU time	Sum
CPU: Original version	19.04 s	—	19.04 s
GPU: Transport step, Thread: Coordinate	15.21 s	2.74 s	17.95 s
(Version w/o synchronization)	15.26 s	2.73 s	17.99 s
(Version w/o calculation)	15.16 s	0.43 s	15.59 s
GPU: Transport step, Thread: Particle	15.30 s	0.82 s	16.12 s
(Version w/o data copy)	10.83 s	0.99 s	11.82 s

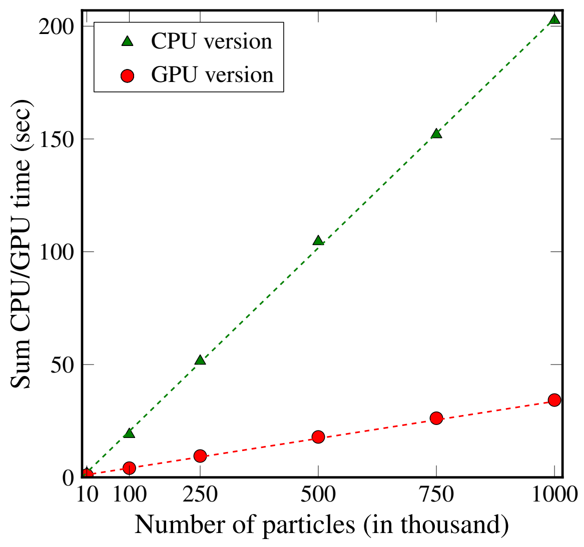


Figure 3: Particle tracking on the GPU, speedup of 6

GPU version is reduced to $\mathcal{O}(\frac{n}{p})$, but data copy imposes additional costs. A positive linear scaling can be also observed for the GPU version, which shows a six-times speedup and is faster above $8,150$ particles. Concluding, using the GPU for particle tracking leads to a good speedup, if the particles are kept on the GPU. However, a comparable measurement with a GPU with 2 multiprocessors instead of 14 was even slower than the CPU version, demonstrating that the number of parallel execution units is of course the main factor.

Several questions were addressed using the parallelized version. First, the impact of particle loss was analyzed. With a maximum thread divergence at 50% particle loss, the GPU time only increased by 1.5%, because each second thread indeed does nothing instead of calculations. Thus particle loss can be neglected and no re-sorting of particles is necessary.

Second, floating point arithmetic was examined. Double precision performance has been a weak point of GPUs in the past. Single precision proved less than 5% faster. Thus double precision can be kept, as it is necessary for long running simulations.

Lastly, the focus was put on output of intermediate results. While output every 5 turns (90 transport steps) results in $10.6s$ for the CPU version and $2.6s$ for the GPU version, output after each turn already takes $19s$ and $4s$ whereas output after each transport step leads to $445s$ and $92s$. Thus output should be limited. To reduce the time for data copy, it was tested to overlap the transport step on the GPU with data copy to the host using streams. Particle arrays were duplicated and pointer switching was used. Measurements showed only 5% performance gain. Due to the dominating memory access, not much can be overlapped and streams do not pay off here. Intermediate results contain calculated beam parameters, that are used to observe the beam quality and intensity. Instead of transferring the full particle data back to the host, those calculations can be done di-

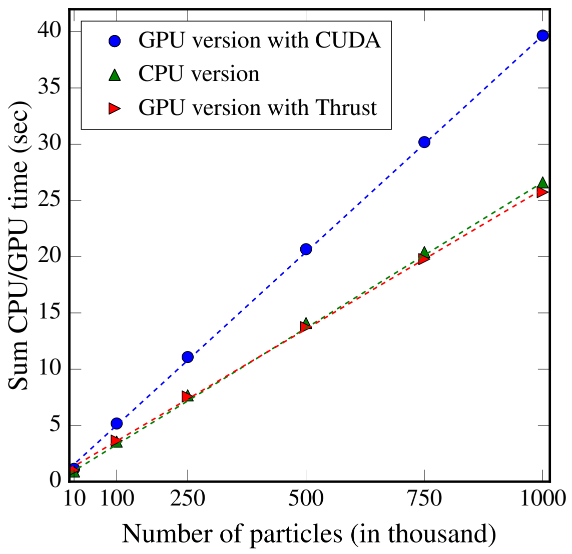


Figure 4: Beam emittance: CUDA vs. Thrust vs. CPU, Thrust slightly faster

rectly on the GPU. As example, the beam emittance was chosen. The rms-emittance (root mean square emittance, 1σ-divergence) as a measure for beam quality describes the (preferably small) geometric bundling of the beam around the optimal orbit, i. e. beam width multiplied by divergence, in mm $\times$ mrad. For the horizontal plane it is defined as: $\varepsilon_x = \sqrt{\langle x^2\rangle\langle x'^2\rangle - \langle xx'\rangle^2}$ [Rei08, p. 321]. The CPU version was compared to two GPU versions using CUDA and Thrust. For the CUDA version, data was summed block-wise using reverse binary reduction and sequential addressing with an atomic update at the end. Although the complexity per block is reduced from $\mathcal{O}(n)$ to $\mathcal{O}(\log n)$, it is slower, see Fig. 4. The version using the Thrust library [NVI13c] with `thrust::reduce` is significantly faster and comparable to the CPU version, since Thrust is highly optimzed. It allows for quick GPU integration, but as the API is limited, the author would not recommend it for realizing complex algorithms. A noteworthy result is that beam parameter calculations are possible on the GPU and in the discussed case Thrust is preferable. But since reduction operations are very expensive on the GPU with hardly any speedup, in general these calculations should be kept on the CPU despite the extra copy steps. This is also favorable with respect to many CPU methods already being in existence that do not need to be ported.

5 Collective Effects

The existing simulation code LOBO (Longitudinal Beam Dynamics Simulations Code) [BFH00] is also developed by the beam physics department and focuses on longitudinal effects which are realized as one dimensional simulation. The represented elements are the radio frequency cavities with their longitudinal forces. The program e. g. simulates the particle capture into particle bunches. The simulation of collective effects between the particles allows to study if the beam becomes unstable. More about the simulations in LOBO can be found in [ABF12]. Here, collective effects are put into focus for parallelization.

LOBO realizes the PIC algorithm described in Section 3. The particles are represented as vector with z als longitudinal positional deviation and dp als momentum deviation, the grids are one dimensional. Every step of the PIC cycle is implemented as separate method. In the original program, 85% of the time is spent on interpolating the particles on the grids, 4% on moving the particles and only 1% on calculating the space charge fields. Since for the latter efficient FFT algorithms exist both for the CPU and GPU, the focus was put on the interpolation steps.

For parallelizing the PIC algorithm using a GPU, two main approaches exist (e. g. [A$^+$12]). In the first approach, the grid information is updated from each particle, thus as many memory accesses are necessary as number of particles. Those memory accesses have to be synchronized, which is the main effort of this approach. In the second approach, the grid information is calculated per grid point using sorted particles. Here, only as many memory accesses are necessary as number of grid points. In this approach the main effort lies in sorting the particles. Both approaches were implemented and evaluated.

For the first approach, as many threads as particles were used. It is necessary to reset the grid information and perform a global synchronization before updating the grid, which was realized with a separate kernel. The concurrent memory accesses were done with atomic updates. For the CPU version, again a linear behavior is expected. For the GPU version the atomic updates of the grid data are expected to be resource-intense. As shown in Fig. 5 the

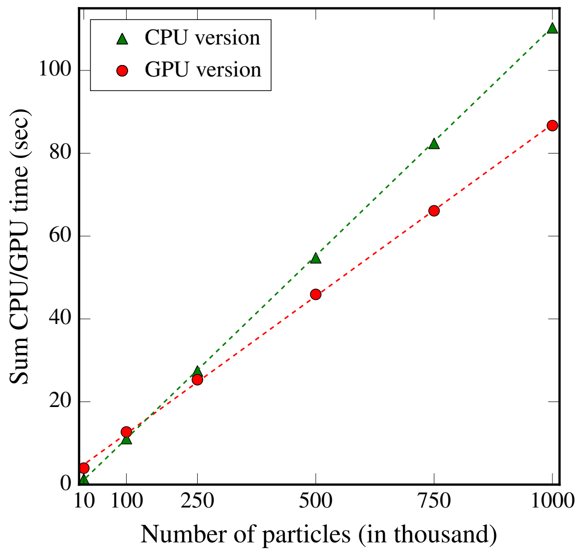


Figure 5: Collective effects: Interpolation step with atomic updates, speedup of 1.19

GPU version has a small speedup of 1.19 and is faster above $170,000$ particles. A more detailed look at the runtime performed with $250,000$ particles shows, that nearly 70% of the GPU time is spent for interpolating the particles on the grids. Compared to 12% for the particle push it is obvious, that the concurrent write access is indeed the bottleneck.

To identify, how the grid size effects the runtime, different measurements with typical grid sizes were performed. For the CPU version a linear correlation between grid sizes and runtime can be observed. The same exists for larger grid sizes on the GPU. That the runtime does not decrease any further below 256 grid points can be attributed to the relatively high number of concurrent write accesses for smaller grids.

Table 2: Collective effects: Comparison of grid sizes

Version / Grid Size	**256**	**512**	**1024**
CPU version	$13.49\ s$	$27.44\ s$	$56.74\ s$
GPU version	$22.38\ s$	$24.81\ s$	$53.16\ s$

The implementation with atomic updates offers only a small benefit compared to the CPU version and depends strongly on the particle numbers and grid sizes. The GPU version can only be recommended beyond $170,000$ particles and 512 grid points.

The second version, where particles are sorted after each particle push according to their mapping to grid points allows to treat the grid points separately. An additional array keeps the necessary mapping information for each particle. As many threads as grid points are needed. Each thread updates the left and right grid point and works with all particles in between. Additional arrays keep the information per thread about the start index, stop index and number of particles it has to interpolate. Update of grid points is done with $2 \cdot k$ atomic updates, where k is the number of grid points. Much less update operations are needed, since the number of grid points is typically 1-2 orders of magnitude smaller than the number of particles. Sorting of the particles is the major effort here and was done using the Thrust functions `thrust::sort_by_key` and `thrust::lower_bound`.

Unfortunately this implementation proved to be eight-times slower than the CPU version. This can be explained with several aspects: the number of total threads is quite low, and also no measures were taken for load-balancing between the threads. This prevented an optimal utilization of the GPU. It is no surprise that sorting is indeed very resource-intense. Much more effort would be needed to optimize the sorting before further following this approach. Additionally, measures should be taken to subdivide the particles and to use more threads for a better GPU saturation, but this implies more synchronization points.

The implemented version for the GPU with pre-sorting was too slow and would need much more adaptation to become more comparable with the CPU-based version. The version with atomic updates already showed a slight benefit compared to the CPU version. Also atomic updates are better supported with each graphics card generation, and it is expected, that they will become faster in the future. Thus it is recommended to use this version as basis for future developments.

6 Summary and Outlook

The possibilities for parallelizing the existing particle simulations PATRIC and LOBO with GPU programming have been studied. Based on a simplified version of PATRIC, a six-times speedup could be achieved for the particle tracking with one particle per thread and while keeping the particles as long as possible on the GPU. This good acceleration is possible, since each particle can be treated independently.

The effect of outputting intermediate results was measured and it was found that it should be limited to a minimum. Overlapping with streams showed no performance gain, since the ratio of calculations to memory accesses is very small. Beam parameters were calculated on the GPU for which the highly optimized Thrust library should be preferred over self-written code. But since no performance gain was observed, the beam parameter calculations can be kept on the CPU to benefit from the reuse of existing routines.

Thread divergence due to particle loss and single vs. double precision floating point performance were evaluated and resulted in only 1.5% and 5% performance loss. Therefore, no further measures have to be taken. Especially, the double precision code can be kept.

Based on a version of LOBO, possibilities for parallelizing collective effects have been investigated. The step of the PIC calculation cycle that interpolates particles on the grid is hard to parallelize with the competing write accesses on the grid as bottleneck. Two approaches followed in other research have been implemented and compared. The version with atomic updates showed a slight benefit compared to the CPU version and will be followed further. The version with pre-sorting of particles was too slow, since the possibilities of the GPU could not be exploited well.

Concluding, the usage of GPUs for particle simulations can be recommended. For well parallelizable problems, a good performance gain can be expected. Tracking was realized with a six-times speedup. For calculations over all particles or the synchronization of many threads, the use of the GPU is limited. But with even a small speedup of 1.19 for collective effects, simulations that combine both aspects can also profit from GPU integration.

While this study concentrated on the GPU, hybrid environments become popular also for PIC algorithms [Dec15]. With the findings it can be suggested for mixed MPI/GPU code that each node should at least calculate $10,000$ particles for tracking and $200,000$ for collective effects. Since the typical number of particles here is up to 1 million, the effort of developing and maintaining hybrid code has to be carefully weighted against any speedups.

In general, GPUs offer good opportunities for parallelization without the need for special parallel computers. Since programming APIs are not yet unified, adaptation effort might arise in the future. For smaller projects, the author suggests to develop not too specialized but instead more algorithm-focused, maintainable code. The tendency of bigger projects to develop specific libraries for GPU integration is no option here. However, libraries could be used, if they were publicly available. Future generic APIs that hide even hybrid parallel architectures and allow for less hardware-oriented programming would be most preferable, as suggested in [B$^+$12]. For the GPU hardware, the trend towards further integration with the CPU will remove the need for data copying. With data copy being a major bottleneck, particle simulations can benefit from that in the future.

References

[A$^+$12] K. Amyx et al., Accelerating Particle-Tracking Based Beam Dynamics Simulations with GPUs, In *Proc. of the GPU Technology Conference (GTC) 2012 - Poster*, San Jose, CA, USA, 2012. Tech-X Corporation, Argonne National Laboratory.

[ABF12] S. Appel and O. Boine-Frankenheim, Microbunch dynamics and multistream instability in a heavy-ion synchrotron, *Phys. Rev. ST Accel. Beams*, 15:054201, May 2012.

[AHU74] A. V. Aho, J. E. Hopcroft, and J. D. Ullman, *The design and analysis of computer algorithms*, Addison-Wesley, Reading, MA, USA, 1974.

[B$^+$12] R. Buchty et al., A Survey on Hardware-aware and Heterogeneous Computing on Multicore Processors and Accelerators, *Concurr. Comput. : Pract. Exper.*, 24(7):663–675, May 2012.

[BFH00] O. Boine-Frankenheim and I. Hofmann, Vlasov simulation of the microwave instability in space charge dominated coasting ion beams, *Physical Review Special Topics - Accelerators and Beams*, 3(10):104202, October 2000.

[BFK06] O. Boine-Frankenheim and V. Kornilov, Implementation and Validation of Space Charge and Impedance Kicks in the Code Patric for Studies of Transverse Coherent Instabilities in the Fair Rings, In *Proc. of the International Computational Accelerator Physics Conference (ICAP) 2006*, pages 267–270, Chamonix, France, 2006.

[BL05] C. K. Birdsall and A. B. Langdon, *Plasma Physics via Computer Simulation*, Taylor and Francis Group, New York, NY, USA, 2005.

[Dec15] V. K. Decyk, Skeleton Particle-in-Cell Codes on Emerging Computer Architectures, *Computing in Science Engineering*, 17(2):47–52, Mar 2015.

[Far11] R. Farber, *CUDA Application Design and Development*, Morgan Kaufmann, Amsterdam, Netherlands, 2011.

[Fly72] M. Flynn, Some Computer Organizations and Their Effectiveness, *Computers, IEEE Transactions on*, C-21(9):948–960, Sept 1972.

[JáJ92] J. JáJá, *An Introduction to Parallel Algorithms*, Addison-Wesley, Reading, MA, USA, 1992.

[KH10] D. B. Kirk and W. W. Hwu, *Programming Massively Parallel Processors: A Hands-on Approach (Applications of GPU Computing Series)*, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2010.

[NVI09] NVIDIA Corp., Fermi Compute Architecture Whitepaper, 2009.

[NVI13a] NVIDIA Corp., CUDA C Programming Guide, Version 5.0, 2013.

[NVI13b] NVIDIA Corp., CUDA Occupancy Calculator, 2013.

[NVI13c] NVIDIA Corp., CUDA Toolkit Documentation: Thrust, 2013.

[Rei08] M. Reiser, *Theory and design of charged particle beams*, Wiley-VCH, Weinheim, Germany, 2. edition, 2008.

[SK10] J. Sanders and E. Kandrot, *CUDA by Example: An Introduction to General-Purpose GPU Programming*, Addison Wesley, Upper Saddle River, NJ, USA, 2010.

[Wil05] K. Wille, *The Physics of Particle Accelerators*, Oxford University Press, New York, NY, USA, 2005, Reprint of the 2000 edition.

Real-Time Vision System for License Plate Detection and Recognition on FPGA

Faird Rosli , Ahmed Elhossini, Ben Juurlink

Embedded Systems Architectures (AES)
Technical University of Berlin
Einsteinufer 17
D-10587, Berlin, Germany
{mohd.f.mohdrosli1,ahmed.elhossini,b.juurlink}@tu-berlin.de

Abstract: Rapid development of the Field Programmable Gate Array (FPGA) offers an alternative way to provide acceleration for computationally intensive tasks such as digital signal and image processing. Its ability to perform parallel processing shows the potential in implementing a high speed vision system. Out of numerous applications of computer vision, this paper focuses on the hardware implementation of one that is commercially known as Automatic Number Plate Recognition (ANPR).Morphological operations and Optical Character Recognition (OCR) algorithms have been implemented on a Xilinx Zynq-7000 All-Programmable SoC to realize the functions of an ANPR system. Test results have shown that the designed and implemented processing pipeline that consumed 63 % of the logic resources is capable of delivering the results with relatively low error rate. Most importantly, the computation time satisfies the real-time requirement for many ANPR applications.

1 Introduction

In recent years, the significant evolution of computer vision can be seen as it is making its way into an increasing number of application domains. The research and development in the field of computer or machine vision has defined methods for processing and analyzing images from real world to provide human capabilities of understanding images to machines and robots. There is a strong and growing demand for computer vision systems in the automotive domain. Intelligent cars that are available in the market nowadays are equipped with various camera-based driver assistance systems such as lane detection, night view assist, pedestrian detection and traffic sign recognition [Gr13].

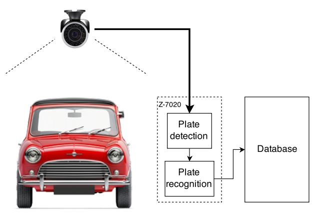


Figure 1: System Configuration for ANPR on FPGA.

These applications are required to work reliably in a large range of lighting and climatic conditions and to process a very high frame rate video signal in real-time. Besides those above-mentioned applications, the Automatic Number Plate Recognition (ANPR) is also one of the computer vision applications that is widely used in the automotive domain. However, it is better known as a surveillance technology rather than a driver assistance system. Vehicles are usually identified by their registration number which are easily readable by humans. But for machines, a plate number is a grey image defined as a mathematical function that represents light intensity at a certain point within an image [Ma07].

The main objective of this paper is to develop a framework of embedded components for custom computer vision applications that run on a reconfigurable hardware and operate in real-time. The proposed framework was customized for an application in the automotive domain. The ANPR application is implemented on FPGA as illustrated in Figure 1 which is suitable for a wide range of applications. The application is constructed using basic computer vision and image processing algorithms. Various hardware cores are developed for each algorithm that later can simply be reused by other vision applications.

The paper is organized as follows: Section 2 gives overview of the related work to this paper. Section 3 describes the proposed methods for license plate detection and recognition. Section 4 reveals the architecture of each processing block. Section 5 presents the testing procedure and analysis of the implemented license plate detection and recognition system. Finally, Section 6 summarizes the whole work that is done so far and concludes the work.

2 Related Work

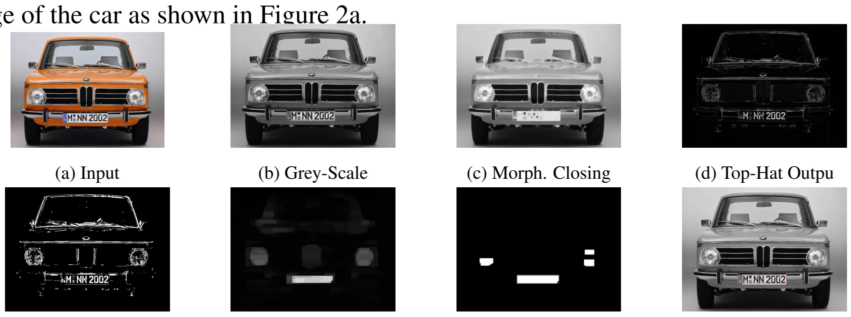
In this paper we investigate employing FPGA to implement a framework for computer vision and specifically for automatic license plate recognition. The most common approaches applied to detect the license plate within an image are the combination of edge detection and binary morphology as explained in [SGD06]. The detection rate with this method is highly affected by the quality of the image. They are based on the assumption that car plates have strong edges that will survive strong filtering. Unfortunately, this is not very effective for urban environment. License plate detection and segmentation algorithm with histogram projection as proposed in [As13] and [Hs09] is found to be challenging especially when the image contains a lot of details in the background. Defining the threshold value require a few steps of mathematical calculation and the process of finding the peak may introduce some delay in the processing. A simpler approach that produces satisfactory result is by using greyscale morphology that is already used in [Iw] and [Od]. Optical character recognition (OCR), that is normally used to translate images of handwritten text into machine encoded text, is applicable for the recognition of license plate characters. The work in [ZDF10] divides a handwritten character into several rectangular zones to obtain a 13-element feature vector. Each element represents the number of foreground pixel in each defined zone. In [SDR12] a similar method is used to divide a 32 x 32 pixel character image into 16 zones. The authors have proposed eight directional distribution features which are calculated for each zone. Three different classification methods, which are artificial neural networks (ANN), support vector machine (SVM) and k-nearest-neighbour (KNN) are used to recognise the characters.

Some of the methods and algorithms described above can be combined and implemented on embedded hardware platform. For example, license plate detection and recognition on an embedded DSP platform was introduced in [ALB07]. The total processing time, beginning from image acquisition until character classification requires 41.35 ms. Another implementation of plate localisation on Xilinx Virtex-4 was described in [ZBR11] and is capable of processing one image in 3.8 ms and it consumes less than 30% of the on-chip resources. No recognition stage was presented in that work. A complete ANPR system is implemented on FPGA in [Je13]. The system utilizes 80 % of the Xilinx Virtex-4 LX40 resources and is capable of processing a standard definition image (640 x 480) in less than 10 ms. Algorithms for license plate detection are also developed in [TKA07] using Handel-C and then translated into Verilog HDL with Celoxca's DK4 to implement on Virtex II Pro. Performance comparison between the software and the hardware implementation is stated at the end of the paper. The resource utilization for their hardware implementation is about 2 to 3 times more than the work in [ZBR11] and yet requires longer processing time for an input image with a smaller resolution. However,their results show that the same algorithms execute 4 times faster with the hardware than the software.

In this paper we adopted some of the listed methods and algorithms to design and implement a pipelined processing system for computer vision on a reconfigurable hardware. The proposed work presents hardware modules for various morphology based filters, along with connected component analysis and k-means classifier. These components are arranged in a single pipeline for ANPR. These components are designed to be easily used in various computer vision applications.

3 Proposed Algorithm

In this section we propose an algorithm for ANPR. The input to the algorithms is a frontal image of the car as shown in Figure 2a.



(a) Input (b) Grey-Scale (c) Morph. Closing (d) Top-Hat Outpu

(e) Binary Output (f) Morph. Cleaning (g) Cleaned Output (h) Bounding Box

Figure 2: Output of Various Processing Stages (Actual Output of the Hardware System)

3.1 License Plate Detection

3.1.1 Top-Hat Filtering

The input image is converted to grey-scale (Figure 2b). Several stages of mathematical morphology are chosen to locate the license plate of the vehicle in the image. Firstly, the closing operation with a structuring element of 7 x 7 pixel is applied to erase the characters of the license plate (Figure 2c). When image subtraction is performed between the resulting image and the initial grey-scale image, an image as shown in Figure 2d is obtained. This operation is known as black Top-Hat morphology. It returns an image containing objects or elements that are smaller than the structuring element and darker than their surroundings. Since European license plates mostly have white background and the characters are black in colour, they will remain as foreground objects in the output image. A binary image (Figure 2e) is obtained from the Top-Hat image by using thresholding.

3.1.2 Background Cleaning and Plate Segmentation

Grey-scale morphology is applied to the top-hat image to find the region that contains the vehicle license plate. The license plate location can be detected roughly with a closing

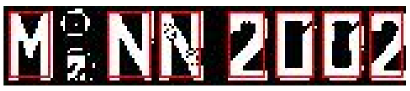


Figure 3: Bounding Box for Each Character.

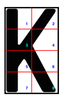


Figure 4: A letter "K" is divided into 8 zones.

operation. The structuring element for this operation has a size of 1 x 45 pixel. The length is chosen such that it has at least twice the length of a license plate character. After that, unwanted elements that does not belong to the license plate area are removed by using morphological opening with a rectangular structuring element 15 x 25 pixel (Figure 2f). The resulting image contains several light areas, and the area of the license plate appears to be lighter than others. Thresholding is applied to produce a binary image that maintains the license plate area and suppresses the darker regions (Figure 2g). Finally, binary morphology dilation with a structuring element of 5 x 5 pixel is applied to enhance the edges of the binary image. Plate segmentation is performed by applying the connected component labelling algorithm to distinguish these regions. A bounding box algorithm is applied to find the rectangular boundary that encloses each region.

3.2 License Plate Recognition

3.2.1 Character Segmentation

The process of finding characters on a license plate is actually the same as finding the license plate in the input image. Connected component labelling is applied to give a label for each character and a bounding box that encloses each character is defined so that it can be segmented from the image and sent to the recognition (classifier) unit. The bounding box that encloses a detected object must have a minimum width and height to distinguish between characters and noise. The result of character segmentation is shown in Figure 3.

3.2.2 Feature extraction

An image of a license plate character is partitioned into 8 zones (Figure 4) and the number of foreground pixels in each zone (pixel density) is calculated. Thus, an image can have a feature vector of at least 8 elements. Additionally, several types of edges of each extracted character can also be determined to produce a feature vector of the character. There are 14 different types of edges (Figure 5) and the number of occurrences of each type is calculated for each zone. Combined with the pixel density of each zone a total of 72 elements in the features vector of each character is calculated.

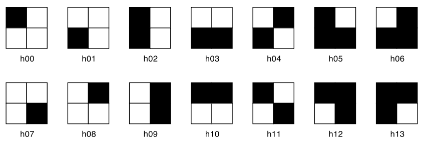


Figure 5: Edge Types in Feature Extraction

3.2.3 Character Classification

The detected characters are classified using a simplified k-means clustering algorithm. For each alphabetical character, at least 5 sample images are taken and a feature vector is extracted from each image. A mean vector for a character is determined by using the extracted feature vectors. Each mean vector is stored in a database which is used for the classification task. There are 36 types or classes of characters that the designed vision system must be able to recognize (26 alphabetical characters + 10 numerical digits).

4 Implementation

In this section we present the details of the proposed architecture. Each component is modelled using VHDL and simulated using Xilinx ISE development tools from Xilinx. The target device is Xilinx Artix-7 architecture. The design is assumed to stream the image data pixel by pixel in every clock cycle which makes it suitable for streaming data from various image sources such as cameras.

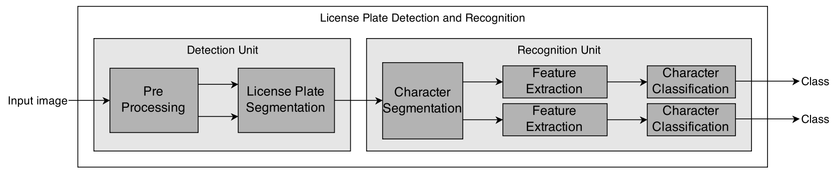


Figure 6: The Architecture of the Processing Pipeline

4.1 Complete Processing Pipeline

The complete processing pipeline shown in Figure 6 is composed of two main parts. The first part is the detection unit, which is composed of the pre-processing and the license plate segmentation units. It receives a stream of input images in RGB format and produces a binary image of a license plate. The second part is the recognition unit that receives the binary image and performs character segmentation in its first processing stage. Two license plate character images are segmented simultaneously. Therefore parallel execution of character classification is possible.

4.2 License Plate Detection Unit

In this stage the position of the license plate is detected using a series of morphological operations and connected component labelling as explained earlier. Two main units in this stage: the pre-processing unit, and segmentation unit.

4.2.1 Image Pre-processing Unit

During the pre-processing stage (Figure 7), the grey converter unit prepares a grey-scale image to the black Top-Hat unit by converting the colour space of the RGB image. The background cleaning and thresholding of the resulting image of black Top-Hat morphology are executed in parallel. This processing unit will produce two different output images similar to that are shown in Figure 2e and Figure 2g.

Morphological Filter: The architecture for grey-scale morphology is based on the design in [Ba11]. For a rectangular structuring element, efficient separable implementations are applicable as shown in Figure 8. The implementation of this architecture allows users to select between various morphological operation and structuring elements sizes using a configuration word. Additionally, the length of the row buffer, which is implemented using block RAM can also be adjusted via the configuration word. To implement opening and closing, the proposed architecture must be duplicated and ordered accordingly.

Top-Hat Filtering: The Top-Hat filter is performed by subtracting the grey-scale image and morphological image . The architecture of a separable morphological filter with structuring element 7 x 7, explained in the previous section, is instantiated twice here. One is used as dilation and the other as erosion. The architecture for the FPGA implementation of the Top-Hat transform is shown in Figure 9.

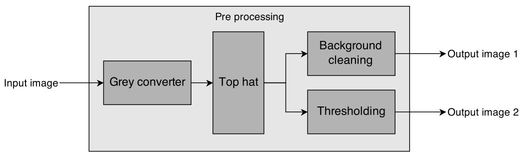


Figure 7: Image Pre-processing Unit

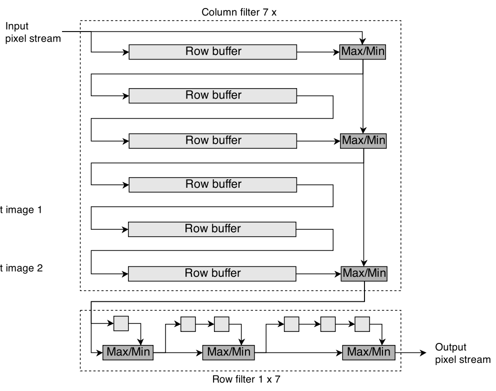


Figure 8: 7x7 Configuration of the Morphological Filter Unit

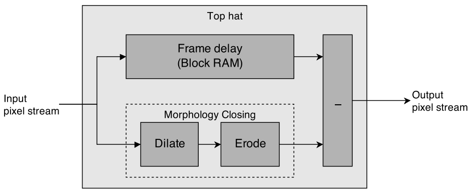


Figure 9: Top-Hat Morphological Filter.

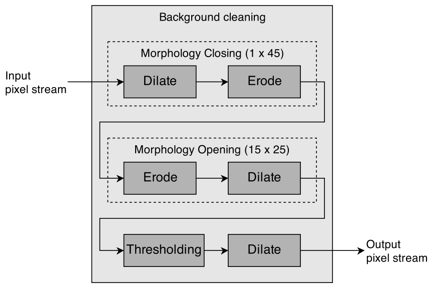


Figure 10: Background Cleaning Unit

Background Cleaning: The background cleaning consists of two morphological operations, which are opening and closing. The morphological closing is done with a horizontal filter, which has a structuring element of 1 x 45, whereas the opening is done with a normal rectangular structuring element of 15 x 25. Additionally, it also contains a global thresholding unit and a binary morphology, which is used to dilate the resulting binary image from the previous processing stages. The processing pipeline for the background cleaning is shown in Figure 10. The architecture shown in Figure 8 is employed in all stages.

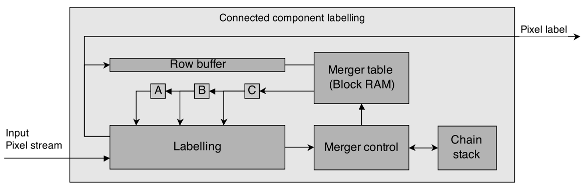


Figure 11: Connected Component Labelling Unit

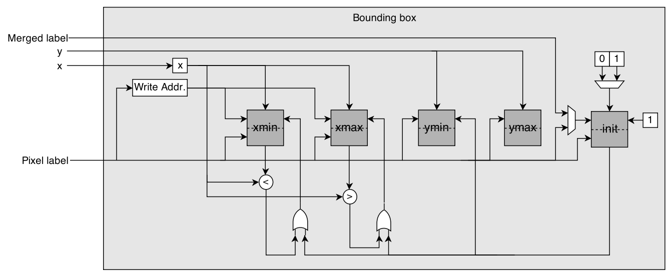


Figure 12: The Bounding Box Module

4.2.2 License Plate Segmentation Unit

Connected Component Labelling: Connected component labelling is a commonly used algorithm segment images based on the neighbourhood of pixels. Connected component labelling gives a label to every group of pixels that are neighbour to each other. The algorithm requires at least two passes to process pixels. A modification for connected component labelling is introduced in [MBJ08] that provides a single pass algorithm that eliminates the need of frame buffering and significantly reducing the latency. A simplified block diagram of the developed unit based on the single pass connected component labelling is shown in Figure 11. The neighbourhood context within a window of the size 2x2 provides the labels of the adjacent pixels to the current pixel. Unlike the architectures proposed in [MBJ08] and [JB08], the neighbourhood pixel labels are stored in registers A, B and C. These are shifted with every clock cycle as the window is scanned across the image. The merger control block updates the merger table when two objects are merged and handles new labels.

Bounding Box: Figure 12 shows the implementation of the bounding box processor for a binary image that contains multiple labels. Compared to other segmentation algorithms such as watershed and Hough transform, it provides low processing and computation cost. When the first pixel of an object is detected, the coordinates of that pixel is loaded into x_{min}, x_{max}, y_{min} and y_{max}. The y-coordinate of the following object pixel is stored into the y_{max}. The current x-coordinate is compared with x_{min} and x_{max}. At the end of the frame, the four registers indicate the extent of the object pixels within the image [Ba11].

4.3 License Plate Recognition Unit

Characters in the license plate are segmented using the same method used in plate segmentation. The image of each character is processed by several modules to extract features that will be used for character classification.

4.3.1 Feature Extraction Unit

Zoning Unit: The character sub-image is divided into eight zones as shown in Figure 4. Zoning of a character image is possible when its width and length are known. These parameters are calculated during the segmentation of a character using bounding box. The borders that define the zones can be calculated by dividing the width and height by 2 and 4 respectively.

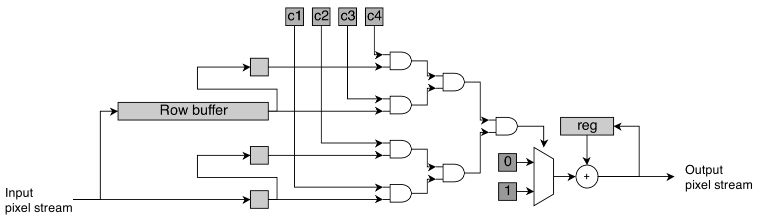


Figure 13: Edge Matching Unit

Edge Matching Unit: As explained earlier, there are 14 types of edges that an image can have. Each type is detected using the circuit shown in Figure 13. The coefficient of each field is initialized in registers c1 to c4 via a configuration word. The row buffer is used to store the previous row of the image. The result of the comparison will switch the multiplexer that selects the operand for the adder. If an edge match occurs, the register that stores the number of detected edges will increment. It will be reset when a new character image is received.

Feature Extraction Unit: As explained in section 4.3.1, a character image is divided into eight zones. Therefore, feature extraction is done for each zone instead of the whole image. The hardware architecture for feature extraction of a character image is implemented according to Figure 14. The zone selector acts like a demultiplexer that receives a zone number from the zoning unit and enables a corresponding zone. Each zone will produce a feature vector that contains 9 elements. Therefore, the architecture will produce a total number of 72 feature elements for a character.

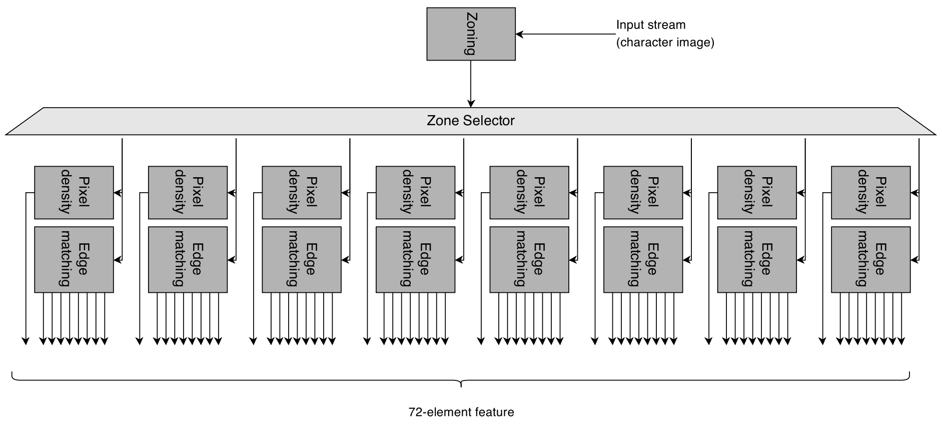


Figure 14: Feature Extraction Module.

4.3.2 Classification Unit

The classification algorithm requires a database that stores the mean vector for each license plate character. Entries of this database are stored in array of registers. The classification unit includes 36 arrays for each class. The absolute error between a feature vector and each mean vector is determined by doing element-wise subtraction of the feature vectors. The sum of absolute error is calculated by simply adding up all the elements in the error vector. Pairwise classification algorithm is used in this case, where two errors are compared at the same time. The pairwise classification unit is implemented as shown in Figure 15.

5 Results

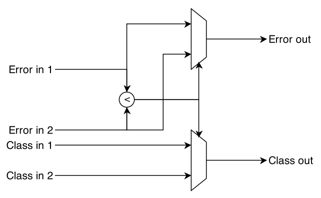


Figure 15: Pairwise Classification Unit

The implemented processing pipeline is tested with several images containing frontal view of cars. A test-bench that is created that reads an image and send it to the processing pipeline. The test-bench simulates a video stream, with standard video frame rate that can be adjusted in the test-bench. The pipeline will process the video stream and produce the output image of each processing stage. These images will be written as a bitmap file by the test bench for debugging purposes. The license plate characters are given out at the other end of the pipeline. Xilinx ISim simulator 14.5 was used to perform all types of simulations, both behavioural and timing as well as synthesising the design.

	Execution Time (ms)			Accuracy		
Processing stage	Proposed Sys.	FPGA Sys.[Je13]	DSP Sys. [Je13]	Proposed Sys.	FPGA Sys.[Je13]	DSP Sys. [Je13]
Platform- Resolution	Zynq 7020	Virtex 4 LX 40	ARM-DSP-Soc	640 x 480	640 x 480	1920 x 1080
Pre-processing	6.012	4.7				
License plate detection	0.12	0.11		90%	97%	97%
Character segmentation	0.0203	1.4				
Character recognition	0.0204	0.7		83.3%	97%	97%
Total	6.1727	6.91	71.35			

Table 1: Performance and accuracy results

5.1 Detection, Recognition, and Performance Results

If the FPGA is set to operate at maximum clock frequency to process an image with VGA resolution, license plate detection can be accomplished in less than 10 ms. This should satisfy real-time processing requirement of any ANPR application, especially for the detection of fast moving cars on a highway. The processing pipeline manages to achieve a plate detection rate of 90 % and recognition rate of 83 % as shown in Table 1. The pre-processing stages (morphology operations plus the connected component analysis) consumes 97% of the total time required by the system to process a single frame. The proposed architecture outperform the FPGA architecture presented in [Je13] in terms of processing time per-frame (Table 1). In addition, the architecture proposed in this paper is fully pipelined. This means that we can achieve higher throughput for a video stream. The DSP implementation presented in [Je13] is operating on Full-HD resolution which will require more time to process (71.35ms). However, the proposed architecture can be simply modified for Full-HD resolution, with minimal effect on the performance. Another advantage of the presented work is that it does not employ any off-chip resource which is not the case in [Je13]. The work presented in [Je13] presented a higher accuracy. This is due to the fact that ANN is employed for the classification stage. The simplified K-means algorithm employed in this paper requires larger training database which can be used to increase the accuracy of the system.

Device utilization summary			(xc7z020-1-clg484)	
Number of Slice Registers:	8456	out of	106400	7%
Number of Slice LUTs:	33975	out of	53200	63%
Number used as Memory:	88	out of	17400	0%
Specific Feature Utilization				
Number of Block RAM/FIFO:	127	out of	140	90%
Number of BUFG/BUFGCTRL/BUFHCEs	4	out of	104	3%
Number of DSP48E1s:	3	out of	220	1%

Table 2: Device utilization summary of the processing pipeline.

5.2 Resource Utilization and Implementation Results

A detailed logic utilization is listed in Table 2. The processing pipeline has a maximum operating frequency of 57.823 MHz. Multipliers are only used for translating pixel coordinate memory addresses. Block RAMs are mostly used as row and frame buffers for morphological filtering, license plate and character image segmentation. Figure 16 summarizes the resource allocation to every stage of the processing pipeline. The classifier takes almost half of the on-chip resources due to the large adder trees used to calculate the sum of absolute errors. Other components that mainly use block RAM for their task such as plate detection and character segmentation requires only 2 %. Compared to the work presented in [Je13], 80% of the Virtex-4LX 4M gate FPGA were consumed to build complete ANPR system. The Zynq 7020 chip employed in this paper has smaller size and only 63% of the resources were consumed which allows more customization of the pipeline.

6 Conclusions

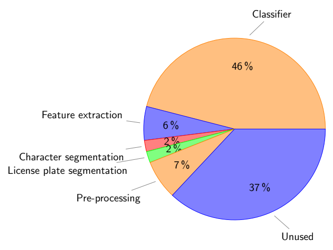


Figure 16: Resource Allocation for Each processing Stage.

In this paper, a processing pipeline for the license plate detection and recognition system has been designed and implemented on an FPGA. It consists of two main parts which are assigned for license plate localization and license plate character recognition respectively. Most of the on-chip resources are allocated to the license plate recognition part to make it capable of processing multiple characters in parallel. The pipeline has been designed to be highly reconfigurable so that it can be ported to another FPGA device that offers more logic resource. According on the test results, the morphological approach is proven to be very effective for the license plate detection task with accuracy up to 90%. The classification with k-means clustering also proves to be reliable with accuracy up to 83%. Parallel execution of the recognition unit reduces the computation time which allows the proposed architecture to process a single frame in approximately 10 ms. The pipelined operation of the system can be used to hide this latency and increase the frame rate. The processing pipeline can be customized to improve the flexibility of the vision system and to support other applications.

References

[ALB07] Arth, Clemens; Limberger, F.; Bischof, H.: Real-Time License Plate Recognition on an Embedded DSP-Platform. In: Computer Vision and Pattern Recognition, 2007. CVPR '07. IEEE Conference on. pp. 1–8, June 2007.

[As13] Ashourian, M.; Daneshmandpoura, N.; Tehrania, O. Sharifi; Moallem, P.: Real Time Implementation of a License Plate Location Recognition System Based on Adaptive Morphology. International Journal of Engineering, 2013.

[Ba11] Bailey, Donald G.: Design for Embedded Image Processing on FPGAs. ”John Wiley and Sons (Asia) Pte Ltd”, 2011.

[Gr13] Grimm, Michael: Camera-based driver assistance systems. Advanced Optical Technologies, 2013.

[Hs09] Hsieh, Ching-Tang; Chang, Liang-Chun; Hung, Kuo-Ming; Huang., Hsieh-Chang: A real-time mobile vehicle license plate detection and recognition for vehicle monitoring and management. In: Pervasive Computing (JCPC), 2009 Joint Conferences on. pp. 197–202, Dec 2009.

[Iw] Iwanowski, Marcin: Automatic car number plate detection using morphological image processing. PhD thesis, Warsaw University of Technology, Institute of Control and Industrial Electronics EC Joint Research Centre, Institute of Environment and Sustainability.

[JB08] Johnston, Christopher T.; Baily, Donald G.: FPGA implementation of a Single Pass Connected Components Algorithm. In: 4th IEEE International Symposium on Electronic Design, Test and Applications. 2008.

[Je13] Jeffrey, Zoe; Zhai, Xiaojun; Bensaali, Faycal; Sotudeh, Reza; Ariyaeeinia, Aladdin: Automatic Number Plate Recognition System on an ARM-DSP and FPGA Heterogeneous SoC Platforms. In: Poster session presented at Hot Chips: A Symposium on High Performance Chips,HC25 ,Stanford, Palo Alto, United States. 2013.

[Ma07] Martinsky, Ondrej: Algorithmic and Mathematical Principles of Automatic Number Plate Recognition Systems. Master’s thesis, Brno University of Technology, 2007.

[MBJ08] Ma, Ni; Bailey, D.G.; Johnston, C.T.: Optimised single pass connected components analysis. In: ICECE Technology, 2008. FPT 2008. International Conference on. pp. 185–192, Dec 2008.

[Od] Odone, Francesca: Experiments on a License Plate Recognition System. PhD thesis, DISI, Università degli Studi di Genova.

[SDR12] Siddharth, Kartar Singh; Dhir, Renu; Rani, Rajneesh: Comparative Recognition of Handwritten Gurmukhi Numerals Using Different Feature Sets and Classifiers. International Conference on Recent Advances and Future Trends in Information Technology (iRAFIT2012), 2012.

[SGD06] Shapiro, Vladimir; Gluhchev, Georgi; Dimov, Dimo: Towards a Multinational Car License Plate Recognition System. Machine Vision and Applications, 17(3):173–183, 2006.

[TKA07] T. Kanamori, H. Amano, M. Arai; Ajioka., Y.: A High Speed License Plate Recognition System on an FPGA. In: Field Programmable Logic and Applications, 2007. FPL 2007. International Conference on. pp. 554–557, Aug 2007.

[ZBR11] Zhai, X.; Bensaali, F.; Ramalingam, S.: Real-time license plate localisation on FPGA. In: Computer Vision and Pattern Recognition Workshops (CVPRW), 2011 IEEE Computer Society Conference on. pp. 14–19, June 2011.

[ZDF10] Zhong, Chongliang; Ding, Yalin; Fu, Jinbao: Handwritten Character Recognition Based on 13-point Feature of Skeleton and Self-Organizing Competition Network. In: Intelligent Computation Technology and Automation (ICICTA), 2010 International Conference on. volume 2, pp. 414–417, May 2010.

Extended Pattern-based Parallelization Approach for Hard Real-Time Systems and its Tool Support

Alexander Stegmeier, Martin Frieb, Theo Ungerer

Systems and Networking
Department of Computer Science
University of Augsburg
86135 Augsburg, Germany
{alexander.stegmeier,martin.frieb,ungerer}@informatik.uni-augsburg.de

Abstract: The transformation of sequential legacy code to parallel applications is hard, especially when timing requirements have to be met. There exists a systematic parallelization approach dealing with this topic. Based on practical experience, we extend it and present our modifications. Our extensions comprise an additional phase dealing with implementation details and another one for quality assurance. Its results may be used to further improve the parallel program. Moreover, we propose tool support which further facilitates the parallelization process.

1 Introduction

It is hard to parallelize a sequential legacy program [MBC11], especially when it has to meet hard real-time requirements. These programs have to finish their execution within a specific time interval (deadline). This is assured by estimating the worst case execution time (WCET)[1] [WEE+08].

The parallelization approach by Jahr et al. [JGU13b, JGU13a] was developed for the parallelization of legacy sequential hard real-time applications. It was applied on several industrial applications in the parMERASA project [UBG+13, UBG+15]. Following this experience, we extend the approach. Our modifications are an additional phase for implementation and another one for checking functional and timing constraints. Since Jahr et al. do not describe any supporting tools and it seems that everything has to be done manually, we give a brief overview on existing and planned tools we propose for assisting the parallelization.

The structure of this paper is as follows: the next section presents related work and the original approach. In section 3, we describe our extensions. Supporting tools are characterized in section 4. Finally, the paper is concluded and an outlook is given in section 5.

[1]The execution time which might occur when running the longest possible path in the program. For applications without real-time requirements, the execution time is also an applicable measure.

2 Related Work and the Original Approach

Well-known parallelization approaches in high-performance computing are the *parallel pattern language* by Mattson et al. [MSM04, MMS99] and the *PCAM approach* by Foster [Fos95]. Inspired by those, Jahr et al. introduced a pattern-supported parallelization approach [JGU13b], which we focus on. It is platform independent and applicable for hard real-time systems [JGU13a]. In the remainder, we will refer to it as *Jahr-approach*.

This approach mainly consists of two phases which are called Reveal Parallelism and Optimize Parallelism. In the first phase (**Reveal Parallelism**), the transition from the existing source code to a model takes place, which is an extended UML2 activity diagram called activity and pattern diagram (APD). The phase is characterized by extracting segments for parallel execution and creating an APD with a high degree of parallelism. Thereby, the usage of the operators fork and join is not allowed anymore. Instead, an additional type of activity node is provided which represents parallel design patterns (PDPs)[2]. These PDPs are the only way to introduce parallelism to the program. They are taken from a pattern catalogue. In the parMERASA project, a pattern catalogue containing only timing-analysable PDPs was assembled [GJU13]. Thereby, if the sequential program is timing-analysable and only these timing-analysable PDPs are utilized to introduce parallelism to the program, the resulting parallel program will also be timing-analysable[3] [JGU+14]. As a further basis for the second phase, the WCETs of the individual segments have to be determined and the dependencies between them are collected.

The goal of the second phase (**Optimize Parallelism**) is to optimize the APD for the target platform and implement the changes in the source code. At the optimization, several optimized APDs form a Pareto front in terms of minimizing their overall WCET and number of threads as well as the number of shared variables to be synchronized. Thereby, the overall WCET estimation is composed of WCET estimations of the code segments extracted in the first phase. These segments represent sequential sections in the input APD. The optimized APDs are generated by re-arranging the segments. Synchronization between the segments may only be estimated. One of the APDs from the Pareto front has to be chosen and is basis for implementing the parallel code. An example of a generated Pareto front can be seen in Figure 1. It shows the relation between several APDs in terms of their overall WCET and the number of applied threads. While the Jahr-approach does not go into detail at the implementation, we extended the approach with a third phase dealing with the implementation, see details in section 3.

In the parMERASA project [UBG+13], the Jahr-approach was applied successfully in automotive, avionics [PQnZ+14] and construction machinery [JFG+14, JFGU14] domains. Four legacy hard real-time applications were considered: motor injection, collision avoidance, stereo navigation and the control program of a foundation crane. All of these applications were written in C. Parallelization results including WCET speedups are provided in [UBG+15].

An alternative to our parallelization approach could be automatic parallelization. It is

[2]PDPs are a textual description of situations where parallelism could be applied.

[3]The targeted multi-core platform and its corresponding system software has to be timing-analysable, too.

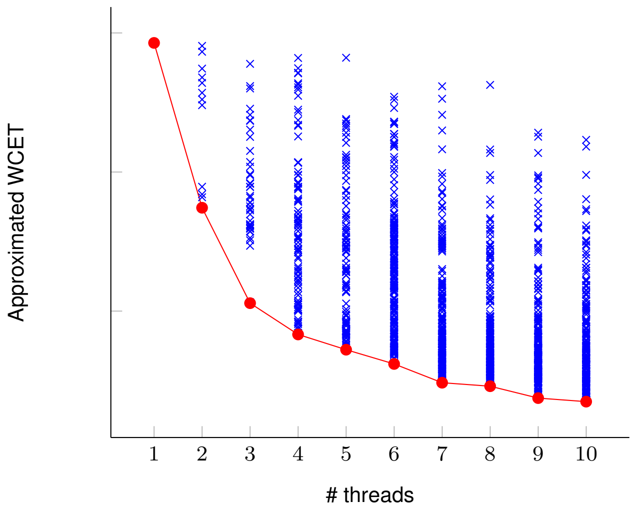


Figure 1: Optimization for minimal approximated WCET and minimal number of threads. Each × represents an APD and ○ are optimal APDs belonging to the Pareto front.

researched e.g. by Cordes et al. [CMM10] or Kempf et al. [KVP11]. However, these approaches are limited to specific situations the tools are able to detect automatically. Situations fitting only roughly will not be taken into account. Therefore, chances for parallel execution may be missed. However, these tools may be utilized as assistance when trying to find situations for parallelism in the first phase.

3 Extended Parallelization Approach

Following the practical experience made in the parMERASA project, we extended the Jahr-approach. Thereby, our goal is to remain platform independent and our extended approach may also be applied on industrial embedded real-time applications.

Figure 2 illustrates how the extended parallelization approach works: **Phase I** (Revealing Parallelism) proceeds like described in the Jahr-approach, which exposes PDPs to facilitate timing analysis. Therefore, the code has to be analysed according to its data dependencies and control flow. Based on these analyses, parallel segments of code can be determined and PDPs applied to build an APD. Furthermore, the parallel segments in code have to be annotated to get a connection between code and model level.

In the Jahr-approach, the optimization as well as the implementation take place in phase II (Optimizing Parallelism). For a clear separation of concerns, we split this phase into optimization (phase II) and implementation (phase III) to focus on their specific issues. Now, **phase II** represents the model level, taking the APD of phase I, the WCETs of the segments and the list of dependencies as input and computing a set of optimized APDs.

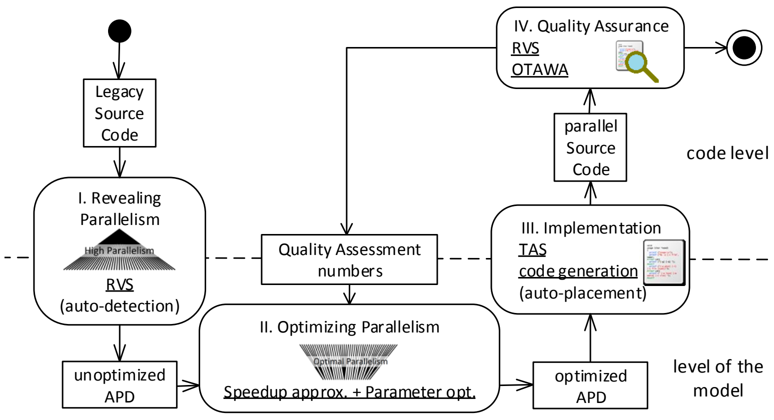


Figure 2: The four phases of the extended parallelization approach are I. Revealing Parallelism, II. Optimizing Parallelism, III. Implementation and IV. Quality Assurance. There is also a refinement loop with a feedback path from Quality Assurance to Optimizing Parallelism which is utilized to perform further optimization. Existing tools are underlined and ideas for future tools appear in parentheses.

Thereby, several variations of APDs are built. The basis for each of them is the input APD, which obtains a high degree of parallelism. The generation of the models takes place by turning on and off some of the PDPs of the basis APD. Afterwards, the generated APDs are evaluated using the estimated WCETs. The results are compared to find an optimal trade-off between minimizing the overall WCET, the synchronization effort and the applied number of threads. Finally, one APD variation is chosen from the resulting Pareto front for the implementation in phase III.

We share the opinion that the implementation is not trivial and should be considered as extra phase requiring own tool-support. In concrete, this new **phase III** includes the integration of synchronization and parallel execution in code. It takes the optimized APD of phase II and modifies the sequential legacy code according to this model. Therefore, the location of each PDP in the APD must be identified in the sequential code utilizing the annotations of phase I. The corresponding code is replaced by an implementation for parallel execution of these segments. Additionally, all synchronization points have to be identified and implemented in code. The result is source code executable on a parallel platform. Phase III leads to a clear separation of the model and code level.

In the original Jahr-approach, there is neither a check whether the deadlines hold in the parallel program, nor one for the functional correctness. Thus, we see the need of an additional **phase IV** called Quality Assurance considering these aspects and to adapt the parallel program if neccessary. The validation of functional correctness takes place by comparing the results and a defined set of variables of the sequential and parallel execution. Thereby, the code coverage is observed to ensure sufficient testing. If functional failures occur, one has to check phase I and phase III looking for possible mistakes[4]. The evaluation

[4]In phase II, it is only decided which program parts (defined by PDPs) will be executed in parallel. Therefore, no functional failures may occur.

of the parallel code's timing behavior is done by an WCET analysis, which concretely considers the overhead caused by the interaction of the parallel segments. The new results can be compared to the timing requirements.

In addition to the four phases there is a refinement loop to enable the optimization of the parallel code based on the results of phase IV. It enables the step back to model level (phase II) if the numbers do not meet the expectations (e.g. the WCET is too high). This loop involves the phases Optimize Parallelism, Implementation and Quality Assurance and utilizes the quality assurance numbers (mainly the calculated WCET) to gain a more realistic estimation of the overall WCETs in phase II. Therefore, the WCET and synchronization estimations of phase II may be updated applying the results of phase IV. This leads to a closer estimation of the WCET and possibly some new variations of APDs are suitable for parallelization. Thus, a new Pareto front can be calculated. Afterwards, an APD with more realistic estimations can be chosen and implemented.

4 Tool Support

Jahr et al. neither describe any tools assisting the parallelization process, nor how the steps may be done efficiently. Therefore, we give a brief overview on some tools that support our extended parallelization approach in this section. The original Jahr-approach as well as our approach are model-based and therefore platform and language independent. However, in this section, we focus on tools for the programming language C. Other tools may be utilized for other programming languages, but also for C. Most of the following proposals were developed or extended during the parMERASA project [UBG$^+$13]. In all phases, standard UML tools may be used to handle the APD. Furthermore, the APD is additionally represented in XML to facilitate the modification of a particular model.

4.1 Tools for Phase I

Several tools were developed by Rapita Systems in their Rapita Verification Suite[5] (RVS) during parMERASA project, e.g. they provide a dependency tool. It utilizes a trace of a program execution to show all accesses to shared variables and the order of these accesses. Additionally, it supports a static mode to just make a list of all occurrences of shared variables.

A measurement-based WCET tool like RapiTime or a static timing analysis tool like OTAWA [BCRS11] has to be employed to determine the WCET of the different program segments.

Unfortunately, until now the code has to be analyzed manually to find situations where PDPs could be applied. We are currently discussing how this could work semi-automatically. A

[5]Homepage: `http://www.rapitasystems.com/` Licence: proprietary; tools for parallel analysis may not have been released yet. RVS includes e.g. RapiTime, RapiCheck, RapiTask and RapiCover.

first step could be the segmentation of code into different pieces based on the obtained data dependencies and existing control structures (conditional blocks, loops, etc.). The extracted segments, combined with the data dependencies, can be applied to suggest parallel sections.

Furthermore, tools for automatic parallelization may be utilized in an assisting role to find situations for parallel execution, see [KVP11, CMM10] for examples.

4.2 Tools for Phase II

To facilitate phase II, a speedup approximation and APD optimization tool has been developed, which is described in [JSK$^+$14]. It is open source and can be downloaded on GitHub[6]. The tool takes a XML file (representing an APD), a list of functions and all global variables accessed by the passed functions. During execution, it applies a genetic algorithm to find optimized APDs. Finally, we get the Pareto front (cf. section 2).

4.3 Tools for Phase III

At the implementation, we see potential to generate code (semi-)automatically: First, for the synchronization of shared variables, mutator methods (`get-/set-`functions) may be automatically generated including the necessary synchronization like e.g. locks or non-blocking functions (see [Her91, HLM03]).

Second, we developed a library composed of timing-analysable algorithmic skeletons (TAS) [SFJU15], which implement PDPs applicable for hard real-time systems [GJU13]. Thereby, these PDPs are the same as the ones usable in APDs. The library is open source[7], a detailed description how to use it can be found in [JSK$^+$14].

Combining the APD (in XML format) with our timing-analysable algorithmic skeletons (TAS), we have the idea that the skeletons could be automatically placed in code where PDPs are applied. Therefore, we plan to use the annotations of phase I, which tag the location of possible parallelisms. Applying the results of phase II, the PDPs may be switched on or off and a tool replaces the corresponding source code with code calling our library. Hence, the skeleton library can be seen as basis for a tool that provides support for implementing parallelization.

4.4 Tools for Phase IV

In this new phase, the functional and timing correctness of the parallel program is analysed. The static analysis tool OTAWA and the measurement-based tool RapiTime were extended

[6]Homepage: `https://www.github.com/parmerasa-uau/parallelism-optimization/` Licence: GNU LGPL v3

[7]Homepage: `https://www.github.com/parmerasa-uau/tas/` Licence: GNU LGPL v3

for the timing analysis of parallel code. These extensions are described in detail in [par14]. Here, we give a short overview on further extensions, which are described in the same deliverable: RapiCheck allows to check the functional equivalency between the parallel and sequential program and thus to verify if the parallelization was performed correctly. In case of a discovered functional failure, the trace viewing tool RapiTask may help to locate it and therefore to determine the step of the parallelization process where the failure occurs. Finally, RapiCover provides code coverage functionality to expose how thoroughly the code is tested. All these tools enable analysis of the resulting code of phase III and can be applied for measuring the quality of the implemented parallelization (i.e. speedups, holding the deadlines, constraint checking and code coverage).

The results determined with these tools show the real behaviour of the parallelized program on the parallel platform. Therefore, they form a good basis for an improved optimization in phase II.

5 Conclusion and Outlook

We presented an extension of the pattern-based parallelization approach which was originally introduced by Jahr et al. [JGU13b, JGU13a]. With the introduction of phase III, more attention is spent to the implementation. The additional phase for quality assurance checks the functional and timing correctness. Its results may be employed to iteratively improve the parallel program.

Although there is not yet a tool available to fully automate the parallelization process or the revealing of potential parallelism, we gave an overview over several assisting tools:
For **phase I**, there is a tool finding dependencies and creating a XML file of them. Further tools exist for timing analysis. We developed a tool for **phase II** which takes a highly parallel APD together with the WCETs of the program segments and a list of dependencies to find optimal APDs. At the implementation in **phase III**, we suggest using a code generator for the synchronization functions. Additionally, our algorithmic skeleton library eases the implementation of PDPs in the source code. We plan a tool to automate this process. In **phase IV**, a timing analysis of the parallel program is done which may lead to further improvements starting from phase II.

Altogether, we plan to further improve our tools and to develop some more tools to enable a (semi-)automated parallelization with our extended parallelization approach.

Acknowledgments

The research leading to these results has received funding from the European Union Seventh Framework Programme under grant agreement no. 287519 (parMERASA). This project ended in september 2014.

References

[BCRS11] Clément Ballabriga, Hugues Cassé, Christine Rochange, and Pascal Sainrat. OTAWA: An Open Toolbox for Adaptive WCET Analysis. In *Software Technologies for Embedded and Ubiquitous Systems*, volume 6399 of *LNCS*, pages 35–46. Springer Berlin Heidelberg, 2011.

[CMM10] Daniel Cordes, Peter Marwedel, and Arindam Mallik. Automatic parallelization of embedded software using hierarchical task graphs and integer linear programming. In *2010 IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*, pages 267–276. IEEE, 2010.

[Fos95] Ian Foster. *Designing and Building Parallel Programs: Concepts and Tools for Parallel Software Engineering*. Addison-Wesley Longman, Boston, MA, USA, 1995.

[GJU13] Mike Gerdes, Ralf Jahr, and Theo Ungerer. parMERASA Pattern Catalogue. Timing Predictable Parallel Design Patterns. Technical Report 2013-11, Department of Computer Science, University of Augsburg, Augsburg, Germany, 2013.

[Her91] Maurice Herlihy. Wait-free synchronization. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 13(1):124–149, 1991.

[HLM03] Maurice Herlihy, Victor Luchangco, and Mark Moir. Obstruction-free synchronization: Double-ended queues as an example. In *Proceedings of the 23rd International Conference on Distributed Computing Systems*, pages 522–529. IEEE, 2003.

[JFG$^+$14] Ralf Jahr, Martin Frieb, Mike Gerdes, Theo Ungerer, Andreas Hugl, and Hans Regler. Paving the Way for Multi-cores in Industrial Hard Real-time Control Applications. In *WiP at 9th IEEE International Symposium on Industrial Embedded Systems (SIES)*, Pisa, Italy, 2014.

[JFGU14] Ralf Jahr, Martin Frieb, Mike Gerdes, and Theo Ungerer. Model-based Parallelization and Optimization of an Industrial Control Code. In *Dagstuhl-Workshop MBEES: Modellbasierte Entwicklung eingebetteter Systeme X, Schloss Dagstuhl, Germany, 2014, Tagungsband Modellbasierte Entwicklung eingebetteter Systeme*, pages 63–72, fortiss GmbH, Munich, March 2014.

[JGU13a] Ralf Jahr, Mike Gerdes, and Theo Ungerer. On Efficient and Effective Model-based Parallelization of Hard Real-Time Applications. In *Dagstuhl-Workshop MBEES: Modellbasierte Entwicklung eingebetteter Systeme IX*, pages 50–59, fortiss GmbH, Munich, April 2013.

[JGU13b] Ralf Jahr, Mike Gerdes, and Theo Ungerer. A pattern-supported parallelization approach. In *International Workshop on Programming Models and Applications for Multicores and Manycores*, PMAM '13, pages 53–62, New York, NY, USA, 2013. ACM.

[JGU$^+$14] Ralf Jahr, Mike Gerdes, Theo Ungerer, Haluk Ozaktas, Christine Rochange, and Pavel G. Zaykov. Effects of Structured Parallelism by Parallel Design Patterns on Embedded Hard Real-time Systems. In *20th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, Chongqing, China, August 2014.

[JSK$^+$14] Ralf Jahr, Alexander Stegmeier, Rolf Kiefhaber, Martin Frieb, and Theo Ungerer. User Manual for the Optimization and WCET Analysis of Software with Timing Analyzable Algorithmic Skeletons. Technical Report 2014-05, University of Augsburg, 2014.

[KVP11] Stefan Kempf, Ronald Veldema, and Michael Philippsen. Is There Hope for Automatic Parallelization of Legacy Industry Automation Applications? In GI, editor, *Proceedings of the 24th Workshop on Parallel Systems and Algorithms (PARS)*, pages 80–89, 2011.

[MBC11] Anne Meade, Jim Buckley, and John J. Collins. Challenges of Evolving Sequential to Parallel Code: An Exploratory Review. In *Proceedings of the 12th International Workshop on Principles of Software Evolution and the 7th Annual ERCIM Workshop on Software Evolution*, IWPSE-EVOL '11, pages 1–5, New York, NY, USA, 2011. ACM.

[MMS99] Berna L. Massingill, Timothy G. Mattson, and Beverly A. Sanders. Patterns for Parallel Application Programs. *Proceedings of the 6th Conference on Pattern Languages of Programs (PLoP'99)*, 1999.

[MSM04] Timothy G. Mattson, Beverly A. Sanders, and Berna L. Massingill. *Patterns for parallel programming*. Addison-Wesley Professional, first edition, 2004.

[par14] Report on support of tools for case studies. Deliverable 3.12 of the parMERASA project, September 2014. `http://www.parmerasa.eu/files/deliverables/Deliverable_3_12.pdf`.

[PQnZ+14] Miloš Panić, Eduardo Quiñones, Pavel G. Zaykov, Carles Hernandez, Jaume Abella, and Francisco J. Cazorla. Parallel Many-core Avionics Systems. In *14th International Conference on Embedded Software*, EMSOFT '14, pages 26:1–26:10, New York, NY, USA, 2014. ACM.

[SFJU15] A. Stegmeier, M. Frieb, R. Jahr, and T. Ungerer. Algorithmic Skeletons for Parallelization of Embedded Real-time Systems. In *3rd Workshop on High-performance and Real-time Embedded Systems (HiRES)*, 2015.

[UBG+13] T. Ungerer, C. Bradatsch, M. Gerdes, F. Kluge, R. Jahr, J. Mische, J. Fernandes, P.G. Zaykov, Z. Petrov, B. Boddeker, S. Kehr, H. Regler, A. Hugl, C. Rochange, H. Ozaktas, H. Casse, A. Bonenfant, P. Sainrat, I. Broster, N. Lay, D. George, E. Quinones, M. Panic, J. Abella, F. Cazorla, S. Uhrig, M. Rohde, and A. Pyka. parMERASA – Multi-core Execution of Parallelised Hard Real-Time Applications Supporting Analysability. In *2013 Euromicro Conference on Digital System Design (DSD)*, pages 363–370, Sept 2013.

[UBG+15] T. Ungerer, C. Bradatsch, M. Gerdes, F. Kluge, R. Jahr, J. Mische, A. Stegmeier, M. Frieb, J. Fernandes, P.G. Zaykov, Z. Petrov, B. Boddeker, S. Kehr, H. Regler, A. Hugl, C. Rochange, H. Ozaktas, H. Casse, A. Bonenfant, P. Sainrat, I. Broster, N. Lay, D. George, E. Quinones, M. Panic, J. Abella, F. Cazorla, S. Uhrig, M. Rohde, and A. Pyka. Experiences and Results of Parallelisation of Industrial Hard Real-time Applications for the parMERASA Multi-core. In *3rd Workshop on High-performance and Real-time Embedded Systems (HiRES)*, 2015.

[WEE+08] Reinhard Wilhelm, Jakob Engblom, Andreas Ermedahl, Niklas Holsti, Stephan Thesing, David Whalley, Guillem Bernat, Christian Ferdinand, Reinhold Heckmann, Tulika Mitra, Frank Mueller, Isabelle Puaut, Peter Puschner, Jan Staschulat, and Per Stenström. The Worst-case Execution-time Problem – Overview of Methods and Survey of Tools. *ACM Transactions on Embedded Computing Systems (TECS)*, 7(3):36:1–36:53, May 2008.

Parallelisierung von Embedded Realtime Systemen: Probleme und Lösungsstrategien in Migrationsprojekten

Marwan Abu-Khalil

Siemens AG, Energy Automation
13629 Berlin, Germany
marwan.abu-khalil@siemens.com

Abstract: Dieser Artikel extrahiert Erfahrungen aus einer Reihe erfolgreicher sowie gescheiterter industrieller Parallelisierungsprojekte, bei denen Embedded Realtime Systeme von Single-Core CPUs auf Multi-Core SMP-Plattformen portiert wurden. Die Kernthese des Vortrages lautet, dass die Parallelisierung von Embedded Realtime Systemen spezifischen Herausforderungen gegenübersteht, die bei anderen System-Klassen, wie Server- oder Desktop-Software, nur eine untergeordnete Relevanz haben. Der Artikel analysiert und kategorisiert diese spezifischen Herausforderungen. Als Resultat werden allgemeingültige Herangehensweisen vorgeschlagen, die zu erfolgreicher Parallelisierung im Embedded-Bereich führen.

1 Einleitung

Heutige Embedded Systeme sind überwiegend als Multi-Thread Systeme auf einer Single-Core CPU realisiert. Dies prägt die gesamte Software-Architektur und insbesondere das Nebenläufigkeitskonzept, welches sich daher einer Migration auf eine Multi-Core CPU mit einem SMP Betriebssystem widersetzt. Eine SMP-Parallelisierung wird somit ein riskantes Unterfangen, bei dem das Systemverhalten komplett verändert wird. Aufgrund der begrenzten Taktratensteigerung aktueller CPUs und der Allgegenwart von Multi-Core CPUs ist jedoch die Embedded-Welt gezwungen, diese über Jahre gewachsenen Systeme auf Multi-Core Plattformen zu portieren. Typische Software-Architekturen von Embedded Realtime Systemen basieren auf Annahmen über das Scheduling-Verhalten auf einer Single-Core CPU und verwenden daher oftmals Konzepte wie die implizite Synchronisation durch Interrupt-Locks oder die Steuerung zeitlicher Abläufe durch Realtime-Prioritäten. Diese Konzepte lassen sich jedoch auf SMP Systemen nicht nutzen. Embedded Systeme stehen daher vor besonderen Herausforderungen bei der Multi-Core Portierung. Dieser Artikel analysiert und kategorisiert diese spezifischen Probleme und präsentiert allgemeine Maßgaben für die erfolgreiche SMP-Parallelisierung von Embedded Realtime Systemen.

Der Artikel basiert auf der Analyse beispielhafter erfolgreicher sowie gescheiterter Projekte. Die Auswahl der Systeme spiegelt die typischen und verbreiteten Software-Architekturen im Feld der Embedded Realtime Systeme wider.

Die These über die spezifischen Schwierigkeiten der Parallelisierung von Embedded Systemen wird in Kapitel 2 dargelegt. In Kapitel 3 werden drei Beispielprojekte präsentiert: Die erfolgreiche Parallelisierung eines Systems aus der Energie-Automatisierung, sowie als Gegenpol zwei teilweise gescheiterte Projekte aus dem Energie- bzw. dem Telekommunikationsumfeld. An diesen Beispielen werden Erfolgsrezepte sowie Fallstricke und Risiken solcher Parallelisierungen erkennbar. In Kapitel 4 werden die spezifischen Probleme analysiert und kategorisiert und die jeweiligen Lösungsvorschläge erarbeitet. Kapitel 5 fasst die Lösungsansätze zusammen.

2 These: Probleme der Embedded Realtime Parallelisierung

Die Parallelisierung von Embedded Realtime Systemen im Zuge der Portierung von Single-Core auf Multi-Core Hardware scheitert oftmals an den im Embedded-Bereich etablierten Software-Architekturen. Daher sind spezifische Herangehensweisen erforderlich, um Embedded Systeme erfolgreich zu parallelisieren. Zwei Problembereiche sind dabei ausschlaggebend:

1. Architekturen von Embedded Systemen, die über Jahre auf Single-Core CPUs entwickelt wurden, basieren oft auf Annahmen über das Scheduling-Verhalten, die implizit voraussetzen, dass die Software sequentiell auf einer Single-Core CPU abgearbeitet wird.
2. Im Embedded-Umfeld sind aggressive Optimierungen und eine hardwarenahe Denkweise weit verbreitet. Abstraktionen von Betriebssystemen oder anderen Laufzeitumgebungen werden oftmals aus Performance-Gründen umgangen.

Diese Problembereiche haben folgende technische und nicht-technische Aspekte:

- Embedded Systeme verwenden, anders als Desktop- oder Server-Software, in hohem Maße implizite Synchronisation durch Interrupt-Locks, diese lässt sich jedoch in einem SMP-System nicht nutzen (siehe 4.1).
- In Realtime Systemen wird Synchronisation oft durch Prioritäten von Threads realisiert, dies lässt sich nicht auf SMP-Systeme übertragen (siehe 4.2).
- Ein typischer Embedded-Programmierer möchte zeitliche Abläufe seines Systems in allen Details kontrollieren, oft wird dafür das prioritätsbasierte Scheduling missbraucht. Dies widerspricht jedoch der Philosophie eines SMP Betriebssystems (siehe 4.3).
- Die Denkweise von Embedded-Programmierern ist oftmals sehr hardwarenah. Dies führt zu riskanten Optimierungen und somit zu schwer zu findenden Fehlern bei der SMP-Portierung (siehe 4.4).

3 Beispiel-Projekte

Hier führe ich exemplarisch drei Beispiele erfolgreicher sowie gescheiterter Projekte an, um meine These zu untermauern. Die unterschiedlichen Parallelitätskonzepte, die in diesen Projekten verfolgt wurden, werden jeweils in einem abstrakten Diagramm dargestellt, um sie mit einer „idealen“ SMP-Parallelisierung vergleichen zu können.

3.1 Projekt 1: Hard-Realtime System zum Schutz von Stromverteilnetzen

Dieses auf dem RT-OS (Realtime-Betriebssystem) VxWorks basierende System, das der Kontrolle und dem Schutz von Stromnetzen dient, wurde von einem Single-Core PPC auf einen ARM Dual-Core (Altera Cyclone V SoC) portiert. Ziel der Portierung war eine signifikante Steigerung der Performance. Das System war schon vor der Portierung in viele VxWorks Tasks (vergleichbar mit Threads) strukturiert, die nach strengen Prioritätsvorgaben geschedult wurden. Zwei Herausforderungen standen bei der Migration im Vordergrund:

1. Ein weitgehendes Erhalten des Laufzeitverhaltens des vorhandenen Systems, um das Risiko der Migration zu begrenzen.
2. Das Ersetzen der impliziten Synchronisation durch explizite Synchronisation.

Konzept zu Anforderung 1:

Situation: Das System besteht aus einem IO-Thread („Input-Output“), der zyklisch Daten aus der Außenwelt abholt und im System verteilt, sowie einer Gruppe von BL-Threads („Business-Logik“), in denen Algorithmen ablaufen, die diese Daten verarbeiten. Die Kommunikation zwischen IO-Thread und BL-Threads erfolgt über Ringbuffer. Das Nebenläufigkeitsmodell im Single-Core Fall stellt sicher, dass die Algorithmen in den BL-Threads in einer fest vorgegebenen Reihenfolge nach jedem IO-Zyklus ablaufen. Dieser Ablauf ist auf Basis der Prioritäten des Realtime-OS organisiert.

Problem: Auf einem SMP-System kann durch Prioritäten diese Ablaufreihenfolge nicht mehr sicher gestellt werden, weil Threads unterschiedlicher Priorität gleichzeitig auf verschiedenen CPUs laufen können („CPU“ und „Core“ sind im Folgenden stets gleichbedeutend).

Lösung: Die Anforderung, das Laufzeit- und Scheduling-Verhalten möglichst wenig zu verändern, hat zu einer Software-Architektur geführt, die weite Teile des Systems quasi wie auf einem Single-Core ablaufen lässt. Alle BL-Threads sind an Core-0 gebunden, dadurch verhalten sich diese Threads untereinander so, als liefen sie auf einem Single-Core. Das Laufzeitverhalten der Applikationslogik bleibt damit weitgehend unverändert. Nur der CPU-hungrige IO-Thread wird an den zweiten Core gebunden. Insgesamt führen diese Maßnahmen zu einer „sanften“ Migration, mit relativ geringem Umbau im Code sowie einer beschränkten Veränderung des Laufzeit-Verhaltens, da lediglich der Datenaustausch zwischen dem IO-Thread und den BL-Threads neu organisiert werden muss.

Konzept zu Anforderung 2:

Situation: In Embedded Systemen, die auf Single-Core CPUs laufen, ist das sogenannte implizite Locking durch das Unterdrücken von Interrupts eine weit verbreitete, etablierte und effiziente Synchronisationsstrategie (siehe 4.2). Im Ausgangsystem wurden Critical-Sections an vielen Stellen durch die entsprechenden VxWorks APIs für Interrupt-Locks oder Task-Locks synchronisiert.

Problem: Auf einem SMP-System erzwingen Interrupt-Sperren keinen gegenseitigen Ausschluss (siehe 4.2).

Das Zielsystem verwendet VxWorks 6.9 in einer SMP-Ausprägung. Hier sind diese impliziten Synchronisationsmechanismen auch in der OS-API nicht mehr verfügbar. Die Empfehlung des Betriebssystemherstellers lautet, Interrupt-Locks durch Spinlocks zu ersetzen und Task-Locks durch Semaphore zu ersetzen. Eine durchgängige Befolgung dieser Vorgabe hätte jedoch folgende Konsequenzen: Der massive Einsatz von Semaphoren hätte das System stark verlangsamt, da der Aufruf eines Semaphors ein vergleichsweise teuer Betriebssystem-Aufruf ist. Der massenhafte Einsatz von Spinlocks auf dem Dual-Core hätte das erreichbare Maß der Parallelisierung unnötig stark begrenzt, da ein Spinlock im Wartezustand die CPU nicht freigibt.

Lösung: Es war somit keine schematische Ersetzung von impliziten durch explizite Synchronisationsmechanismen möglich. In diesem Projekt wurde daher für jede einzelne implizite Critical-Section untersucht, wie diese effizient und sicher auf das SMP-System portiert werden kann. Dabei standen die folgenden Optionen zur Verfügung, für die hier jeweils ein Beispiel genannt wird:

1. Semaphor: Die Kommunikation zwischen Threads auf unterschiedlichen Cores ist durch Semaphore synchronisiert.
2. Spinlock: Die Synchronisation zwischen Algorithmen der BL-Threads ist durch Spinlocks realisiert. Dies kann als problematische Low-Level Optimierung gesehen werden, weil hier die Eigenschaft der Spinlock-Implementierung ausgenutzt wurde, sich Core-Lokal wie ein Interrupt-Lock zu verhalten.
3. Lockfreie Datenstruktur: Der Datenaustauch zwischen dem IO-Thread und den BL-Threads wird über eine lockfreie Ringbuffer-Implementierung realisiert, da dies der performancekritischte Teil des Systems ist. Bemerkenswert ist, dass aus Performancegründen selbst die vom CPU-Hersteller geforderten Memory-Barriers teilweise nicht verwendet werden, weil in Tests kein Fehlverhalten nachgewiesen werden konnte. Auch dies ist ein Beispiel für riskante Mikro-Optimierungen in Embedded Systemen.

Bewertung: In einem „idealen“ SMP-Konzept verteilt der Scheduler des Betriebssystems alle Threads frei über alle vorhandenen CPUs, um zu einer optimalen Auslastung zu kommen. Im Vergleich dazu ist das Nebenläufigkeitskonzept dieses Projektes mit wesentlichen Einschränkungen verbunden. Der OS-Scheduler ist bezüglich der Zuteilung der verschiedenen CPUs außer Kraft gesetzt. In der folgenden schematischen Graphik werden die Abweichungen vom Ideal deutlich gemacht:

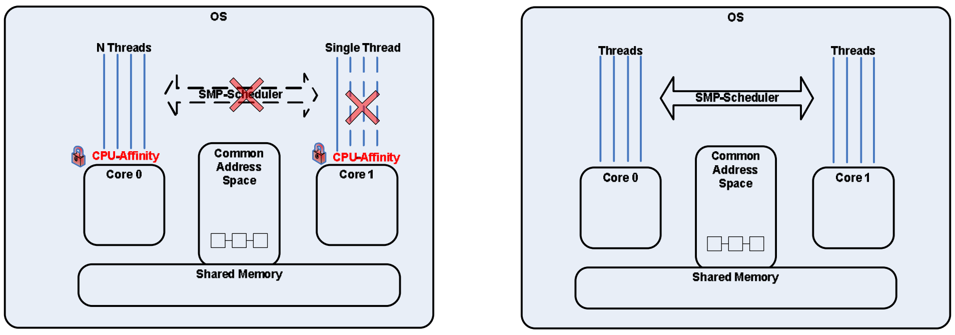


Abb. 1: Projekt 1 (links) im Vergleich zum "idealen SMP" (rechts)

Das Projekt kann bezüglich des Parallelitätskonzeptes daher nur als „teilweise erfolgreich“ bewertet werden. Die einfache Migration wurde mit folgenden gravierenden Defiziten in den nichtfunktionalen Eigenschaften der Architektur erkauft:

1. Die starre Bindung von Threads an CPU-Cores führt zu einer suboptimalen Performance, da Idle-Zeiten auf einem Core nicht von beliebigen Threads genutzt werden können.
2. Eine solche Software-Architektur skaliert nicht, da z.B. ein Vier-Kern Prozessor damit nicht ausgenutzt werden könnte.

3.2 Projekt 2: Parallelisierung einer Soft-Realtime Software für Power-Quality

Das hier geschilderte System dient der Qualitätskontrolle und Fehlererkennung in Stromverteilnetzen.

Die Software dieses Systems lief ursprünglich auf einem Blackfin Single-Core unter dem Betriebssystem µC/OS. Die geplante Dual-Core Migration hatte eine bessere Performance zum Ziel, die neue Features ermöglichen sollte. In diesem Projekt ist mit zwei unterschiedlichen Ansätzen versucht worden, die Portierung auf eine Dual-Core CPU zu realisieren. Der erste Ansatz ist gescheitert.

Im ersten Ansatz war eine Migration auf SMP-Linux und einen ARM Dual-Core mit folgenden Konzepten geplant:

- Ersetzung der µC/OS Tasks durch Linux P-Threads.
- Restrukturierung des Nebenläufigkeitskonzeptes: Ursprünglich hat eine zentrale „schwergewichtige“ Task an einem Synchronisationsprimitiv („Event“) gewartet, das viele unterschiedliche Ereignisse signalisieren konnte (ähnlich einem select Systemcall). Diese Task sollte in viele einzelne Threads aufgebrochen werden, die jeweils eine spezifische Aufgabe haben.
- Ersetzung der µC/OS Events durch P-Thread Condition-Variable.

- Einsatz der Linux Realtime Fähigkeiten, um dem ursprünglichen Realtime-Scheduling in µC/OS nahe zu kommen.
- Ersetzung impliziter Synchronisation durch explizite Synchronisation: Interrupt-Locks und Außerkraftsetzung des OS-Schedulers wurden im Ursprungssystem für die Realisierung von gegenseitigem Ausschluss benutzt. Dies wäre im SMP-Fall nicht mehr möglich gewesen (siehe 4.1).

Dieser Ansatz wurde nach einem Jahr Projektlaufzeit aus folgenden Gründen verworfen:

1. Das Nebenläufigkeitskonzept der ursprünglichen Software basierte wesentlich auf Annahmen über die sequentielle Abarbeitung der Threads auf dem Single-Core, die gewünschte Abfolge wurde durch Prioritäten gewährleistet. Dies ließ sich jedoch im SMP-Fall nur schwer nachbilden und hätte grundlegende Restrukturierungen in der Software-Architektur erfordert.
2. Die ursprüngliche Software war stark an den proprietären Synchronisationsmitteln von µC/OS orientiert (Events, Queues). Die Ersetzung dieser Mittel durch auf P-Threads basierenden Ansätzen (Condition-Variablen, blockierende Queues), veränderte die Semantik des Systems und war daher schwieriger als erwartet.

Man hat sich dann in einem zweiten Migrationsversuch für eine Hardware- und Software-Architektur entschieden, die auf den in der Vergangenheit des Produktes bekannten und etablierten Technologien basierte, um das technische Risiko zu minimieren. Das System wurde auf einen Blackfin Dual-Core Prozessor portiert, als Betriebssystem wurde weiterhin µC/OS genutzt, obwohl es dort keine Unterstützung für SMP auf diesem Prozessor gibt. Daher wurden zwar bestimmte Aufgaben auf den zweiten Core ausgelagert, sie laufen jedoch dort ohne ein Betriebssystem („Bare-Metal“).

Die schematische Architektur zeigt die Abweichungen vom „Ideal“ (vgl. Abb. 1):

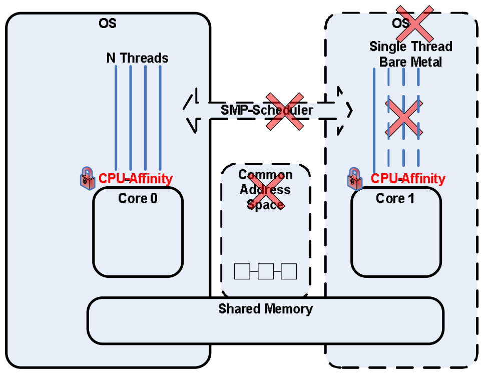


Abb. 2: Projekt 2, zweiter Core ohne OS

Bewertung: Dieses Projekt muss unter dem Gesichtspunkt der Parallelisierung als teilweise gescheitert angesehen werden, weil die gewählte Architektur die Vorteile der Multi-Core CPU nicht effizient ausnutzt. Es wurde weder ein klassischer SMP-Ansatz noch ein asymmetrischer aber zumindest betriebssystemunterstützter Ansatz realisiert. Stattdessen wurde eine starre Aufgabenaufteilung vorgenommen, ohne OS-Unterstützung auf dem zweiten Core. Dies bringt Performance-Nachteile mit sich, da auf dem zweiten Core keine weiteren Tasks geschedult werden können, und es impliziert ein komplexes Programmiermodell, da die Kommunikation zwischen Software auf unterschiedlichen Cores nicht durch Betriebssystemabstraktionen unterstützt wird.

3.3 Projekt 3: Multi-Core-Portierung einer Telekommunikationssoftware

Dieses Parallelisierungsprojekt bezog sich auf ein Embedded System, das von Telekommunikationsprovidern genutzt wird, um DSL-Kommunikationsstrecken zu Endanwendern zu realisieren. Die Software ist komplett innerhalb eines Linux-Kernels realisiert. Der Grund, die Trennung von Kernel-Address-Space und User-Address-Space, die Linux anbietet, nicht zu nutzen, sind die Performance-Kosten der Systemcalls, die auf diese Weise vermieden werden sollen. Das System wurde von einem Single-Core Power-PC auf eine ARM Dual-Core CPU portiert. Die über Jahre gewachsene Software nutzt viele der Nebenläufigkeitsfeatures und Synchronisationsprimitive, die im Linux-Kernel existieren, was zu einem komplexen Design geführt hat. Der Aufwand dafür, dennoch bei SMP auf dem Dual-Core ein korrektes Verhalten der Anwendung zu erzielen, hat sich im Verlauf der Portierung als so hoch herausgestellt, dass schließlich auf die Nutzung des zweiten Cores komplett verzichtet wurde. Somit läuft die gesamte Software nun trotz vieler Threads auf nur einem Core der Dual-Core CPU.

Schematische Architektur und Abweichung vom idealen SMP (vgl. Abb. 1):

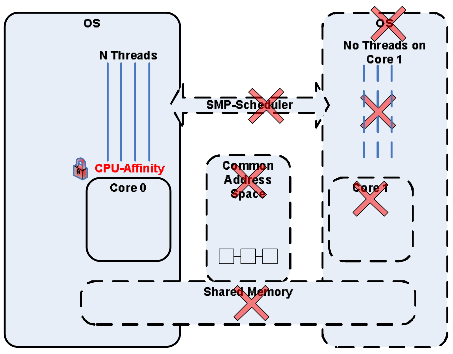


Abb. 3: Projekt 3, Dual-Core ohne Nutzung des zweiten Cores

Bewertung: Dieses Projekt ist aus Perspektive der Parallelisierung als komplett gescheitert anzusehen. Grund für das Scheitern ist eine Software-Architektur, die die Abstraktionen eines Betriebssystems nicht wie vorgesehen nutzt, nur um dadurch kurzfristig einen Performance-Gewinn zu erzielen. Die Architektur war im Single-Core Umfeld möglicherweise effizient, für die Dual-Core Nutzung aber nicht mehr tragfähig, weil sie aufgrund der mangelnden Abstraktion von Hardware und OS zu komplex wurde. Wäre von Anfang an das gesamte Multithreading auf der P-Thread Schnittstelle des OS aufgebaut worden, wäre eine Dual-Core Portierung aussichtsreicher gewesen.

4 Probleme und Lösungen bei der Embedded Parallelisierung

Ausgehend von den oben genannten Beispielen, die charakteristisch für die Situation in der Entwicklung von Embedded Systemen sind, werden nun die wesentlichen Herausforderungen bei der Parallelisierung im Zuge einer Portierung von Single-Core Systemen auf SMP-Hardware im Embedded Systeme Bereich analysiert, und es werden Lösungsstrategien dafür vorgeschlagen. Die Probleme haben neben ihrem technischen Aspekt, der sich mit konkreten Implementierungsstrategien lösen lässt, oft auch einen „Mindset-Anteil", der sich eher auf Denkweisen oder eine „spezifische Kultur" im Embedded Umfeld bezieht. Die folgende Tabelle gibt eine Übersicht. Anschließend wird jedes Problem detailliert behandelt.

Ref	***Situation***	***Problem***	***Lösung***	***Beispiel***
4.1	Impl. Synch. Interrupt-Lock	Kein SMP Mutex	Nur Expl. Synch. in Anwend.-SW	Proj. 1: BL-Blöcke Synchronisation
4.2	Impl. Synch. Prioritäten	Kein SMP Mutex	Prioritäten nie für Mutex nutzen	Projekt 2: Startup-Thread
4.3	Zeitl. Ablauf via Prioritäten	Nicht SMP-fähig	Kontrolle an Scheduler abgeben	Projekt 1: BL-Reihenfolge
4.4	Real-Time Optimierungen	Komplexität verhind. Portierg.	Datenkapselung, OS-Abstraktion	Projekt 1: Linux-Kernel Applikation
4.5	Tools f. Server optimiert	Embedded: traditionelle Para.	z.B. EMB² und MTAPI nutzen	Work-Stealing nicht in Embedded OS
4.6	Ausbildung Entwickler	Synchronisation Memory-Modelle	Gezielte Aus- und Weiterbildung	Erfahrungswerte aus Trainertätigkeit
4.6	Management: Naive Sicht	Falsche Planung	Bewusstsein für Komplexität	Projekt 3: Komplett unterschätzt

Tab 1: Probleme der Embedded Parallelisierung

4.1 Implizite Synchronisation durch Abschalten von Interrupts

Situation: Implizite Synchronisation kann definiert werden, als der Schutz einer Critical-Section auf Basis von Annahmen über das Scheduling-Verhalten des Betriebssystems. Durch das Abschalten von Interrupts lassen sich die beiden folgenden Varianten herstellen:

1. Sowohl die Interrupt-Behandlung (ISRs) als auch der Betriebssystem-Scheduler werden unterdrückt („Interrupt-Lock").
2. Lediglich der Betriebssystem-Scheduler wird unterdrückt, aber die ISRs werden zugelassen. Dies ist eine sanftere Variante des obigen Ansatzes („Task-Lock").

Diese Art der impliziten Synchronisation ist auf einem Single-Core sehr effizient, weil sie mit dem Abschalten von Interrupts im Wesentlichen auf Hardware-Ebene realisiert werden kann. Weder sind teure System-Calls mit Kontext-Switch in den OS-Kernel erforderlich noch Scheduling-Vorgänge des OS, die beispielsweise beim Akquirieren eines Semaphors nötig werden können. In Single-Core Embedded Realtime Systemen ist dieses Vorgehen daher gängige und etablierte Praxis.

Problem: Interrupt-Locks lassen sich nicht in ein SMP-System übertragen, da sie auf der Annahme basieren, dass der exklusive Besitz eines CPU-Cores garantiert, dass kein anderer Code gleichzeitig ausgeführt wird. Diese Annahme ist jedoch in einem Multi-Core SMP-System nicht gültig, da zu dem Zeitpunkt an dem die Interrupts unterdrückt werden, auf einem anderen CPU-Core bereits der konkurrierende Code ausgeführt werden kann.

Lösung: Auf implizite Synchronisation durch Interrupt-Locks sollte im Anwendungscode prinzipiell verzichtet werden. Es handelt sich hier um ein Low-Level Synchronisationsmittel (z.B. für Treiber), dessen Verwendung in Anwendungscode auf mangelhaftes Design hindeutet. Da in Zukunft immer mehr Systeme auf Multi-Core-Hardware portiert werden müssen, sollte auch im Single-Core Fall auf Interrupt-Locks verzichtet werden, da sie sich nur schwer wieder ausbauen lassen.

Beispiel: Im Projekt 1 waren weite Teile der Synchronisation durch Interrupt-Locks realisiert. Es war kein schematisches Ersetzen möglich, sondern jede solche Critical-Section hat eine spezifische Umbaumaßnahme erfordert (siehe 3.1).

4.2 Implizite Synchronisation durch prioritätsbasiertes Realtime Scheduling

Situation: Prioritätsbasierte RT-OS stellen sicher, dass ein Thread nur von höher priorisieren Threads unterbrochen wird. Das wird oft dazu genutzt, eine Art gegenseitigen Ausschluss durch Prioritäten zu realisieren, indem ein auszuschließender Thread gleich oder niedriger priorisiert wird.

Problem: Dieser Ansatz ist nicht auf SMP übertragbar, da verschiedene Threads unabhängig von ihrer Priorität gleichzeitig auf unterschiedlichen CPUs laufen können. Die Gefahr bei einer SMP-Portierung ist, dass zunächst kein technisches Problem sichtbar wird, aber das Systemverhalten trotzdem korrumpiert wird. Dadurch ist diese Art der impliziten Synchronisation noch heimtückischer als das Interrupt-Locking.

Lösung: Prioritäten in RT-OS sollten prinzipiell nicht zur Synchronisation verwendet werden. Es sollten immer explizite Synchronisationsmechanismen genutzt werden.

Beispiel: Der Startup-Thread in Projekt 2 startet niedriger priorisierte Threads, die erst laufen dürfen nachdem der Startup-Thread beendet ist. Dieser Ablauf ist im SMP-Fall nicht gewährleistet.

4.3 Organisation zeitlicher Abläufe durch Realtime-Prioritäten

Situation: Ein typischer Embedded Entwickler ist bestrebt, zeitliche Abläufe innerhalb der Applikationslogik, wie Abarbeitungsreihenfolgen, Synchronisation und Datenfluss, explizit zu kontrollieren und vorherzusehen**,** um so die begrenzten Ressourcen optimal auszunutzen. Anders als beispielsweise ein typischer Programmierer eines Programmes, das in einem Java Application-Server läuft, vermeidet er es, Entscheidungen über Ablaufreihenfolgen einem Scheduler zu überlassen. Oft wird das prioritätsbasierte Scheduling dafür missbraucht, diese zeitlichen Abläufe zu organisieren.

Problem: Dies widerspricht fundamental der Philosophie eines SMP-Betriebssystems oder moderneren Ansätzen feingranularer Parallelisierung z.B. Work-Stealing, dort kann schon aus technischen Gründen durch Prioritäten keine Ablaufreihenfolge gewährleistet werden. Da zudem ein optimaler Scheduling-Algorithmus ein NP-hartes Problem darstellt, ist es im Allgemeinen auch aus Perfomance-Gründen sinnvoller, Scheduling-Entscheidungen einem Scheduler zu überlassen, als sie explizit zu manipulieren, zumal Work-Stealing dem theoretischen Optimum relativ nahe kommt (vgl. [HER] S. 380).

Beispiel: In Projekt 1 realisieren Anwendungsthreads einen Graph von BL-Elementen, durch den Daten in einer bestimmten Reihenfolge fließen. Im Single-Core Fall ist diese Reihenfolge durch Prioritäten garantiert. Im SMP-Fall mussten alle Anwendungsthreads an einen Core gebunden werden, um den Singe-Core Ablauf zu imitieren.

Lösung: Die Embedded-Welt sollte lernen, mehr Kontrolle an Scheduler abzugeben. Software-Architekturen sollten entweder mit unterschiedlichen Reihenfolgen umgehen können, wie z.B. bei einem Server, der eingehende Requests abarbeitet, oder falls Reihenfolgen wichtig sind, diese explizit programmatisch realisieren und nicht auf Basis impliziter (Scheduling-)Annahmen. Prioritäten sollten ausschließlich genutzt werden, um zeitkritische Ereignisse vorrangig zu behandeln.

4.4 Riskante Mikro-Optimierungen zur Erfüllung von Realtime-Anforderungen

Situation: Die Denkweise von Embedded-Programmierern ist oftmals sehr hardwarenah. Dies führt zu Optimierungen, die sich auf eine bestimmte Hardware beziehen und die aus Performancegründen die Abstraktionen eines Betriebssystems umgehen. Es kommt auch vor, dass bestimmte Synchronisationsmittel, die aus Gründen der Datenkonsistenz sinnvoll wären, nicht eingesetzt werden, um die damit verbundenen Performance-Kosten zu vermeiden.

Problem: Diese Optimierungen basieren meist auf impliziten Annahmen über HW oder OS, die im SMP-Fall nicht immer gelten. Dies führt zu schwer zu findenden Problemen.

Beispiele: Die Vermeidung von System-Calls in Projekt 3 hat zu einer Linux-Kernel-Anwendung geführt. Dies hat aufgrund mangelnder Abstraktion die SMP-Portierung verhindert. Im Projekt 1 wurde die im Ringbuffer beim Lesen formal erforderliche Memory-Barrier weggelassen, da im Test kein Fehler nachweisbar war.

Lösung: Low-Level-Optimierungen sollten prinzipiell vermieden werden: OS-Abstraktionen sollten konsequent genutzt werden. Saubere Synchronisation kostet einen gewissen Teil der Performance, die ein SMP-System liefert. Dies muss bei der Dimensionierung der HW berücksichtigt werden. So sollten z.B. Memory-Barriers nicht explizit im Anwendungscode stehen sondern durch Synchronisationsprimitive ersetzt werden, denn die Barrier-Semantik ist sehr subtil und CPU-spezifisch und daher sind Memory-Barriers nicht naiv einsetzbar und nicht risikolos portierbar (vgl. [ARM] A 3.8

„Memory access order“ und [INT] 8.2 „Memory ordering“).

4.5 Tools und Paradigmen sind für Server und Desktop optimiert

Situation: Moderne Paradigmen der Parallelisierung, wie z.B. Work-Stealing, sind nicht im Embedded-Umfeld etabliert. Parallelisierungstools stammen oftmals aus dem Desktop- und Server-Bereich und sind daher nicht direkt für die Entwicklung von Embedded Systemen optimiert.

Problem: Die Embedded-Welt setzt moderne Parallelisierungsparadigmen nur zaghaft ein. Sie verharrt teilweise in traditionellen Denkmustern, dies führt zu den oben genannten Problemen der „schwergewichtigen“ starren Parallelisierung (vgl. 3.1).

Beispiele: Viele Embedded-OS wie z.B. VxWorks liefern keinen Work-Stealing Task-Scheduler mit (iOS hingegen schon, GCD). Es fehlen z.B. bei den general-purpose Work-Stealing Task-Schedulern Prioritäten. Auch klassische Thread-Bibliotheken sind nicht immer ideal für Realtime-Anforderungen, da oftmals Datenstrukturen benutzt werden, die implizit Semaphore verwenden und dynamisch Speicher allokieren (z.B. blockierende Queues auf Basis von C++ STL-Containern). In Realtime-Applikation sind jedoch unter Umständen Lock-Free und Wait-Free Datenstrukturen zu bevorzugen.

Lösung: Spezialisierte Embedded-Parallelisierungsbibliotheken wie z.B. EMB² von Siemens CT (vgl. [EMB]) schließen diese Lücke: Keine dynamische Speicherallokation, Embedded Work-Stealing Task-Scheduler, Lock-Free Datenstrukturen. EMB² entspricht überdies dem MTAPI-Standard für Embedded Task-Scheduling (vgl. [MTA]).

4.6 Denkweise von Entwicklern und Management bzgl. Parallelisierung

Situation: Eine bei Managern und Projektleitern weit verbreitete Vorstellung ist, dass eine Software, die mit vielen Threads auf einer Singe-Core CPU korrekt läuft, ohne wesentliche Umbauten auch auf einer Multi-Core CPU korrekt läuft. Dabei wird außer Acht gelassen, dass bestimmte Nebenläufigkeitsprobleme, die in einem Single-Core nur selten wahrnehmbar sind, im Multi-Core-Fall jedoch leicht zu einem inkonsistenten Systemzustand führen können (z.B. die Implikationen von Instruction-Reordering).

Auf Seiten der Entwickler gibt es oftmals ein Ausbildungsproblem. Aus meiner Erfahrung als Trainer und Berater weiß ich, dass die Bereiche Synchronisation und Memory-Modell vielen Entwicklern nur oberflächlich bekannt sind. Nur wenige der durchschnittlich ausgebildeten Entwickler können das Monitor-Pattern oder die Funktionsweise eines Spinlocks korrekt erklären oder die Semantik eines Relaxed-Consistent Memory-Modells erläutern.

Problem: Aus der vereinfachten Management-Sicht resultieren falsch geplante Projekte. Aus der mangelhaften Entwicklerausbildung resultieren der unsachgemäße Einsatz von Technologien und, was noch schwerer wiegt, nicht tragfähige Software-Architekturen.

Beispiele: Projekt 3 ist aufgrund falscher Technologie-Entscheidungen (Kernel-Applikation) in eine Schieflage geraten. In Projekt 2 wurde der Parallelisierungsaufwand unterschätzt, was zu dem zweiten Anlauf geführt hat.

Lösung: Die Entwickler-Ausbildung sollte Paradigmen und Theorie der Parallelität anhand moderner Technologien vermitteln. Manager und Entscheider sollten erkennen, dass Parallelisierung ein komplexes Unterfangen ist, das Eingriffe auf allen Ebenen des Technologie-Stacks erfordert, und dass Multi-Core Hardware Herausforderungen an die Software-Architektur stellt.

5 Fazit: Strategien für Parallelisierung von Embedded Systemen

Die folgenden Regeln sind das Kondensat der hier analysierten Erfahrungen. Sie können als Richtschnur für die erfolgreiche Parallelisierung von Embedded Systemen dienen.

1. Performance-Optimierungen in der Mikro-Ebene vermeiden, OS-Abstraktionen nutzen, Applikation von OS-Kernel trennen.
2. Explizite Synchronisation im Realtime-Embedded Bereich etablieren.
3. Parallelisierungsbibliotheken und Tools verwenden, die für Embedded Systeme optimiert sind, z.B. den EMB² Work-Stealing Task-Scheduler.
3. Schrittweise Migration von Single-Core auf SMP, Big-Bang Migration ist riskant.
4. Zeitliche Kontrolle an OS oder Scheduler abgeben, zeitliche Abläufe nicht über Prioritäten sicherstellen.
5. Ausbildung verbessern bzgl. Synchronisation, Memory-Modell, Task-Scheduling.

Literaturverzeichnis

[ARM] ARM Architecture Reference Manual, ARMv7-A and ARMv7-R edition http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.ddi0406c/index.html

[DUF] Duffy, J.; Sutter, H.: Concurrent Programming on Windows. Addison-Wesley 2008.

[EMB] EMB², https://github.com/siemens/embb

[GLS] Gleim, U.; Schüle, T.: Multi-Core Software, dpunkt 2011.

[HER] Herlihy, M.; Shavit, N.: The Art of Multiprocessor Programming. Morgan Kaufmann 2008.

[INT] Intel IA 32-64 Architecture-Manual Vol. 3, http://www.intel.com/content/dam/www/public/us/en/documents/manuals/64-ia-32-architectures-software-developer-system-programming-manual-325384.pdf

[MTA] MTAPI-Spezifikation, http://www.Multi-Core-association.org/workgroup/mtapi.php

[QUA] Quade, J.; Kunst, E.: Linux-Treiber entwickeln. dpunkt 2011.

[QUM] Quade, J.; Mächtel, M.: Moderne Realzeitsysteme kompakt, dpunkt 2012.

[VXW] VxWorks 6.9 Kernel-Programmers Guide, Wind-River Systems Inc. 2013.

Parallel Symbolic Relationship-Counting

Andreas C. Doering
IBM Research Laboratory
8803 Rüschlikon, Switzerland
ado@zurich.ibm.com

Abstract:
One of the most basic operations, transforming a relationship into a function that gives the number of fulfilling elements, does not seem to be widely investigated. In this article a new algorithm for this problem is proposed. This algorithm can be implemented using Binary Decision Diagrams. The algorithm transforms a relation given as symbolic expression into a symbolic function, which can be further used, e.g. for finding maxima. The performance of an implementation based on JINC is given for a scalable example problem.

1 Introduction

Next to sets, binary relations are one of the most basic mathematical objects. Given two sets A and B, a relation $\sim$ is defined by a subset $R_\sim$ of $A \times B$, such that

$$a \sim b, a \in A, b \in B \Leftrightarrow (a, b) \in R_\sim.$$

Given a relation on finite sets one of the most basic questions is how many elements are related to each element of A. This can be represented as a function

$$C_\sim : A \to \mathbb{N} : a \mapsto |\{b : a \sim b\}|,$$

called 'counting function' for the relationship in the following. When viewing the relationship as the edges of a directed graph, the function gives the out-degree of each node. When the relationship defines a function from B to A, the transformation counts the pre-images for each function image.

Of course, there is also a counting function with respect to elements of B. Without loss of generality, in this paper only the first version is studied.

As a simple example consider the relation "less than" $a < b$ on numbers $0 \dots m$. For each given a there are $m - a$ numbers greater than a from the given interval. Formally, $C_<(a) = m - a$. In this paper the more general relation $(a\%p) > (b\%q)$ is used as benchmark, which allows scaling the complexity of the function and the number of bits for $A = \{0 \dots 2^N - 1\}$ and $B = \{0 \dots 2^M - 1\}$. % stands for the modulo operation. For this relation

$$C(a) = max(0, 2^M/q * (q - a\%p)) + max(0, 2^M\%q - (a\%p)).$$

The proposed algorithm can be implemented using Binary Decision Diagrams [MT98]. I am not aware whether there are other representations of Boolean (or more generally finite) functions which allow the same set of operations, including testing for equality, indexing, composition; if there are then the proposed algorithm can be implemented using those representations as well.

The computer algebra system RELVIEW [BN05] is implemented on the basis of Binary Decision Diagrams, but it does not contain the presented function. JINC [OB08] is a BDD library that supports multithreading on a shared-memory system. It was used for the experimental implementation for this paper.

Algorithms to determine the number of fulfilling inputs for a given function are called SAT-COUNT [Thu12]. This is a more restricted problem as it yields a constant for a given function. This problem is equivalent with relationship counting when the set A has only a single element. SAT-COUNT can handle functions in a more general representation, the creation of a BDD for a given input function can be as complex as SAT-COUNT.

BDDs are used for circuit optimization, circuit verification, state space exploration and similar tasks. The proposed algorithm allows more sophisticated tests, and analysis in these domains. The parallel nature of the algorithm gives hope that it can be used also at a large scale, where the representation of A and B requires 100 or more bits.

In the next section two algorithms for determining the counting function are presented.The first version, a recursive algorithm, is simpler to implement and has weaker requirements to the underlying function representation. The second algorithm is based on Binary Decision Diagrams for the basic data type and exploits the representation structure.

In Section 3 some applications for the algorithm are proposed. One of them is the analysis of network topologies [Dör10]. In Section 4 performance results for a C++ implementation based on the BDD library JINC are given. In the outlook further plans for the refinement of the algorithm are given.

2 Relationship-Counting

One basic idea of the algorithm is that counting the number of elements in a subset can be done by summing the characteristic function over all elements of the underlying universe. Consider for example the set of prime numbers, and assume we have given a function $\texttt{isprime} : \mathbb{N} \to \{0,1\}$, then we can determine the number of primes up to a limit l by $\sum_{i=1}^{l} \texttt{isprime}(i)$. Therefore, given a relation $\sim$ by its characteristic function $r : A \times B \to \{0,1\}, r(a,b) = 1 \Leftrightarrow a \sim b$, determining the counting function can be done by summing over all elements of B:

$C_\sim(a) = |\{b : a \sim b\}| = \sum_{b \in B} r(a,b)$.

The summing operator can be considered a function of $|B|$ inputs:

$S : \{0,1\}^{|B|}, S(x_0, \ldots, x_{|B|-1}) = \sum_i (x_i)$.

Hence $C_\sim(a) = S(r(a,b_0), \ldots, r(a,b_{|B|-1}))$.

Note, that here functions are added, not numbers, i.e. $(f + g) : a \mapsto f(a) + g(a)$. To do this symbolically, a function representation is needed that allows concatenation of two functions, applying partial fixed values to inputs (i.e. concatenation with constant functions) and forming of elementary functions such as addition. The summing function can thus be built from addition functions by concatenation.

For the remainder of the paper it is assumed that natural numbers, including intermediate sums and the result of the counting function are represented as bit vectors with binary encoding ($\sum_i d_i 2^i$). For the representation of elements from A and B, n_a and n_b respectively bits are used. The relation $\sim$ can therefore be represented as a function $\{0,1\}^{n_a+n_b} \to \{0,1\}$. If not all the codes are used to represent elements of A or B, it is assumed that the representation of $\sim$ will always yield 0 when applied to unused codes. Any given representation can be easily modified to fulfil this condition by AND-ing the characteristic functions for the sets A and B to the relation. It is a particular advantage of BDDs that they can handle sparsely represented sets well. For instance permutations, routing structures, and similar objects can be represented with comparably long bit vectors and one-hot encoding with good performance.

The first algorithm is presented for illustrative purposes. It recursively branches on the individual bits `bi` of B. The recursion stops when a function is found that does not depend on b anymore, including constant functions. All the functions found at the bottom of the recursion are collected in a list. Such a recursion on the BDD-representation, limited to certain layers is typical for many BDD algorithms. In particular, this recursion is a part from the fulfilment set counting algorithm found in [MT98]. In a second step an addition tree is built whose leaves are these collected functions. In the simplest form of this algorithm, all functions are added with weight 1, hence the addition tree corresponds to a population count function.

```
collect(f,i)
{
if (f does not depend on bi)
  return {f}
else
  return append(collect(compose(bi,0,f)),
                collect(compose(bi,1,f)))
}
```

`compose(v,e,f)` replaces the input variable v of function f by the expression e. In order to test whether f depends on b or not for a BDD representation the variable order can be used. By arranging the variables for a at the bottom levels and the b-variables at the top, testing the level of the root node of the function f is sufficient and requires only constant time. Other representations, such as polynomials or conjunctive forms might only have a semi-test. This would also be sufficient but could increase the run-time of the algorithm. One improvement of the recursive algorithm is to test, whether the two compositions (compose(bi,0,f) and compose(bi,1,f)) for the recursion parameter represent the same function. In that case the recursion needs to be calculated only once, and the result is returned with weight two, i.e. every list element is extended by a weight, a natural number. This requires of course an equality test on functions, and, again, a semi-test, that can confirm equality but not exclude it, could be applied as well.

The summation step calculates $\sum w_i f_i$ as multi-bit BDD-function, with weights w_i and Boolean functions f_i. The summing is done by constructing a tree from multi-bit adders, where the width of adders grows as necessary from level to level upwards. The algorithm that constructs the adder accounts the sum of weights in the subtrees and can thus determine the required result width. To incorporate the weight w_i for function f_i into the sum on the leaves of the adder tree, a vector of either the constant-zero function or the function itself is formed. If for instance the weight is eleven, the formed vector would be $(f_i, 0, f_i, f_i)$. This is possible because the function is either 0 or 1, so multiplication is identical with the expression:

```
if (fi) then
  return wi
else
 return 0
```

This optimization already contains the idea of the second algorithm. The idea exploits the observation that during the calculation of bitwise projections, frequently the same sub-functions are reached. By collecting the information and refining the function level by level, each sub-function needs to be handled only once. When using BDDs this approach is already known with the only difference being, that BDDs typically do the weight accumulation down to the leaves and do not stop at an intermediate level as is needed here. BDDs can be viewed as representing a dynamic program and the weight calculation corresponds to the well-known dynamic programming algorithm.

The second algorithm consists of two parts, the collection part and the summing part. Both are similar to the recursive algorithm. In the collection part the variable order of the BDD is modified such that variables for b are at the top and the variables for a at the bottom, if needed. Then, the BDD-graph representing the relation is traversed starting from the root level by level and the weights per node are computed. In some BDD libraries the weight can be stored in the graph nodes itself, otherwise a hash table can be used. The root node is assigned weight one. A child node's weight is incremented by weight of the parent multiplied with the power of two of the number of skipped levels, i.e. by the difference of the variable levels of parent and child minus 1. This is because each skipped level corresponds to a input variable that is not relevant for the given situation, and for that reason, every partial assignment for the inputs corresponding to b results in two assignments including the next level variable, one where the variable is one, and another where it is zero. In all well-known BDD implementations this update has constant effort. The terms (the BDD nodes reached which do not depend on b variables) can also be stored in a hash table.

The summing part is identical to the recursive algorithm.

This second algorithm requires the ability to index functions to store them for instance in a hash table or tree.

Both parts of the algorithm exhibit parallelism. Computing a sum of many terms with complex addition steps can naturally be done in parallel by forming an addition tree. Also, the additions itself can be done partially in parallel, for instance by using carry-inputs per digit and composing the digits afterwards or by calculating the prefix sum for the carries in parallel with known methods.

The algorithm can be improved in several ways. One option relates to the addition of weighted vectors on the lowest level of the addition tree. Assuming we add two vectors for functions f and g with weights 5 and 13. This means that we add the vectors $(0, \dots, 0, f, 0, f)$ and $(0, \dots 0, g, g, 0, g)$, forming

$$(0, \dots, 0, g \text{ xor } f\&g, f \text{ xor } g, f\&g, f \text{ xor } g).$$

As can be seen, two terms, $f\&g$ and f xor g can be reused. The caching and reuse of previously computed expressions is part of most BDD implementations. JINC uses a per-thread computed table, so, if the sum of the vector is computed in one thread, the reuse might happen automatically in the BDD functions. Doing it explicitly guarantees the reuse, in case the size of the computed table is not sufficient.

Another idea for improvement is using carry-save adders for the summation tree, as a hardware implementation would do. BDDs are canonical and hence the resulting BDD is determined only by the given relation and not by the algorithm computing it. However, by using carry-save addition one can expect that the intermediate terms will be smaller.

When building the addition tree it is also not clear which sequence is better, first building the weighted addition tree with abstract variables as inputs and then composing the tree with collected functions or building the addition tree directly with the collected terms. Because JINC does not contain a function for vector compose, only the second variant was implemented.

Since addition is commutative and associative, the addition tree can be structured arbitrarily. In order to reduce the complexity of the intermediate operations it might be advantageous to compute more pairwise sums than needed, testing their size and choosing a set of intermediate sums with the smallest size.

3 Applications

Since the described algorithm handles a basic problem, the range of applications is very wide. One application described in [Dör10] is the counting of paths in a network, where routing is constrained by methods for deadlock avoidance or the existence of faults. In this case, one side of the relation covers the start and end points of the path while the other side represents the path of the network as a vector of nodes. The relation is defined by the properties of the network (adjacent nodes in the path have to be connected), and the requirement that the first node of the path is the start and the last node is equivalent to the end point. Further restrictions such as avoiding certain turns or faulty nodes can be added, thereby refining the relation. The proposed algorithm allows finding the number of minimal paths under these restrictions as a function of the start-end node pair. Since the result is a BDD-represented symbolic function, further transformations, such as finding extrema or comparing to the path-counting function for the fault-free network can be carried out in the same framework.

A second type of applications is the evaluation of approximation algorithms, for instance for scheduling. In this case two relations are built, one that describes the set of solutions, and the second representing the set of solutions that a given algorithm can find. BDDs are traditionally used for state space exploration of digital circuits, the set of solutions of an algorithm is a typical outcome of such a process. I am currently working on investigating crossbar scheduling algorithms, such as iSLIP, with this method. It is yet to be seen what size of problem can be covered on typical machines.

A third class of problems deals with configurable circuits. Figures of interest in this context are the set of functions that can be implemented with a given configurable circuit or a minimal configurable circuit that covers a given set of target functions. As an example, consider a configurable random number generator. It consists of a combinational circuit that implements a function with two sets of inputs, one for a uniform random number source and one for the configuration bits. Several classes of circuits can be considered individually, for instance all circuits consisting of three levels of NAND gates. A circuit in such a class is defined by the wiring between the gates which can be represented as a BDD. Doing so results in a relation that combines the wiring, the inputs and the result. With the proposed algorithm the frequency for each output when the random input is varied can be computed as a BDD function. Further processing, including sorting of result vectors, is needed. First experiments have shown that this method works quite efficiently for reasonable circuit sizes.

4 Results

The second algorithm was implemented using JINC on a 64-bit Linux system based on Intel core i5 processors (two cores, two threads each). The tests were repeated on a Freescale T4240-based system called RDB which provides 24 processor cores, and the results were comparable. Table 1 lists the run time of the two parts of the algorithm for several example problems.

As can be seen, scalability works well, and is better for larger problems. Note, that JINC needs internal locking for shared data structures, in particular the so called “unique” table, with one table per variable. Scaling of part 1 is limited by the global lock for the hash table that collects the individual functions. This can be improved by using a hash-map implementation that allows concurrent insertion, as is provided by Intel’s Threading Building Blocks library. However, the thread management of the TBB is somewhat different than that of the boost library. Since a concurrent hash table is in preparation for the boost library, this improvement was postponed.

Table 1: Results for some example runs, note that 1536=3*512 resulting in a particular simple BDD for the modulo operation. p1 and p2 are the runtimes of the two parts of the second algorithm in seconds.

N	M	p	q	threads	p1	p2	terms
16	16	1697	1879	1	0.05	33	2047
16	16	1536	1879	1	0.04	1.75	2047
16	16	1697	1536	1	0.03	7.3	2047
16	22	10697	1035641	1	0.96	42.5	16383
22	16	1035641	10697	1	0.15	42.5	16384
16	16	1697	1879	2	0.07	18	2047
16	16	1536	1879	2	0.1	1.0	2047
16	16	1697	1536	2	0.06	11.2	2047
16	22	10697	1035641	2	2.08	29.7	16383
22	16	1035641	10697	2	0.3	37.7	16384
16	16	1697	1879	3	0.09	15.7	2047
16	16	1536	1879	3	0.18	0.93	2047
16	16	1697	1536	3	0.08	11.5	2047
16	22	10697	1035641	3	2.5	26.8	16383
22	16	1035641	10697	3	0.4	30.3	16384

5 Outlook

It has to be noted that the chosen example has some particular properties, as can be seen from the number of terms. Another property that was observed is that there are few edges that skip one or several levels. This is a property of the BDD representing the input relation. These level-skipping edges however contribute to the acceleration of the algorithm. More level-skipping edges improve the parallel performance of the first step because the locking for weight updates are better distributed; the current implementation uses one lock per level.

Therefore, one of the most important next steps is the use of other examples, in the best case by applying the algorithm to applications or by integration into a more general tool, such as RELVIEW.

Furthermore, using a concurrent hash table once it is available in the boost library should improve the scalability of the first step of the algorithm.

References

[BN05] Rudolf Berghammer and Frank Neumann. RelView - An OBDD-Based Computer Algebra System for Relations. In Victor G. Ganzha, Ernst W. Mayr, and Evgenii V. Vorozhtsov, editors, *CASC*, volume 3718 of *Lecture Notes in Computer Science*, pages 40–51. Springer, 2005.

[Dör10] Andreas C. Döring. Analysis of Network Topologies and Fault-Tolerant Routing Algorithms using Binary Decision Diagrams. *Parallel and Distributed Processing Workshops and PhD Forum, 2011 IEEE International Symposium on*, 0:1–5, 2010.

[MT98] Christoph Meinel and Thorsten Theobald. *Algorithms and Data Structures in VLSI Design*. Springer-Verlag New York, Inc., 1998.

[OB08] Jörn Ossowski and Christel Baier. A uniform framework for weighted decision diagrams and its implementation. *Int. J. Softw. Tools Technol. Transf.*, 10:425–441, September 2008.

[Thu12] Marc Thurley. An Approximation Algorithm for #k-SAT. In Christoph Dürr and Thomas Wilke, editors, *STACS*, volume 14 of *LIPIcs*, pages 78–87. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2012.

Parallel Processing for Data Deduplication

Peter Sobe, Denny Pazak, Martin Stiehr
Faculty of Computer Science and Mathematics
Dresden University of Applied Sciences Dresden, Germany
Corresponding Author Email: sobe@htw-dresden.de

Abstract: Data deduplication is a technique for detection and elimination of duplicated data blocks in storage systems. It creates a set of unique data blocks and places references accordingly, which allows to access the original data within a reduced amount of data blocks. For deduplication, hashes of data blocks are calculated and compared in order to detect and remove duplicates. It can be seen as an alternative to data compression that allows to save storage capacity in large storage systems. A storage capacity saving is reached at the cost of additional computational effort that originates when data blocks are written and updated. This computational effort increases with the size of the storage system. On a single processor system, deduplication influences the performance in a negative way, particularly the write and update rates drop. The utilization of parallelism is a rewarding task to compensate this performance drop, particularly for hash value calculations and comparisons of hashes. In this paper we explain in which parts of a deduplication system it is worth to parallelize and how. Exemplarily, we show the performance results of two deduplication algorithms and their parallel implementations, based on multithreading and on parallel GPU computations.

1 Introduction

In this paper, parallelism concepts for data deduplication algorithms are presented. Deduplication first appeared in the field of backup systems and repositories for source code and technical documents (such as git) that contain many versions of files with little alterations. Meanwhile, general purpose file systems and database systems got equipped with deduplication. Examples for the use of deduplication are ZFS, lessfs [les15], opendedup [ope15] and OracleDB for storage of large data objects in databases.

Deduplication improves the storage utilization and is capable to reduce the storage operation cost for large systems. This benefit comes with extra computational effort for write and update operations. Deduplication gets more effective with increasing storage size. Unfortunately, the computational cost increases as well with the amount of stored data. This computational cost indirectly reduces the storage access performance, but can be compensated by parallel computing, particularly by using multiple cores or by utilizing the GPU of a computer system.

Recently, research started to evaluate several possibilities of parallel computations for deduplication, such as [XJF+12] for exploiting thread-level parallelism in multi core systems to reach acceptable high I/O throughput for data deduplication. Besides of parallelism for the deduplication task, distributed storage systems are in the focus of research as well.

In [KMBE12] a cluster solution using data deduplication is presented.

The contribution of this paper is the description of two algorithms for deduplication. These algorithms are called (i) Backward Referencing and (ii) Hashmap-based Referencing. These two algorithms reflect state-of-the-art principles that were found in existing deduplication solutions. The description is followed by the identification of algorithm phases for parallel execution and performance reports about two different approaches for parallel processing. One approach is thread-level parallelism utilized by task-parallel programming with the Intel TBB class library. Another approach consists in offloading of the search for duplicates to the GPU. The GPU is capable to perform a huge number of comparisons in parallel.

The remainder of the paper is organized as follows. The principle of data deduplication and two algorithmic approaches are described in Section 2. In Section 3 the algorithms are revisited with respect to parallelism. Last, in Section 4 we present the performance gain of parallel execution for the two algorithms. A summary concludes the paper.

2 Data Deduplication

2.1 Original Data Layout

For the description of deduplication we start with a small example of 8 data blocks that contain repetitions. The sequence of blocks is shown in Fig. 1 where equal letters represent equal block content. These can be equal files, duplicates of emails or equal blocks in different versions of files.

sequence of blocks

A	B	B	C	D	A	D	C

Figure 1: Example of a sequence of blocks taken as original data layout.

The objective of deduplication is to remove double blocks from the storage and to represent their existence by references to unique blocks.

In the following two different deduplication algorithms with different data structures are compared. One algorithm we call Backward Referencing (see 2.2) and the other Hashmap-based Referencing (see 2.3)

2.2 Backward Referencing

Fig. 2 illustrates one possible deduplicated data layout. Data blocks are indirectly referenced via an index that contains pointers to data block descriptors. A data descriptor holds a reference to the data block X and in addition the hash value h_X and a reference counter.

To store the hash value is optional, but beneficial for the integration of new blocks.

In file systems such index structures to blocks are already present. Thus, the integration of deduplication does not cause noticeable extra access cost for read operations.

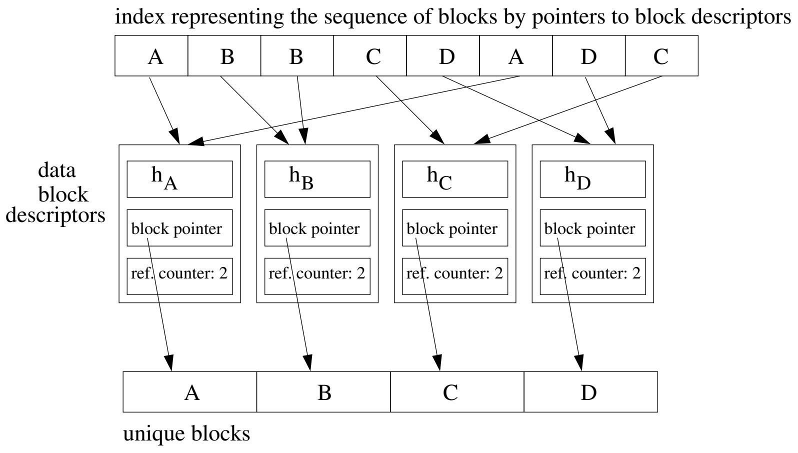


Figure 2: Deduplicated data layout.

The transformation from the original to the deduplicated data layout is done stepwise as described below. For each step we give the time complexity of the sequential algorithm step, depending on the number of blocks in the system n.

1. **Chunking and Hashing:** Original data is divided into blocks (a so called chunking). For each block a hash value is calculated. In order to keep the probability of hash collisions considerably low, strong cryptographic hash functions are used, such as MD5. At the end of this step, a sequence of block references and corresponding hash values is obtained. The computational complexity of this step is $O(n)$.

2. **Indexing:** In a first walk through the sequence of hash values, multiple occurrences of hash values are tagged. For every hash value at position i, all hashes from position $i + 1$ to the end of the sequence are compared. In case of equal hash values, a backward reference is added. At the end of this step, the sequence of hash values got extended by duplicate identification tags and backward references. This step involves $n(n - 1)/2$ comparisons for n blocks, and is characterized by $O(n^2)$.

3. **Re-indexing:** To obtain structures for fast access to data blocks, the references have to reflect the concentration of data blocks to a sequence of unique blocks. Thus the backward references have to be corrected to the block numbers in the sequence of unique blocks. For all blocks references, all other references that possibly point back to it have to be corrected. At the end of this step, a sequence of references to unique blocks is obtained. Equally to the the Indexing step, $n(n - 1)/2$ references must be visited and a complexity of $O(n^2)$ is present. This last step can include a

reorganisation of data blocks to a set of unique blocks and references that correctly address unique data blocks.

This last step can include a reorganisation of data blocks to a set of unique blocks and references that correctly address unique data blocks.

Updates within a deduplication storage system work similar to block write operations. An altered block is handled like a completely newly written one. In case that the new block content is a new unique block, this block is stored and referenced via a new block descriptor. Otherwise a reference to another already existing block is included and the reference to the old block descriptor is removed. With the help of the reference counter, the decision can be taken, whether a block can be finally deleted or not.

Reading blocks from a deduplicated storage layout is done by accessing the index, following the link to a data block descriptor and finally fetching the entire data block. There is no extra computational overhead apart from referencing blocks via an index which is already present in most file systems.

2.3 Hashmap-based Referencing

Another algorithm for deduplication appeared that uses a hashmap data structure for the detection of duplicated blocks. The first phase of hashing is similar to backward referencing and consumes a compute time according $O(n)$. A difference compared to the backward referencing algorithm is that a sequence of hash values is kept as initial reference to the the blocks, and to represent the order of original blocks.

In a second step, tuples consisting of a reference counter and a block pointer are inserted into a hashmap data structure that places entries with the key h_X according to a hash function h(h_X) at a defined place. The hash values from the first phase are taken as keys for the tuples.

In the case that the entry h_X is a new one, it gets newly placed in the hashmap. The related data block X is moved to the set of unique blocks and referenced from the hashmap entry. The reference counter is set to 1. In case that the entry for h_X already exists, solely the reference counter is increased. The corresponding data block is a duplicate and can be discarded. At the end of step 2, a data layout as depicted in Fig. 3 is obtained. The time of hashmap insertion for n blocks corresponds to the complexity order of $O(n)$, because of $O(1)$ for a single hashmap access operation.

Updates of blocks first cause a lookup of the old hashmap entry and its removal when the reference counter is 1, otherwise the reference counter is decreased. For the new content, an according hash value is inserted in the sequence of hash values. The rest is done according to a write operation of the block.

The read operation first reads the hash from the sequence of hashes and then accesses the corresponding data block through the hash map.

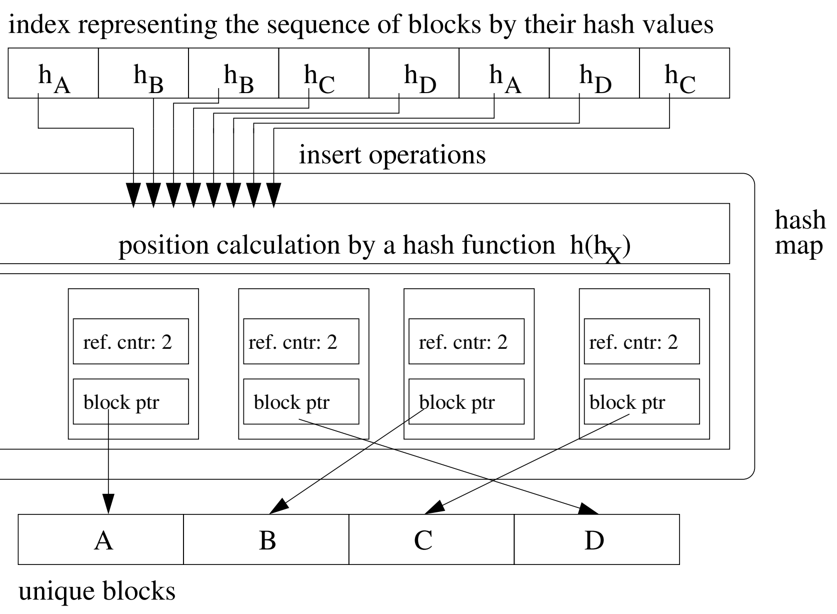


Figure 3: Hashmap-based Referencing: Deduplicated data layout.

3 Parallelism Concepts

For both algorithms, the hashing phase can be executed independently on different blocks. With p processor cores, a single core at most has to calculate $\lceil n/p \rceil$ hash values. This reduces the time consumption of this phase by the factor $1/p$.

3.1 Parallelism of the Backward Referencing algorithm

The phases Indexing and Re-indexing offer room for parallel execution. Every phase requires that for a sequence of entries, all entries must be selected sequentially, and all entries positioned right of the selected one must be visited for hash comparison or for a correction of the reference pointer. For a data set of n blocks, this requires $n(n-1)/2$ steps. With p processors the number of steps can be obviously reduced to approximately $n(\lceil (n-1)/p \rceil)/2$ because of the independence of the operations that relate to the same selected entry. Ideally, a speedup of p can be reached for a large n.

3.2 Parallelism of the Hashmap-based Indexing algorithm

The hashmap-based algorithm works differently and delegates the duplicate detection to the insert operation of the hashmap data structure. The insert operation signals a second colliding entry with the same key. From this perspective, parallel tasks can only be utilized in the way of concurrent insert operations.

It strongly depends on the implementation of the hashmap access operations whether parallel inserts are possible at all. Insert operations are allowed to run parallel in the case that the insert positions do not collide. In the case of a collision, typically locking is applied. In the case that the first write operation to a specific memory place wins, locking can be mapped to the placement of an entry itself.

When the hashmap supports concurrent inserts, the workload can be distributed to p processors. A number of n insert operations can be executed in $\lceil n/p \rceil$ steps and a speedup of approximately p can be reached.

3.3 Comparison

A comparison of the estimated number of steps of both algorithms is shown in Table 1.

	Backward Referencing	Hashmap-based Referencing
Hashing	n	n
Indexing	$n(n-1)/2$	-
Re-indexing	$n(n-1)/2$	-
Hashmap insert	-	n
total steps / sequential	$n + 2(n(n-1)/2)$	$2n$
total steps / parallel	$\lceil \frac{n}{p} \rceil + 2(n(\lceil (n-1)/p \rceil)/2)$	$2\lceil \frac{n}{p} \rceil$
memory	low memory requirements	highly memory intensive
our implementation	@GPU, CUDA	@CPU, Intel TBB

Table 1: Comparison of the algorithms regarding the number of computation steps and memory consumption.

The comparison shows that both algorithms are well suited for parallel execution and ideally utilize the processors or cores that are available. Another insight is that the Hashmap-based Referencing algorithm generally is less costly in terms of computation steps. This result is based on the assumption that the hashmap allows insert operations in $O(1)$ which is true for sparely filled hashmaps and a large memory that can be used potentially. The benefit in terms of computation cost is combined with a higher memory consumption of the hashmap.

4 Performance Evaluation

In this section, we report first on the parallel implementation of the Backward Referencing algorithm on a GPU using CUDA (4.1) and second on the parallel implementation of the Hashmap-based Referencing algorithm using multithreading and Intel TBB data structures (4.2).

4.1 Backward Referencing using GPU computing

The CUDA version of the backward referencing algorithm searches a specific hash value on different positions in parallel, utilizing a huge number of GPU threads. When a specific hash value has to be compared with a number of x other hash values, the one hash is transferred in the constant memory of the GPU, and the other x hash values for comparison are copied to the GPU global memory. Values are compared by parallel threads. Identical values are found faster compared to a sequential iterative execution. The GPU kernel thread execute according actions in parallel (tagging of duplicates, setting or correction of backward references). The implementation revealed performance results as depicted in Fig. 4. The bars show the relative speedup compared to a sequential CPU implementation on an AMD Phenom II, X4, 840, 3.2 GHz. The GPU used is a consumer model (Type NVidia Quadro 600) with 96 CUDA cores and 1.28 GHz clock rate. The reached speedup does not fully utilize the parallelism of the GPU, due to the interplay of the sequential CPU execution and the parallel GPU execution.

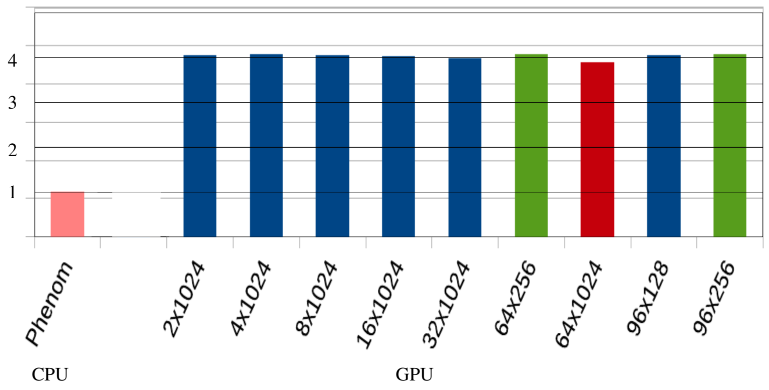


Figure 4: Deduplication throughput using a CPU and a GPU implementation using CUDA. The numbers (e.g. 2x1024) identify the number of blocks and treads for the CUDA kernel invocation.

4.2 Multithreaded Hashmap-based Referencing using Intel Task Building Blocks

A parallel implementation of the Hashmap-based Referencing algorithm was created using C++, a fast MD5-hash implementation[Thi91] and the Intel TBB library (TBB: task building blocks) [Sof15]. This task-parallelism library does not only provide mechanisms to manage several tasks on a multithreaded system, it also provides data structures that can be accessed by parallel tasks. Specifically, the TBB concurrent hashmap [Gue12] was used. This data structure implementation fulfills the above stated requirement of concurrent insert operations that do not block.

The experiments run on an Intel Core i5-750, 2.7 GHz system and a 64-Bit gcc compiler was used. The measurements cover the deduplication of a 1.07 GByte ISO image. The execution time for sequential execution and parallel execution with 2,3, and 4 tasks is shown in Fig. 5.

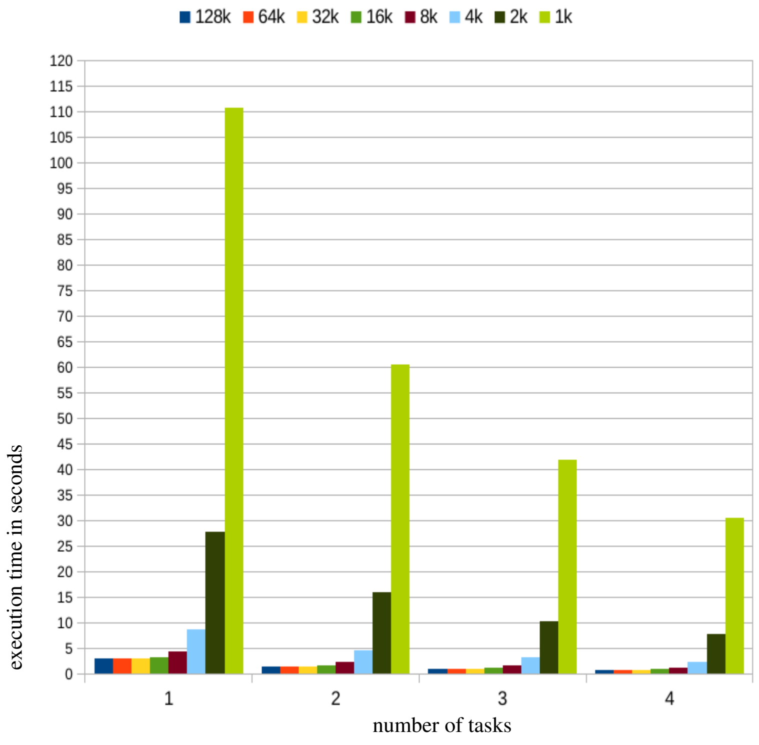


Figure 5: Deduplication execution times, parallel Hashmap-based Referencing algorithm

The plots represent different runs of the deduplication algorithm with specific minimal blocks sizes (128k to 1k). Actually, the implementation covers deduplication with adaptive block lengths and first tries to find equal blocks of a maximum length (128 kByte). For all remaining differing blocks the blocksize for deduplication is divided by 2 until a

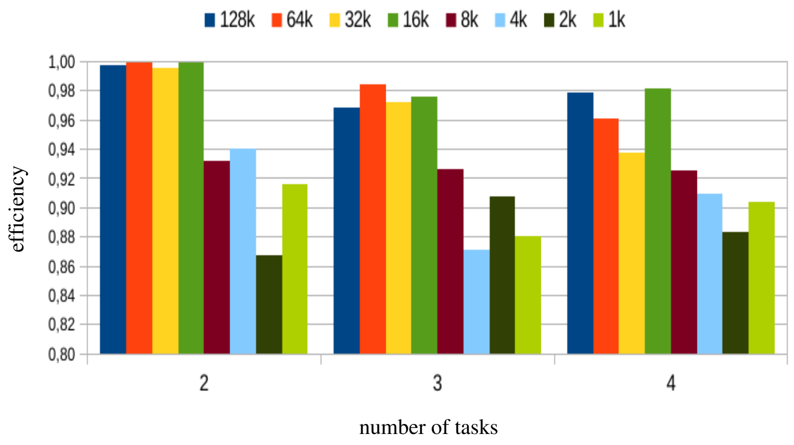


Figure 6: Efficiency of the parallel Hashmap-based Referencing algorithm using Intel TBB

minimum block length is reached. A smaller minimum block size increases the computational cost, because more hash values are generated and have to be processed. These block size variations confirm the general observation of efficient parallel execution.

In general the hashmap-based algorithm turned out as well suited for parallel execution, provided the hashmap allows parallel insert operations. A notable performance could be reached, e.g. roughly the deduplication of 1 GByte could be done in 30 seconds, down to a small block size of 1 kByte. This is a data throughput of 33 MByte/s for a very fine-granular deduplication. The efficiency of the parallel Hashmap-based Referencing algorithm execution is shown in Fig 6.

5 Summary

It could be shown that deduplication is well suited for parallel execution. The two algorithms are different regarding their strategy to compare blocks. Backward referencing works iteratively, but independently on blocks. It compares a number of block pairs, which is the source of parallelism. This can be seen as data parallelism generated from the control flow of a program.
Another approach is based on a hashmap data structure and maps the comparison to a placement problem. Equal entries collide and different entries get placed at different positions. This saves one iteration stage of the sequential algorithm. Even this principle could be modified for parallel execution, where the parallel non-blocking access to the data structure turned out to to be central point for the parallelisation. The Intel task building block library provided the task management and the concurrent hashmap implementation.

References

[Gue12] P. Guermonperez. Parallel Programming Course - Threading Building Blocks (TBB). Intel Software, www.Intel-Software-Academic-Program.com, 2012.

[KMBE12] J. Kaiser, D. Meister, A. Brinkmann, and S. Effert. Design of an Exact Data Deduplication Cluster. In *IEEE 28th Symposium on Mass Storage Systems and Technologies (MSST)*, 2012.

[les15] lessfs. http://www.lessfs.com, accessed 2015, February 9, 2015.

[ope15] opendedup. http://opendedup.org, accessed 2015, February 9, 2015.

[Sof15] Intel Software. Intel Threading Builiding Blocks. www.threadingbuildingblocks.org, accessed: March 20, 2015.

[Thi91] F. Thielo. C++ MD5 Implementation. http://www.zedwood.com/article/cpp-md5-function, accessed: 2015, March 20, 1991.

[XJF$^+$12] W. Xia, H. Jiang, D. Feng, L. Tian, M. Fu, and Z. Wang. P-Dedupe: Exploiting Parallelism in Data Deduplication System. In *IEEE 7th Intl. Conference on Networking, Architecture and Storage*, 2012.

Energy-aware mixed precision iterative refinement for linear systems on GPU-accelerated multi-node HPC clusters

Martin Wlotzka[1], Vincent Heuveline[1]

[1] Heidelberg University, Interdisciplinary Center for Scientific Computing (IWR)
Engineering Mathematics and Computing Lab (EMCL), Speyerer Str. 6, 69115 Heidelberg
martin.wlotzka@uni-heidelberg.de, vincent.heuveline@uni-heidelberg.de

Abstract: Modern high-performance computing systems are often built as a cluster of interconnected compute nodes, where each node is built upon a hybrid hardware stack of multi-core processors and many-core accelerators. To efficiently use such systems, numerical methods must embrace the different levels of parallelism from the coarse-grained distributed memory cluster level to the fine-grained shared memory node level parallelism. Synchronization requirements of numerical methods may diminish parallel performance and result in increased energy consumption. We investigate block-asynchronous iteration methods in combination with mixed precision iterative refinement to address this issue. We depict our implementation for multi-node distributed systems using MPI with a hybrid node level parallelization for multi-core CPUs using OpenMP and multiple CUDA-capable accelerators. Our numerical experiments are based on a linear system arising from the finite element discretization of the Poisson equation. We present energy and runtime measurements for a quad-CPU and dual-GPU test system. We achieve runtime and energy savings of up to 70% for block-asynchronous GPU-accelerated iteration using mixed precision compared to CPU-only computation. We also encounter configurations where the CPU-only computation is advantageous over the GPU-accelerated method.

Keywords: energy-aware numerics, high-performance computing, mixed precision, asynchronous iteration, graphics processing units

1 Introduction

Numerical simulations play a key role for scientific discovery, complementing theoretical analyses and experiments. The computational power of high-performance computing (HPC) systems in terms of peak floating point operations per second (flops) has increased currently above the petaflops level [ww15]. Seeking to further increase the computational power towards the exascale level, the HPC community is facing the power wall. Simply upscaling current technology would result in a prohibitive power demand in the order of several hundred Megawatts for one exascale system. The issue of energy consumption has therefore become a major issue in the HPC field [ww14].
Many HPC systems are built as a cluster of interconnected compute nodes. Each node usually comprises one or more multi-core processors, and may additionally include many-core accelerators. Thus, HPC clusters often represent a hybrid form of distributed memory interconnected nodes with shared memory multi-core CPUs and possibly many-core devices on the node level. Such systems offer different levels of parallelism. In order to

leverage the computational power, numerical methods must exploit the parallelism provided on the different levels. However, synchronization requirements may diminish the parallel performance and result in increased energy consumption. Asynchronous iteration methods allow to circumvent typical synchronization requirements of classical iteration methods. To address the issues of synchronization and energy consumption, we investigate the block-asynchronous variant in combination with mixed precision iterative refinement for the solution of linear systems of equations.
Related work and paper contribution
The idea of ”chaotic relaxation” was proposed by Rosenfeld [Ro69], who used ”parallel processor computing systems” to simulate the distribution of current in an electrical network. Chazan and Miranker in 1969 [CM69] were the first to study this type of methods on a rigorous theoretical basis. They established a characterization of the chaotic relaxation schemes for the solution of symmetric positive definite linear problems and gave conditions for convergence, as well as examples for divergence. Meanwhile, the denomination ”asynchronous iteration” has been established in the literature. An overview of asynchronous schemes and convergence theory can be found in [FS00]. Asynchronous iteration has successfully been used in the context of HPC, see e.g. [EFS05] and references therein.
Earlier works reported in [An11b] and [An13] investigate convergence properties and performance of block-asynchronous iteration on GPU-accelerated systems, both as plain solver and in combination with mixed precision iterative refinement. However, these works are restricted to single node, single host process configurations, and the host CPUs are not taken into account for computations. We extend this setup to the case of distributed memory machines with several host processes running on the same node and sharing devices. Additionally, we compare with asynchronous CPU-only methods which also benefit from relaxed synchronization requirements. Finally, we perform actual energy measurements to investigate the energy consumption of the methods.
Paper organization
This paper is structured in the following way: We outline the mathematical background in Section 2. In Section 3, we describe the setup of our experiments. In particular, we present the main features of our hybrid implementation using MPI [Me12] for distributed systems and OpenMP [Op13] on the shared memory local level, as well as CUDA [NV14a] for graphics processing units (GPU). Section 4 is devoted to the discussion of the results, and Section 5 concludes the work.

2 Mathematical background

In this work, we investigate the performance and energy consumption of asynchronous iteration schemes in the context of mixed precision iterative refinement. In this section, we introduce the mathematical background of the methods we use.

2.1 Mixed precision iterative refinement

The idea of iterative refinement for the solution of a system of linear equations comes from Newton's method for approximating the solution of $f(x) = 0$, where f is a smooth function. Considering the special case of a linear function $f(x) = b - Ax$ with a regular matrix $A \in \mathbb{R}^{n\times n}$ and a vector $b \in \mathbb{R}^n$, solving $f(x) = 0$ is equivalent to solving the linear system $Ax = b$. For an approximate solution x^k, the residual is denoted $r^k = b - Ax^k = f(x^k)$. Using $\nabla f \equiv -A$ leads to the following linear iterative refinement method

$$x^{k+1} = x^k + A^{-1}r^k \qquad (k = 0, 1, ...) .$$

In each iteration, the current approximation x^k is improved by the correction $c^k = A^{-1}r^k$, which is the solution of the error correction equation $Ac^k = r^k$. If an exact correction could be computed, the iterative refinement process would end after one iteration with the correct result. However in practice, often only approximate error corrections $\tilde{c}^k$ can be computed by means of numerical solvers. Let $q^k = r^k - A\tilde{c}^k$ be the residual of the error correction equation. After the correction, the updated solution $x^{k+1} = x^k + \tilde{c}^k$ yields the residual

$$r^{k+1} = b - Ax^{k+1} = b - A(x^k + \tilde{c}^k) = r^k - A\tilde{c}^k = q^k .$$

Thus, the accuracy of the error correction solver determines the accuracy of the solution of the overall iterative refinement process.
Instead of solving the error correction equation in the working precision, one can transfer the system to a lower precision. This approach amounts to the mixed precision iterative refinement (MPIR) [Ba09], see Algorithm 1. For MPIR, we use an absolute stopping criterion based on a given tolerance $\varepsilon > 0$.
For computing the correction in step 6 of Algorithm 1, one may choose any appropriate numerical solver. In this work, we focus on the asynchronous methods explained in the next section.

Algorithm 1 Mixed precision iterative refinement (MPIR)

```
 1: Typecast $A^{\text{low}} \leftarrow A^{\text{high}}$.
 2: Set initial solution $x^{\text{high}}$, tolerance $\varepsilon > 0$.
 3: Compute residual $r^{\text{high}} = b^{\text{high}} - A^{\text{high}}x^{\text{high}}$.
 4: **while** $\|r^{\text{high}}\| > \varepsilon$ **do**
 5:     Typecast $r^{\text{low}} \leftarrow r^{\text{high}}$.
 6:     Solve $A^{\text{low}}c^{\text{low}} = r^{\text{low}}$ approximately.
 7:     Typecast $c^{\text{high}} \leftarrow c^{\text{low}}$.
 8:     Correct $x^{\text{high}} \leftarrow x^{\text{high}} + c^{\text{high}}$.
 9:     Compute residual $r^{\text{high}} = b^{\text{high}} - A^{\text{high}}x^{\text{high}}$.
10: **end while**
```

2.2 Block-asynchronous iteration

The asynchronous iteration methods under investigation in this work can be derived from the classical Jacobi relaxation method [An13]. The Jacobi method relies on an additive splitting of the system matrix $A = L + D + U$ into a lower triangular matrix L, a diagonal matrix D and an upper triangular matrix U. Assuming D to be regular, the Jacobi iteration reads [Me11]

$$\begin{aligned} x^{k+1} &= D^{-1}\left[b - (L+U)x^k\right] \\ &= Bx^k + d \end{aligned} \qquad (k = 0,1,...) \; ,$$

where $B = -D^{-1}(L+U)$ is the iteration matrix and $d = D^{-1}b$. A necessary and sufficient condition for the convergence of the Jacobi iteration is $\rho(B) < 1$, where $\rho(B)$ denotes the spectral radius of B. In terms of the system matrix A, a sufficient condition is strict diagonal dominance of A, or diagonal dominance and irreducibility [Sa00].
The parallelization of this method is straightforward. Each compute unit may compute a part of the new iteration vector x^{k+1}. Note that for computing its part of the new iterate x^{k+1}, any compute unit potentially uses components of the preceding iterate x^k which belong to other compute units. This requires a synchronization of the compute units after each iteration to make sure that all needed values are updated from the last iteration.
The idea of asynchronous iteration is to overcome the synchronization requirements. On the theoretical level, this is accomplished by introducing a shift function s and an update function u in the iteration:

$$x_i^{k+1} = \begin{cases} \frac{1}{a_{ii}}\left[b_i - \sum_{j\neq i} a_{ij}x_j^{k-s(j)}\right] & \text{if } i = u(k) \\ x_i^k & \text{if } i \neq u(k) \end{cases} \; , \; (k = 0,1,...)$$

The shift function allows to use not only values from the last iteration, but also older or newer values. The update function chooses one component at a time to be updated, leaving the other components unchanged [An11a]. A sufficient condition for convergence is uniform boundedness of s, and u must take each value in $\{1,...,n\}$ infinitely often, and $\rho(|B|) < 1$ [CM69].
A natural modification of the basic asynchronous scheme is the aggregation of components into blocks [Ba99]. Let L be the number of blocks, and $I_l \subset \{1,...,n\}$ be the index set of all components belonging to block $l \in \{1,...,L\}$. The block-asynchronous iteration reads

$$i \in I_l \; : \quad x_i^{k+1} = \frac{1}{a_{ii}}\left[b_i - \sum_{j\notin I_l} a_{ij}x_j^{k-s(k,j)} - \sum_{j\in I_l, j\neq i} a_{ij}x_j^k\right] \qquad (k = 0,1,...) \; .$$

This scheme is synchronized only with respect to the vector components within each block. The block scheme implies a decomposition of the systems matrix A into diagonal and off-diagonal parts

$$D_l = \Big(a_{ii}\Big)_{i\in I_l} \; , \quad A_l^{\text{diag}} = \Big(a_{ij}\Big)_{i,j\in I_l, i\neq j} \; , \quad A_l^{\text{offdiag}} = \Big(a_{ij}\Big)_{i\in I_l, j\notin I_l}$$

and a decomposition of the vectors x and b into local parts

$$x_l^{\text{local}} = \Big(x_i\Big)_{i\in I_l} \; , \quad b_l^{\text{local}} = \Big(b_i\Big)_{i\in I_l} \; , \quad x_l^{\text{non-local}} = \Big(x_j\Big)_{j\notin I_l} \; .$$

Such block decomposition is sketched for A and x in Figure 1. The update step for any block l then reads

$$x_l^{\text{local}} \leftarrow D_l^{-1}\left[b_l^{\text{local}} - A_l^{\text{diag}}x_l^{\text{local}} - A_l^{\text{offdiag}}x_l^{\text{non-local}}\right] \; .$$

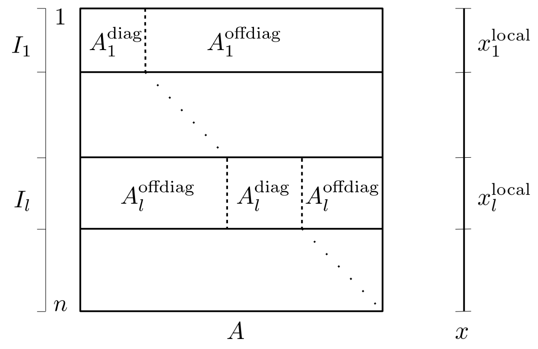


Fig. 1: Decomposition of the system matrix A and solution vector x into blocks.

Note that the actual block sizes in the decomposition depend on the number of compute units, and on the load distribution among them. For balanced load distributions, the local block sizes decrease with increasing number of compute units, while the non-local parts grow.

The block-asynchronous scheme can be extended by performing multiple iterations on the local block before updating the values in the non-local vector part. We denote the resulting algorithm as async-(m) to indicate m local steps between non-local updates. Obviously, each block can be mapped to one compute unit, resulting in Algorithm 2. We use async-(m) with a relative stopping criterion based on a given tolerance $\delta > 0$.

Algorithm 2 async-(m)

```
1:  Set initial solution $x$, tolerance $\delta > 0$.
2:  Compute initial residual $r^0 = r = b - Ax$.
3:  while $\|r\| > \delta\|r^0\|$ do
4:      for all blocks $l = 1, ..., L$ in parallel do
5:          for k=1,...,m do
6:              $x_l^{\text{local}} \leftarrow D_l^{-1}\left[b_l^{\text{local}} - A_l^{\text{diag}} x_l^{\text{local}} - A_l^{\text{offdiag}} x_l^{\text{non-local}}\right]$
7:          end for
8:      end for
9:      Update $x_l^{\text{non-local}}$ with corresponding values from other blocks.
10:     Compute residual $r = b - Ax$.
11: end while
```

3 Experimental setup

3.1 Linear problem

In our experiments, we use the linear system of equations arising from a finite element discretization of the two-dimensional Poisson equation [EG04]. This equation can be used to model the equilibrium heat distribution in a physical domain with given environmental temperature and heat sources or sinks. The problem definition reads

$$\begin{aligned} -\Delta u &= f && \text{in } \Omega\,, \\ u &= g && \text{on } \partial\Omega_\mathrm{D}\,, \\ \nabla u \cdot n &= 0 && \text{on } \partial\Omega_\mathrm{N}\,, \end{aligned}$$

where $\Omega \in \mathbb{R}$ is the physical domain, f represents any heat sources or sinks and g is the environmental temperature given through the Dirichlet condition on the boundary part $\partial\Omega_\mathrm{D}$. Thermal insulation is modeled by the homogeneous Neumann boundary condition on the boundary part $\partial\Omega_\mathrm{N}$. For our experiments, we chose the domain Ω to be the unit square. Our finite element discretization with 262,144 mesh cells results in $n = 263{,}169$ unknowns and 2,362,369 non-zero elements for the system matrix A.

3.2 Implementation for GPU-accelerated multi-core shared and distributed memory HPC clusters

Our implementation spans three levels of parallelism. It supports multi-node distributed memory systems where the nodes are connected by a network. Communication between the nodes is done by data transfer over the network using MPI [Me12]. On the node level, it supports both multi-core shared memory systems by means of OpenMP [Op13] as well as CUDA-capable devices [NV14a].
The implementation is integrated in the HiFlow^3 package [An12]. It uses the MPI-parallelized matrix and vector data structures for input and for the MPI communication between nodes. The matrix and the vectors are distributed among the MPI processes, thus defining the block decomposition. The communication pattern is derived from the matrix structure and avoids any unnecessary data transfer. Only vector components corresponding to non-zero entries in the off-diagonal matrix parts of other MPI processes are transferred.
In Algorithm 2, the parallelism of the local block updates corresponding to steps 4-8 is achieved by concurrency of the MPI processes. All computations of the error correction solver are either executed on the host CPUs, or on the accelerator devices. Again, the CPU implementation is parallelized with OpenMP on the node level, while the accelerator version is implemented with CUDA. The update step 9 implies MPI communication and, if the CUDA version is used, data transfer between host and devices.

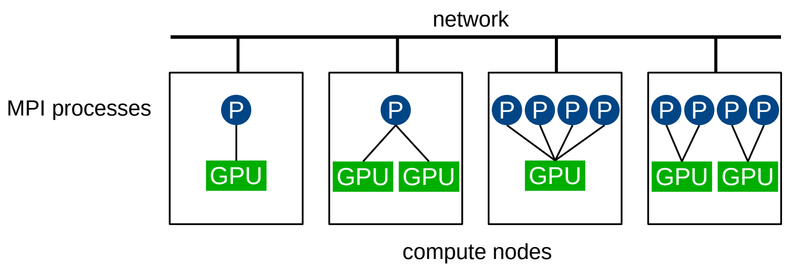


Fig. 2: Supported configurations of MPI process scheduling among compute nodes and GPU usage.

One or multiple MPI processes may be scheduled onto each node. Within each node, the MPI processes can use multiple GPUs. The actual utilization may be configured depending on the number of MPI processes and on the number of available devices. If only one MPI process is scheduled onto a node, this process may use all available devices on that node, as sketched in the two left configurations in Figure 2. In case of multi-GPU usage of a single MPI process, the matrix and vector blocks of this process are further split into sub-blocks as depicted in Figure 3. However, if multiple MPI processes are scheduled onto the same node, GPU utilization must be split such that each process uses only one of the available devices, see the two right configurations in Figure 2. This limitation is imposed by a constraint of the GPU architecture, which we briefly explain in the following.
Each host process establishes its own CUDA context, but there can only be one CUDA context active at a time on the device. If several host processes access the GPU, a time-sliced scheduler switches between contexts to serve them, which implies a serialization. To efficiently use the same device by several host processes, the multi-process service (MPS) [NV14b] can be used. With MPS, host processes connect to the MPS server instead of directly accessing the device. The MPS server maps the different host CUDA contexts into one context on the GPU. This avoids the context switching and enables to benefit from the Hyper-Q feature of devices based on the Kepler architecture [NV12]. With Hyper-Q, up to 32 independent CUDA streams may be executed concurrently on the GPU. The drawback of MPS is that the MPS server can only manage one device such that any process can only use one GPU. If multiple devices are available on the node, one MPS server instance is needed for each device, and host processes need to connect to exactly one of them. A practical way to meet these technical requirements can be found in [WSC14]. For Algorithm 1, all steps except the solution of the error correction equation in step 6 are implemented in C++ for execution on CPUs. In addition to the MPI parallelization for distributed systems, all local computations are parallelized with OpenMP to exploit multi-core shared memory nodes. The error correction solver itself uses Algorithm 2 and can be executed either on CPUs or on accelerators.

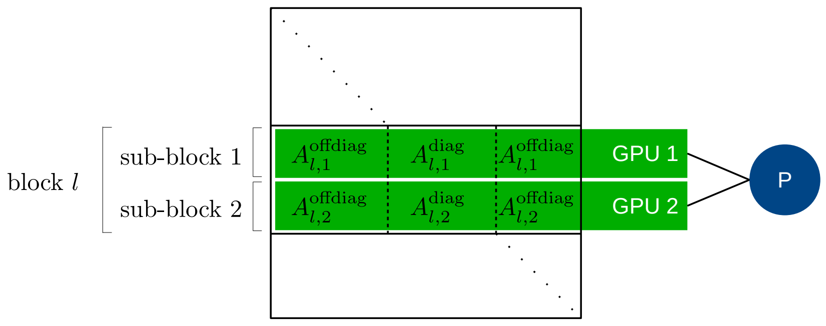

block l
sub-block 1
sub-block 2
$A_{l,1}^{\text{offdiag}}$
$A_{l,1}^{\text{diag}}$
$A_{l,1}^{\text{offdiag}}$
$A_{l,2}^{\text{offdiag}}$
$A_{l,2}^{\text{diag}}$
$A_{l,2}^{\text{offdiag}}$
GPU 1
GPU 2
P

Fig. 3: Sub-block decomposition in case of multi-GPU usage by a single MPI process.

3.3 Solver parameters

As the working precision, denoted high precision in the context of MPIR, we chose IEEE 754 double precision floating point format, and as low precision we chose IEEE 754 single precision floating point format [In85]. We set an absolute tolerance of $\varepsilon = 10^{-6}$ as stopping criterion for the iterative refinement method in Algorithm 1. This absolute tolerance is achievable in both double and single precision. For the error correction solver, we chose a relative tolerance of $\delta = 10^{-1}$ in Algorithm 2. This resulted in several error correction loops, each improving the high precision residual by the factor 10^{-1}.

3.4 Hardware and measurement system

Our test system consisted of one compute node equipped with 4 x Intel Xeon E-4650, 512 GByte DDR3 main memory and 2 x Nvidia Tesla K40. We used GCC compiler version 4.8.2, OpenMPI version 1.6.5, CUDA version 6.5.12, and NVIDIA device driver version 340.65.

For power measurement, we used the ZES Zimmer Electronic Systems LMG450 external power meter. Our test system comprises two power supply units, each connected with one line to the external power source. The LMG450 has four independent measurement channels. We used one channel for each of the two input lines, and the other two channels were left unused. We attached the power sensors of the LMG450 to the input lines between the external power source and the power supply units of the compute node. Thus, we measured the total power consumption of the whole node. We used the maximum possible sampling rate of 20 Hz of the LMG450 power meter. The measurement was controlled using the `pmlib` tool [Ba13]. We instrumented the solver code using the `pmlib` client API to measure exactly that portion of the overall program which constitutes the solution process. This excluded all initialization overhead from the measurements. The `pmlib` server ran on

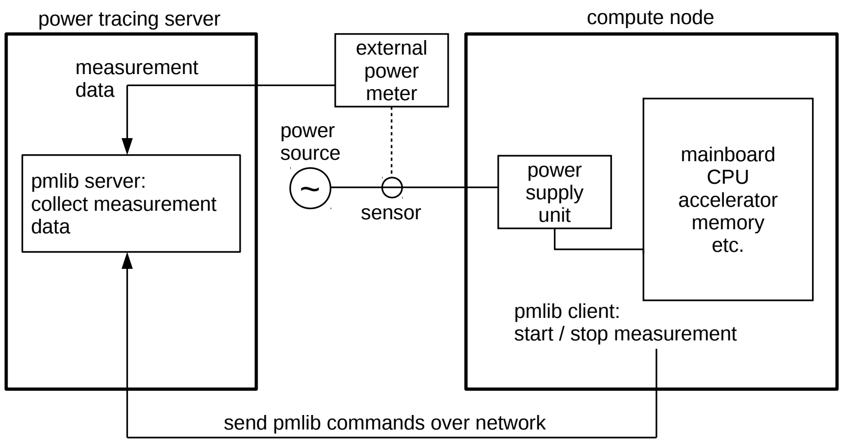

power tracing server
measurement data
pmlib server: collect measurement data
external power meter
power source
sensor
compute node
power supply unit
mainboard CPU accelerator memory etc.
pmlib client: start / stop measurement
send pmlib commands over network

Fig. 4: Measurement setup using an external power meter controlled by the pmlib tool.

a separate machine to avoid a perturbation of the system under investigation. The setup is shown in Figure 4.

4 Results

Through empirical testing, we figured out a number of $m = 20$ local block updates to be a reasonable choice for the linear system at hand. We carried out three series of tests:

1. **MPIR async-(20) GPU**
   Mixed precision iterative refinement using block-asynchronous iteration as error correction solver in single precision running on the GPUs.
2. **dp IR async-(20) GPU**
   Iterative refinement using block-asynchronous iteration as error correction solver in double precision running on the GPUs.
3. **MPIR async-(20) CPU**
   Mixed precision iterative refinement using block-asynchronous iteration as error correction solver in single precision running on the host CPUs.

We defined these test series to evaluate two effects. On the one hand, to evaluate the effect of using single precision error correction in contrast to using double precision error correction. On the other hand, to evaluate the effect of using accelerators in contrast to using only the host CPUs.
Figures 5 and 6 show plots of the performance related data of runtimes and total number of iterations. The parallel configuration is denoted $p \times t$, where p is the number of MPI processes, and t is the number of OpenMP threads per MPI process. We scheduled the MPI processes to run on distinct CPUs when using $p = 1,2,4$, and to equally share CPUs when using $p = 8,16,32$. The number of OpenMP threads was chosen to use all of the eight cores of the CPU available for the corresponding MPI process.

The host-only test runs from MPIR async-(20) CPU showed a reduction of the runtime for p ranging from 1 to 32. The fact that speedups were clearly inferior to the ideal linear speedup can be explained by the increasing number of iterations and the increased communication overhead. The phenomenon of increasing number of iterations for growing p

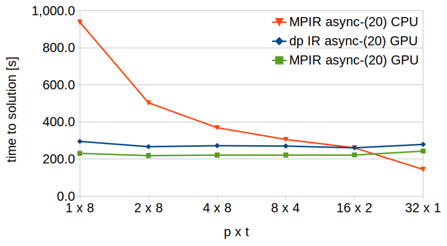


Fig. 5: Time to solution plot.

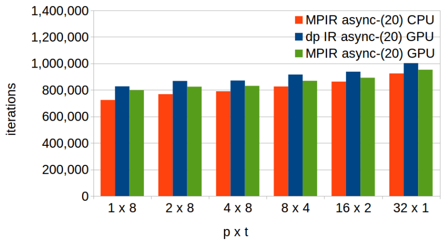


Fig. 6: Total number of iterations.

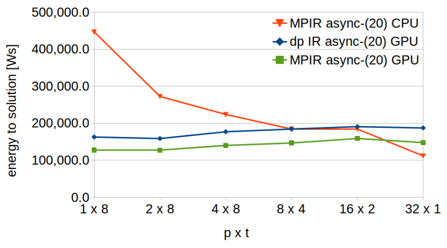


Fig. 7: Energy to solution plot.

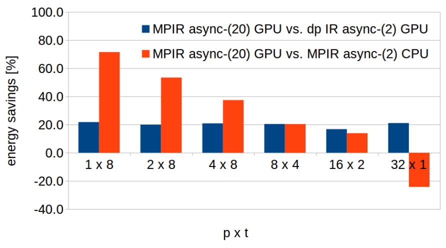


Fig. 8: Energy savings of MPIR async-(20).

was similar for all methods we tested. The reason is the decomposition of the matrix and vectors into smaller blocks, causing the local vector parts to shrink and the non-local parts to grow. Thus, the local block updates include more potentially outdated information from the non-local vector parts, and an increased number of overall iterations is necessary to compensate this effect.

In contrast, the methods using the GPUs showed a different behavior. The runtimes were nearly constant for $p = 2,4,8,16$ with slightly larger runtimes for $p = 1$ and $p = 32$. The GPU methods perform almost the whole computations on the GPUs. Only the double precision residual computation in step 9 of Algorithm 1 is performed on the host CPUs, but this computational effort is negligible. Instead, the nearly constant runtime reflects the fact that the sum of the problem sizes of all processes using the same GPU is constant, namely half of the total problem size.

Using single precision error correction instead of double precision gave approximately 20% improvement in the runtimes. The pure floating point arithmetic is twice as fast in single precision than in double precision, but our algorithms require data transfer between host and devices and across MPI processes for each update of the non-local vector parts. Although we used the fast transfer between devices and page-locked host memory, these memory copy operations required a substantial portion of the overall runtime. The high precision residual computation on the CPUs and the typecasts in case of mixed precision were negligible in this context, since they were not performed every 20 iterations, but only once for each error correction solving loop. Altogether, the runtimes of GPU methods were advantageous over the CPU method for $p = 1,2,4,8$. For $p = 16$, dp IR async-(20) GPU and MPIR async-(20) CPU had nearly equal runtime, while MPIR async-(20) GPU was still faster. Finally, for $p = 32$ the CPU method outperformed the GPU methods.

As Figure 7 shows, the energy consumption of the methods strongly correlated to the runtimes. Figure 8 shows the energy savings of MPIR async-(20) GPU compared to dp IR async-(20) GPU and MPIR async-(20) CPU. We calculated the percentages relative to the latter two methods. Using mixed precision instead of only double precision in the GPU methods gave savings of about 20% with slight variances for $p = 1$ and $p = 16$. However, performing the computations on the GPUs instead of CPUs gave massive savings of $\approx$71% for $p = 1$ and $\approx$53% for $p = 2$. We observed still remarkable savings of $\approx$37% for $p = 4$, while the benefit lay around 20% for $p = 8,16$. Only in the case $p = 32$, the CPU method consumed $\approx$24% less energy than the GPU method. We calculated this last percentage relative to MPIR async-(20) GPU.

5 Conclusion

We investigated block-asynchronous iteration methods for solving linear systems of equations with respect to performance and energy consumption. We introduced the mathematical background of mixed precision iterative refinement and of block-asynchronous iteration methods. We presented our implementation of these methods with support for distributed memory systems be means of an MPI parallelization, as well as shared memory support using OpenMP, and support of CUDA-capable accelerator devices. We ran a series of tests on a compute node equipped with four Intel Xeon E-4650 CPUs and two Nvidia Tesla K40 GPUs. We varied the number of MPI processes and OpenMP threads for the different test runs. All GPU tests used both devices. We designed the tests to assess the effect of using mixed precision instead of plain double precision computations, and to assess the effect of employing accelerator devices for the computations instead of only using host CPUs. We measured performance in terms of runtime, and energy consumption was measured with the help of an external high precision power meter.
We found that massive runtime and energy savings of more than 70% are possible on GPU-accelerated systems compared to CPU-only platforms. However, the actual amount of saved energy for a particular test run depends on the parallel configuration of MPI processes and OpenMP threads. We also found that CPU-only computations may outperform the GPU-accelerated methods if enough CPU resources are available. Also, we found that using mixed precision instead of only double precision gives a benefit of roughly 20% for runtime and energy consumption in the GPU tests. The frequent data transfer between host and devices imposes a substantial overhead which diminishes the impact of the doubled performance of the single precision floating point arithmetic.
Our results show that using accelerators for block-asynchronous iteration methods combined with mixed precision can lead to tremendous benefits in terms of runtime and energy consumption. The largest benefits can be expected for small host systems with only 16 or even less cores. This fits many HPC systems where often not more than two CPUs are available per node. On the other hand, large host systems or "fat nodes" may provide superior performance over the GPUs.

Acknowledgement

This work was supported by the European Commission as part of the collaborative project "Exa2Green - Energy-Aware Sustainable Computing on Future Technology"(research grant no. 318793).

References

[An11a] Anzt, H.; Dongarra, J.; Gates, M.; Tomov, S.: Block-asynchronous multigrid smoothers for GPU-accelerated systems. EMCL Prepr. Ser., 15, 2011.

[An11b] Anzt, H.; Dongarra, J.; Heuveline, V.; Luszczek, P.: GPU-Accelerated Asynchronous Error Correction for Mixed Precision Iterative Refinement. EMCL Prepr. Ser., 17, 2011.

[An12] Anzt, H.; Wilhelm, F.; Weiß, J.P.; Subramanian, C.; Schmidtobreick, M.; Schick, M.; Ronnas, S.; Ritterbusch, S.; Nestler, A.; Lukarski, D.; Ketelaer, E.; Heuveline, V.; Helfrich-Schkarbanenko, A.; Hahn, T.; Gengenbach, T.; Baumann, M.; Augustin, W.; Wlotzka, M.: HiFlow3: A Hardware-Aware Parallel Finite Element Package. pp. 139–151, 2012.

[An13] Anzt, H.; Tomov, S.; Dongarra, J.; Heuveline, V.: A block-asynchronous relaxation method for graphics processing units. J. Parallel Distrib. Comput., 73:1613–1626, 2013.

[Ba99] Bai, Z.Z.; Migallon, V.; Penades, J.; Szyld, D.B.: Block and asynchronous two-stage methods for mildly nonlinear systems. Numerische Mathematik, 82:1–20, 1999.

[Ba09] Baboulin, M.; Buttari, A.; Dongarra, J.; Kurzak, J.; Langou, J.; Langou, J.; Luszczek, P.; Tomov, S.: Accelerating scientific computations with mixed precision algorithms. Computer Physics Communications, 180:2526–2533, 2009.

[Ba13] Barrachina, S.; Barreda, M.; Catalan, S.; Dolz, M.F.; Fabregat, G.; Mayo, R.; Quintana-Orti, E.S.: An Integrated Framework for Power-Performance Analysis of Parallel Scientific Workloads. In: ENERGY 2013: The Third Int. Conf. on Smart Grids, Green Communications and IT Energy-aware Technologies. 2013.

[CM69] Chazan, D.; Miranker, W.: Chaotic relaxation. Lin. Alg. Appl., 2:199–222, 1969.

[EFS05] El Baz, D.; Frommer, A.; Spiteri, P.: Asynchronous iterations with flexible communication: contracting operators. J. Comp. App. Math., 176:91–103, 2005.

[EG04] Ern, A.; Guermond, J.-L.: Theory and Practive of Finite Elements. Springer, 2004.

[FS00] Frommer, A.; Szyld, D.: On asynchronous iterations. J. Comp. Appl. Math., 123:201–216, 2000.

[In85] Institute of Electrical and Electronics Engineergs: , IEEE Standard for Binary Floating-Point Arithmetic, 1985.

[Me11] Meister, A.: Numerik linearer Gleichungssysteme. Vieweg+Teubner, 2011.

[Me12] Message Passing Interface Forum: , MPI: A Message Passing Interface Standard, Version 3.0, 2012.

[NV12] NVIDIA Corporation: , Kepler GK110, 2012.

[NV14a] NVIDIA Corporation: , CUDA Toolkit Documentation v6.5, 2014.

[NV14b] NVIDIA Corporation: , Multi-Process Service, 2014.

[Op13] OpenMP Architecture Review Board: , OpenMP Application Program Interface, 2013.

[Ro69] Rosenfeld, J.: A case study in programming for parallel processors. Commun. ACM, 12:645–655, 1969.

[Sa00] Saad, Y.: Iterative Methods for Sparse Linear Systems. 2 edition, 2000.

[WSC14] Wende, F.; Steinke, T.; Cordes, F.: Multi-threaded Kernel Offloading to GPGPU Using Hyper-Q on Kepler Architecture. ZIB Report, Konrad-Zuse-Zentrum für Informationstechnik Berlin, 2014.

[ww14] www.green500.org: , The Green500 List, November 2014.

[ww15] www.top500.org: , TOP500, June 2015.

Test@Cloud – A Platform for Test Execution in the Cloud

Steffen Limmer[1], Alexander Ditter[1], Dietmar Fey[1], Matthias Pruksch[2],
Christopher Braun[2], Florian Drescher[2], and Martin Beißer-Dresel[2]
[1]Friedrich-Alexander University Erlangen-Nürnberg (FAU)
Chair of Computer Architecture, Erlangen, Germany
{steffen.limmer, alexander.ditter, dietmar.fey}@cs.fau.de
[2]sepp.med GmbH
Röttenbach, Germany
{matthias.pruksch, christopher.braun}@seppmed.de
{florian.drescher, martin.beisser-dresel}@seppmed.de

Abstract: The generation and the execution of tests for complex software systems are usually very time consuming tasks. With the help of model-based testing it is possible to accelerate the process of generating test cases. Thus, it is possible to automatically generate a large number of test cases in a small amount of time in order to achieve an accurate test coverage. However, with an increasing number of test cases the problem of the long execution time is intensified.

A possible solution can be the parallel execution of the tests on compute resources of a public or private cloud. In the project Test@Cloud a service was developed that enables the distributed execution of tests in cloud infrastructures. The present paper describes this service and its operating principle. Additionally, the results of an evaluation of the service with help of benchmark measurements are presented. The evaluation shows that the service can yield notable Speed-Ups. By distributing the execution of a test series over 19 virtual machines, it could be executed in 32 minutes, instead of 10.5 hours like it was the case when only one virtual machine was employed.

1 Introduction

Since the complexity of software systems is growing, two consequences are arising. First, the number of test cases to achieve a specified test coverage grows exponentially and with that the effort for the test design. And second, the time for test execution also explodes. An appropriate way to decrease the design effort is the usage of *model-based testing*, where the system under test (SUT) is described by one or more behavioral models. These models can be easily processed by a machine to automatically generate independent test cases according to a predefined test coverage strategy. Thus, a huge number of test cases can be generated in a minimum of time.

Of course, it is worthless to be able to generate more test cases in a shorter time, if these test cases cannot be executed in an acceptable time. Fortunately, the independence of the test cases allows the parallel execution of them. Employing parallelism can shorten the test execution times fundamentally. This makes cloud technology attractive for the execution of tests. In a cloud a virtually arbitrary number of virtual machines (VMs) can be started and used for the processing of tests. This enables users to meet peak demands in computational power without the need to acquire and maintain the necessary hardware resources. The usage of virtualization makes it possible for the user to install a runtime environment for his/her tests as required.

In the project Test@Cloud a platform was developed, which enables testing in the cloud. With the help of a cloud service, it is possible to start, control, and monitor the execution of test cases distributed over multiple VMs in a cloud.

The rest of the paper is organized as follows: In the next Section the platform underlying the Test@Cloud service is described before the service itself is described in detail in Section 3. Section 4 presents results of an experimental evaluation of the Test@Cloud platform and finally, Section 5 closes the paper with a short conclusion and outlook.

2 Overview Test@Cloud Platform

The execution of test cases in the cloud is done over a specific Test@Cloud service. This service was implemented on the basis of a platform, which emerged from the project Cloud4E [1]. Before the service is described in detail in the next section, this section provides an overview over the underlying platform.

In order to avoid vendor lock-ins, *OCCI (Open Cloud Computing Interface)* [2] is used for the remote control of services. OCCI is an open standard for a cloud interface developed by the Open Grid Forum. In its current form, OCCI is mainly intended for the control of Infrastructure as a Service (IaaS) – for example, starting, monitoring, and stopping VMs. However, in Cloud4E the OCCI interface was extended to allow the control of Software as a Service (SaaS). In order to enable the control of SaaS over OCCI, certain extensions to the so called *rOCCI server* [3] were implemented.

The rOCCI server is a Ruby implementation of an OCCI server together with a corresponding client library, which can be used for the provisioning of an OCCI interface to an existing cloud infrastructure. An overview over the architecture of the rOCCI server can be seen in Figure 1. The server consists of a frontend for the communication with clients over OCCI and multiple backends for the control of different cloud middlewares over their corresponding proprietary interfaces. Currently there are backends for OpenNebula and OpenStack, an EC2 compatible backend for the control of the Amazon Web Services or Eucalyptus and a dummy backend for testing purposes. Besides extensions that enable the control of SaaS over OCCI, another extension was made to the rOCCI server that allows the usage of the *Advanced Message Queuing Protocols (AMQP)* [4, 5] as transport protocol. The usual transport protocol for the OCCI communication is HTTP. AMQP provides certain advantages compared to HTTP. For example, it supports asynchronous communication and the broadcast of messages to multiple communication partners. The most important advantage is that all the communication is done over a central AMQP server. Thus, only the AMQP server has to be publicly reachable and the communication partners do not have to reach each other directly. This permits the communication between resources of different cloud providers, which in turn enables the usage of a federated cloud.

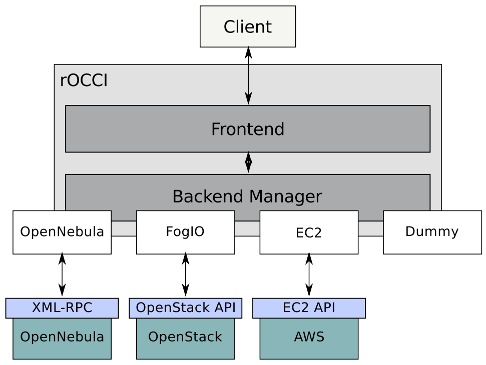


Figure 1: Overview over the architecture of the rOCCI server.

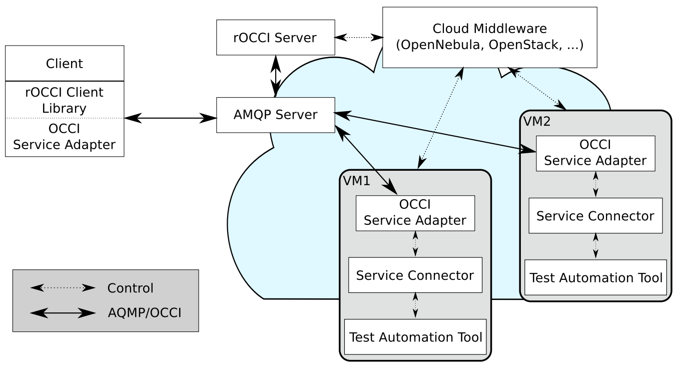


Figure 2: Overview over the basic components of the Test@Cloud platform.

Figure 2 gives an overview over the entire Test@Cloud platform. The central components are a rOCCI and an AMQP server, which can run on a physical node or a VM in the cloud. It is even possible to deploy them outside of the cloud at the client side. Over the AMQP server all the OCCI communication is done. The rOCCI server is attached to one (or if necessary to multiple) cloud middleware(s) for the IaaS management. In the project Test@Cloud OpenNebula is employed as cloud middleware. With help of the rOCCI client library a client can start VM instances over the rOCCI server in the cloud. In OCCI VMs are represented in form of so-called *OCCI compute resources*. Such OCCI compute resources provide attributes that can be used to obtain information about the VM like the number of physical cores or the status of the VM. Additionally, OCCI compute resources provide actions that allow to perform certain operations on a VM – e.g. stopping or restarting the VM. Over these attributes and actions a running VM can be controlled over OCCI. With help of the afore-mentioned extensions to the OCCI interface, it is possible to control cloud services in a similar way. The services are executed in VMs in the cloud and consist of three components.

One component is a test automation tool (like Ranorex or HP Unified Functional Testing).

A second component is a so called *OCCI Service Adapter (OSA)*. This is a daemon that provides an OCCI interface for the test automation tool in form of actions and attributes. After the start it registers the actions and attributes to the rOCCI server. This is done by linking the actions and attributes in form of so-called *OCCI mixins* to the OCCI compute resource of the VM in which the OSA is running. After the OSA has registered the interface to the rOCCI server, it listens for incoming OCCI requests and handles them – for example, when an action of the registered interface is triggered.

The declaration and implementation of the actions and attributes of the service interface is done in a third component of the service: the so called *Service Connector*. This is a Ruby class, which declares the available attributes and actions in a special syntax (domain specific language) and which implements the actions in form of usual Ruby methods. One can say that the OSA together with the Service Connector form some kind of OCCI wrapper around an existing application (a test automation tool in our case), whereby the OSA is application independent and everything application dependent is specified and implemented in the Service Connector.

After the interface of a service is registered at the rOCCI server, a client can use it with help of a library provided by the OSA. Over this library the client can query the available actions and attributes of the service interface from the rOCCI server and use them in order to control the service. For the control of a service, requests can be sent either directly to the OSA of the service or to the rOCCI server, which redirects the requests to the OSA. Since the OSA provides client-side functionality, it can be used for the communication between different Service Connectors. For example, it is possible that one Service Connector queries the attributes of another Service Connector.

In order to get detailed information over the state of a service or to investigate failures related to a service, it is important that the client can retrieve log messages from the service. Since AMQP is already used for the control of the services, it is also employed for the transfer of log messages. With help of library functions of the OSA, Service Connectors can send log messages to the AMQP server. Afterwards the client can retrieve the messages from the AMQP server. The AMQP protocol permits that the messages can be retrieved simultaneously by other applications. Thus, it is possible to retrieve the messages with the client and at the same time to use a tool like Logstash [6] for the advanced storage, filtering, and post-processing of the messages.

The data management in the Test@Cloud project is done over Samba and SSHFS (Secure Shell File System). Every cloud user is provided with a portion of a shared file system that is located on a physical host of the cloud. The file system can be mounted via Samba or via SSHFS on the local host of the user as well as in the VMs. That makes it possible to provide input files for services and to retrieve output files from computations executed over services. Furthermore, data can be shared among multiple service instances. However, the Test@Cloud service is mostly independent from the concrete data management solution. Thus, it is possible to replace the Samba and SSHFS based data management with another solution without big effort.

3 Test@Cloud Service Connector

Based on the platform described in the last section, a service – or more precisely a Service Connector – was implemented that allows the execution of test cases over different test automation tools in the cloud. So far, the service was tested with the two popular test automation tools Ranorex and HP UFT. Both tools allow to create test cases in form of so called *test suites* that can be executed over the test tool. Thus, the Test@Cloud service has to be able to execute such test suites distributed in the cloud. In order to make it able to distribute test suites over multiple VMs, the service works according to the master-worker-scheme, like illustrated in Figure 3. The client controls a master service that in turn controls multiple worker services over which the executions of the test cases are distributed. Over the Samba and SSHFS based cloud file system (cloud FS), the master and

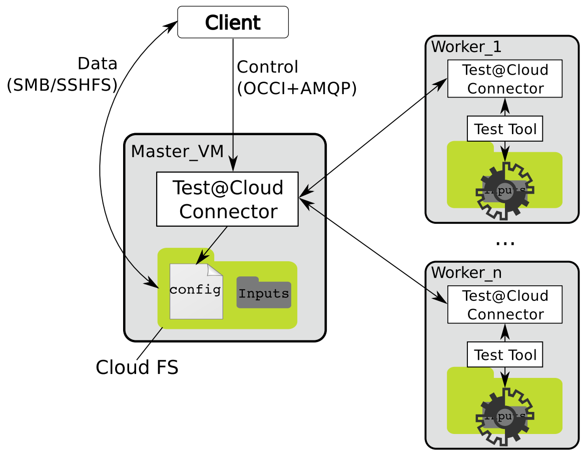


Figure 3: Operating principle of the Test@Cloud service according to the master-worker-scheme.

the workers have collective access to shared files. The Test@Cloud connector provides the following four actions, which can be triggered by the client in order to control the execution of test series: *start_workers*, *stop_workers*, *start_test_series*, and *abort_test_series*.

The execution of a series of test suites with help of the Test@Cloud service is done in the following way: First, the user copies all the test suites belonging to the test series from his local host to the cloud FS. Additionally he creates a configuration file on the cloud FS, which contains all the information about the test series that is relevant for the master service. It specifies the files of the test suites to execute, the number of workers that should be used for the execution of the test suites and from which VM templates[1] the workers should be started. After the user has copied all input files to the cloud FS, he starts the VM for the master service. Inside of the VM automatically the master service is started after the VM is booted. Afterwards, the user triggers the action *start_workers* of the master and passes the path to the configuration file on the cloud FS as argument to

[1]A VM template is a configuration of a VM that can be used by the cloud middleware to start the VM accordingly.

the action. The configuration file is parsed by the master and thus it knows the test suites to start and the desired number(s) and type(s) of workers. This information is used by the master to start the requested workers.

When all workers are started, the actual test execution can be initiated. This is done by triggering the action *start_test_series* of the master. That causes the master to start the individual test suites on the workers and to ensure that thereby no test suite is started on the wrong type of worker (according to the specifications in the configuration file passed to the action *start_workers*). Initially, each worker is assigned with one test suite. When a worker has finished the execution of its test suite, it contacts the master, which assigns the worker with the next test suite (if there are unassigned test suites left).

After all test suites are processed or if necessary also during the execution, the user can access the outputs of the tests over the cloud FS. When the workers are no longer required, the master can be prompted to stop them over the action *stop_workers*. The last step of the test execution in the cloud is the shutdown of the master VM.

During the execution of a test series the user is able to abort the test series at any time. This is done by triggering the action *abort_test_series*.

In the project Test@Cloud a prototypical graphical client was implemented, which allows the convenient control of the Test@Cloud service.

4 Experimental Evaluation

In order to evaluate the described cloud platform, a series of benchmark measurements was performed. An OpenNebula based cloud infrastructure on the compute resources of the Chair of Computer Architecture of the FAU served as testbed for the measurements. On the head node (with two AMD Opteron 2435 hex-cores) of this infrastructure version 0.5 of the rOCCI server runs over Ruby 2.0.0 and OpenNebula 4.0.1 functions as cloud middleware. Multiple compute nodes (with two 2.4 GHz AMD Opteron 2216HE dual-cores, each) are connected to the middleware and can be used to host VMs, whereby QEMU 1.6.1 is used as hypervisor for the VMs. The communication between physical machines is done over 10 Gbit/s InfiniBand and VMs are connected to a Gigabit Ethernet LAN. For the measurements VMs with Windows XP as operating system were used. For the scheduling of VMs to physical hosts, a self-implemented scheduler *i3sched* [7] is used instead of the OpenNebula scheduler. The *i3sched* delegates the scheduling decisions to a batch system (Slurm) in order to better integrate the cloud platform in an existing cluster. It has to be noted that *i3sched* is part of the cloud infrastructure and not of the Test@Cloud platform, which is independent of the scheduler.

In a first step, we determined how much time it takes to start multiple workers (the worker VMs together with the worker services inside them). For that purpose we measured the timespan between triggering the *start_workers* action of the master and the moment, when the master recognizes that all workers are started, for different numbers of workers to start. The times averaged over 10 measurements can be seen as T_s in Figure 4. In order to determine how much of these times are the pure start times of the VMs, we equipped the

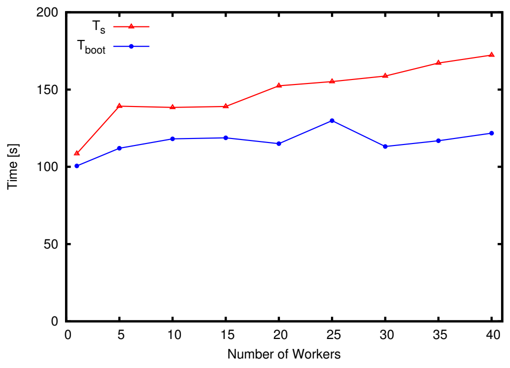


Figure 4: Time it takes to start and boot different numbers of worker VMs over OpenNebula (T_{boot}) and time it takes to start the worker VMs and additionally the worker services inside the VMs (T_s).

worker VMs with a simple client, which starts automatically after the boot and contacts a server on the head node of the cloud. For different numbers of VMs we measured how long it takes from the start of the VMs over command line tools of OpenNebula until all clients inside the VMs have contacted the server. These times (averaged over 10 measurements) are shown as T_{boot} in Figure 4. The scheduling interval of *i3sched* was set to 10 s, implying that 10–20 s of T_{boot} and of T_s arise from the scheduling.

It can be seen that the start times T_{boot} of the worker VMs stay nearly constant with an increasing number of workers to start and that they lie in the range of about two minutes. The times T_s, which include the start times of the worker services, grow with an increasing number of workers to start. The reason is that the worker services have to register their interface to the rOCCI server during their start and this takes more time, the more worker services are started simultaneously. But nevertheless, the start times stay in an acceptable range. The start of 40 workers can be done in approximately three minutes. A more detailed investigation of the start times and a description of optimizations that were done to keep the start times in an acceptable range can be found in [8].

In a next series of measurements we determined the overhead caused by the distribution of test cases over multiple workers. When a test series is started, the master service contacts the available worker services over AMQP and delegates test cases to them (one test case per worker). The worker services start their test cases on the command line and when a worker service has finished its test case it contacts the master, which delegates a next test case to the worker. This requires a certain amount of time. In order to quantify this time, we measured how long it takes to execute $2n$ test cases on n workers, whereby nothing is calculated in the test cases. Thus, the measured times are the pure overhead associated with the distribution of the test cases. They can be seen in Figure 5 for different numbers of workers and averaged over 10 measurements.

The overhead lies in the range between one and two seconds. The execution of 60 “empty” test cases on 30 workers causes an overhead of about 1.7 s. Thus, if the runtimes of the test cases to execute are very short, this overhead has a noticeable impact on the overall

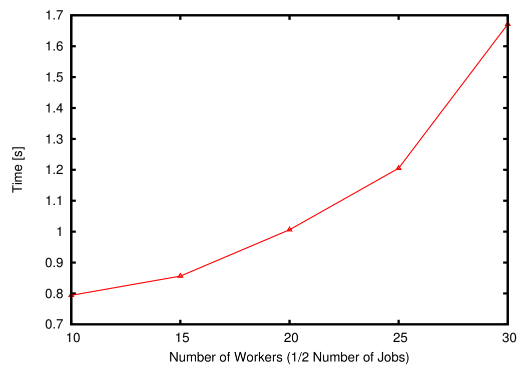


Figure 5: Time to execute $2n$ "empty" test cases on n workers for different values of n.

runtime. But it can be assumed that in most practical cases the overhead is tolerable.

In order to evaluate the Test@Cloud platform regarding its suitability for the practical usage and to investigate the scalability of test executions in the cloud, we executed a series of 1296 test cases (or test suites, respectively) in the described testbed. The test cases are provided in form of executable Ranorex (version 4.1) test suites. The SUT is the tool MBTsuite (a tool to generate test cases from models). The test series covers all possible combinations of inputs to a dialog GUI window of the MBTsuite. The GUI dialog contains text/numerical input fields, combo, and check boxes. The test cases have been derived automatically from a model of the SUT. Each test case consists of 15 to 17 test steps, ranging from starting the SUT, the various steps of filling the dialog's elements, checking the response of the SUT, and the final cleaning of the SUT. Test cases with both, positive and negative expected results are taken into account. Their generation is based on threshold related test development methods, for instance the input of numerical values, which are limited to a certain range or unexpected text inputs. Thus, the test series covers all possible scenarios which are taken into account by the model, this is often referred to as a *full-path-coverage*.

In order to get an impression of the execution times of the single test cases, we first executed them on a local PC (with an 3.4 GHz Intel Core i7-3770 quad-core) outside of the cloud. The shortest runtime of a test case was 15.6 s. Only one of the 1296 test cases had with 41.89 s a runtime over 25 s. 1258 test cases had execution times between 15 s and 20 s and for 37 test cases the execution time lay between 20 s and 25 s. In average the test cases took 17.69 s to execute and the total runtime of the complete test series was 6.4 h. Thus, the test cases have relatively uniform and short runtimes.

The execution of the test series in the cloud was done distributed over 1, 2, 4, 8 and 19 workers with two cores (VCPUs) per worker VM. The runtimes averaged over 5 measurements are shown in Figure 6. They do not contain the start times of the workers (Figure 4).

With one worker, the runtime is about 10.5 h and thus it is notably higher than on the local PC outside of the cloud. But this comes as no surprise since the employed PC is

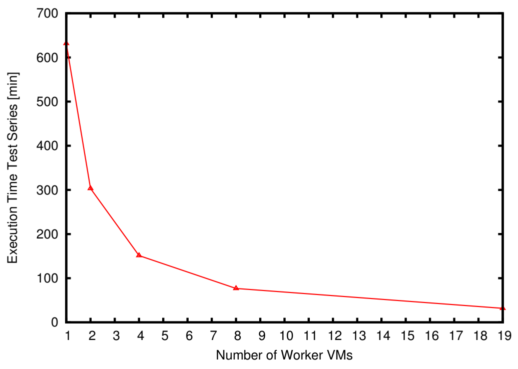


Figure 6: Average time to execute a series of 1296 test cases on the Test@Cloud platform distributed over different numbers of workers with two cores per worker VM.

significantly more performant than the physical hosts in our available cloud testbed.

With two workers the average execution time reduces to 5.06 h. With four workers it is further reduced to 2.52 h. With eight workers the execution time is 1.28 h and with nineteen workers it is 0.53 h. The standard deviation of the measured execution times ranges from 0.44 min for nineteen workers to 8.96 min with one worker.

The average speed-up values with respect to the execution with one worker can be seen in Figure 7. It is slightly super-linear. For nineteen workers the speed-up is around 19.89.

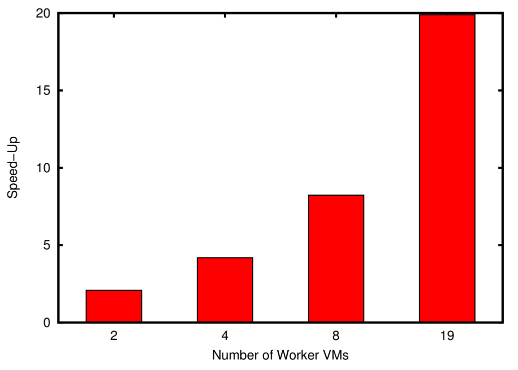


Figure 7: Average speed-up of the execution of the test series distributed over multiple workers with respect to the execution on one worker.

Although this might be the result of measurement deviations, it shows that no relevant overhead is introduced by the distribution of the test cases over multiple worker VMs.

5 Conclusion

In the present paper we proposed a platform, which allows the remote start and control of the execution of test cases in cloud infrastructures independently of the employed test automation tool. In an experimental evaluation its applicability for the practical usage is shown.

In order to transfer the current prototypical implementation into a production-ready status, we plan to deploy it on the resources of a commercial cloud provider and to start a piloting phase with representative test users. For that reason, we are currently working on an OCCI binding to the Windows Azure Pack cloud platform.

Furthermore, it is planned to perform more detailed measurements and to investigate the validity of non-functional tests in virtual environments.

Acknowledgment

The Test@Cloud project is publically funded in the context of the program *Informations- und Kommunikationstechnik (IuK)* of the German federal state Bavaria.

References

[1] Cloud4E website: `http://www.cloud4e.de/`, Retrieved March, 2015.

[2] A. Edmonds, T. Metsch, A. Papaspyrou, A. Richardson: Toward an Open Cloud Standard, Internet Computing, IEEE , vol.16, no.4, pp. 15–25, 2012.

[3] rOCCI server at GitHub: `https://github.com/gwdg/rOCCI-server`, Retrieved March, 2015.

[4] Advanced Message Queueing Protocol: `http://www.amqp.org/`, Retrieved March, 2015.

[5] S. Vinoski: Advanced Message Queuing Protocol, IEEE Internet Computing, vol. 10, no. 6, pp.87–89, 2006.

[6] Logstash website: `http://logstash.net/`, Retrieved March, 2015.

[7] A. Ditter, S. Limmer, D. Fey: i3sched – Ein OpenNebula Scheduler für die Oracle Grid Engine (in German), Proceedings of Grid4Sys 2013, pp. 1–4, 2013.

[8] S. Limmer, M. Srba, D. Fey, Dietmar: Performance Investigation and Tuning in the Interoperable Cloud4E Platform, FedICI 2014, LNCS vol. 8806, pp. 86-97, Springer International Publishing Switzerland, 2014.

Quantifying Performance and Scalability of the Distributed Monitoring Infrastructure SLAte

Marcus Hilbrich, Ralph Müller-Pfefferkorn

Software Quality Lab (s-lab)
Universität Paderborn
marcus.hilbrich@uni-paderborn.de
Center for Information Services and High Performance Computing
Technische Universität Dresden
ralph.mueller-pfefferkorn@tu-dresden.de

Abstract: Job-centric monitoring allows to observe the execution of programs and services (so called jobs) on remote and local computing resources. Especially large installations like Grids, Clouds and HPC systems with many thousands of jobs can have large benefits from intelligent visualisations of recorded monitoring data and semi-automatic analyses. The latter can reveal misbehaving jobs or non-optimal job execution and enables future optimisations to establish a more efficient use of the allocated resources.

The challenge of job-centric monitoring infrastructures is to store, search and access data collected on huge installations. We take this challenge with a distributed layer-based architecture which provides a uniform view to all monitoring data. The concept of this infrastructure called SLAte, a performance evaluation, and the consequences for scalability are presented in this paper.

1 Introduction

Direct observability of computing tasks is more and more lost by using external and distributed resources for getting calculations done. Thus, misbehaving or obscure behaving jobs are rarely found and the optimisation potential of job execution is not satisfied. Job-centric monitoring is a service which takes the challenge to fill the gap between using external resources and direct observability of jobs. Therefore monitoring data of each job are recorded, giving detailed information about the used resources of running and completed jobs. Grid middlewares or batch systems usually just provide information about the job status like running or finished. Using more detailed information allows a semi-automatic analysis process to observe job execution with a minimal expenditure of time for users and administrators. It also enables the analysis of job failure reasons which is not possible by just checking the exit codes of jobs delivered by batch-systems.

Job-centric monitoring is most benefiting for environments with large numbers of jobs and resources which are shared by various users. In these systems, users cannot easily observe their jobs like on local resources by using desktop monitoring. Resource monitoring does not solve this problem because it does not provide job specific information. It observes only the characteristics of resource usage, not the impact of single users or concrete jobs.

The architectures we have in focus for job-centric monitoring are Grids (offering huge amounts of heterogeneous resources), Clouds (offering virtual resources on demand) and HPC or cluster systems operated by computing centres. For the measurements and analyses presented in this paper we focused on the D-Grid infrastructure[1] which is a research network for scientists in Germany and the Globus Toolkit 4 (GT4) [Fos05] Middleware's web services which were common for many different Grid projects.

One of the challenges is the development of analysis strategies [HWT13] which handle the huge quantities of job-centric monitoring data or measurement series in general. But before we can analyse the data we have to handle them.

Our answer to the challenge of storing, accessing and searching huge amounts of job-centric monitoring data is the infrastructure SLAte[2] [HMP10]. It is built on a concept of distributed servers organised in three layers. These layers allow to distinguish between different network capabilities (e.g. within a site[3] and between different sites), capacities of various storage locations, and the localities of users or analysis services. The performance of each of the three layers can be increased by installing additional servers and thus can be adapted to needs.

This paper presents work related to job-centric monitoring, shortly describes the SLAte architecture, gives a performance evaluation for the different server types of SLAte and analyses the results in the context of the scalability concept of the job-centric monitoring infrastructure. Closing, conclusions and future work are presented.

2 Related Work

Job or system observation is an already established concept used for different needs. So it has been realised by different fields of research:

Monitoring of local computing resources: For local monitoring command line tools like ps, top or free[4] and graphical ones like Gnome System Monitor[5] can be used. On clusters Ganglia[6] gives information about the utilisation of resources. The impact of a specific user or job cannot be identified directly by these tools.

Tracing and profiling: Profiling tools like GNU gprof[7] give information which functions of a program are used and how long they are used. This information is often presented as statistical evaluation. Even more detailed information are given by

[1]`http://www.dgrid.de/`

[2]SLAte stands for **S**calable **L**ayered **A**rchi**te**cture

[3]A site is a set of resources of a resource provider (in the Grid context) at a single location.

[4]`http://procps.sourceforge.net/index.html`

[5]`http://library.gnome.org/users/gnome-system-monitor/stable/index.html`

[6]`http://ganglia.info/`

[7]`http://sourceware.org/binutils/docs/gprof/`

tracing tools like Score-P [MBB+12] and Vampir [BHJR10]. To record the data the applications need to be instrumented, meaning adapted. Depending on the level of detail of the tracing a significant overhead may be introduced. Moreover, profiling or tracing infrastructures are usually designed to handle just one application/job and only on local resources. Thus it can not be easily adapted for job-centric monitoring.

Accounting: Accounting is used to measure resource utilisation and as a base for the billing of resources usage. An example is SGAS [EGM+]. Only basic and summary information of a job are needed and thus the amount of data to be handled is low. As a database is usually able to handle all accounting data, scalability is only a minor issue.

Resource monitoring: The task of resource monitoring is to record information about the (distributed) resources. Collected are information about e.g. hardware, offered services, outages and utilisation. This allows to create statistics or to assign jobs to resources. Examples are projects like D-Mon [BBK+09] and CMS Dashboard [ACC+10]. A lot of the information collected by these tools are static and have no need for a high frequency of updates.

The specific needs for handling job-centric monitoring data are not properly considered by any of these tools. Thus, we developed new concepts for job-centric monitoring which are explained in the following sections.

3 Architecture of SLAte

SLAte is designed to increase the performance of handling monitoring data by deploying additional servers. This is needed if more users access the monitoring or new computing resources are added. The network capacity may be one limiting factor, the CPU capacity to process the data or the storage capacity are other ones. When observing many jobs in SLAte the overall performance can be adjusted by the number of deployed servers. For easy access to the distributed monitoring data we provide a unified and global view to easily access data. To this we created a concept based on three layers which is schematically shown in Fig. 1. We use the Short Time Storage (STS) layer to receive data from the monitoring clients running on the compute nodes and to store the data temporarily. The Long Time Storage (LTS) layer accumulates the data from the STS servers, stores them persistently and distributed over multiple servers. The outer layer is the Meta Data Service (MDS) which provides the global view on the data.

The STS servers are to be installed local at a site to be close to the computing nodes on which the monitoring data are produced. To avoid that the monitoring data use too much resources on the computing node (e.g. if stored in memory) and to avoid that the monitoring data are copied after the job is done (which might prevent the next job to start), the monitoring information have to be moved to an STS server as early as possible [HMP10]. This results in a transfer for each single measurement and tends to many small packages which are handled by the local network (within a site or a computing system).

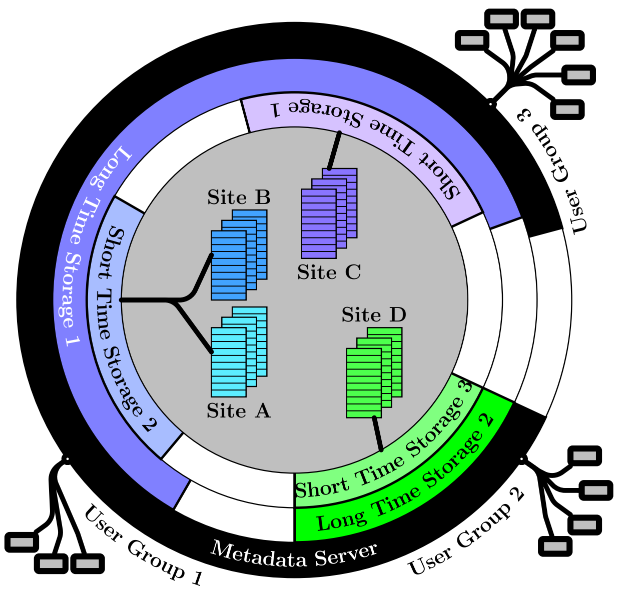


Figure 1: The layer-based SLAte infrastructure.

The monitoring data cached on the STS server have to be moved to the LTS server over a network connection which can be used more efficient when transferring packages of according size. To improve the performance of such a connection the monitoring data is transferred in batches. In most cases the monitoring data of one job are moved at once[8]. By repacking the monitoring data to large packages network congestion is avoided.

Like the STS layer, the LTS layer can consist of multiple servers to store the monitoring data in a distributed way. In contrast to the STS the LTS servers store the data persistently. The monitoring data of several individual STS servers can be merged (in Fig. 1 symbolised by the sites A to C). Furthermore, an LTS server makes locally stored data accessible and searchable to users and analysis systems.

To provide a unified view on the monitoring data distributed over the LTS servers the MDS layer is used (see Fig. 1). This layer can consist of multiple servers too. Unlike the LTS and STS servers an MDS server has a global view to all data. Via a single MDS server monitoring data can be easily accessed, without knowing other MDS or any of the LTS and STS servers.

In our architecture we distinguish between monitoring data and their meta data. The monitoring data are (time) series of measurements (like used CPU-time, load average, used or free memory) while the meta data hold information to search for jobs. Such informations are for instance the job ID, the time frame the job was active in and the executive system.

[8]To realise online monitoring it is needed to get monitoring data of running jobs. In this case the data are accessed on the STS server and the data recorded afterwards thus have to be transferred in an additional package.

To look up or search for jobs only the meta data are considered. This search can be performed by an LTS or an MDS server with the distinction that an LTS server holds only the meta data of the locally stored data while an MDS server has the meta data of all LTS servers but does not hold the job data. Thus an MDS server can search on all monitoring data and the amount of data transferred to the MDS servers is dramatically reduced. In concrete a search request is answered with a list of storage locations for the data on LTS servers. With this list, a client can access the data directly and download it in parallel.

Not covered with this paper is the topic of protecting user-related data. Our first publication which concerns security for SLAte was [HMP12b]. Later on a very detailed explanation of the used protection concept was shows by [Hil14], which also covers an analysis of the protection demand of the stored data.

4 Performance Evaluation

4.1 Hardware and Software Used

SLAte has been implemented as a demonstration of the layer-based concept in the context of D-Grid. Thus we used software and hardware common for this research federation. The STS, LTS, and MDS servers are based on GT4 web services and use the certificate-based authentication and authorisation of GT4 with the public-key-infrastructure of D-Grid. This is a proper usage scenario with the drawback of web services as performance bottleneck.

As test hardware (see Tab. 1) for the three types of servers we used systems which where common for the D-Grid infrastructure and which could be allocated for exclusive use for our experiments. These were *Sun Fire X4100*™ with dual core *AMD*® *Opteron™ 256* at a clock rate of 3.0 GHz equipped with 8 GB *DDR* main memory and *Gigabit Ethernet* network connection. It is clear that a more recent system could lead to a higher performance. To classify the results we give a short example of a system we used for production in D-Grid. This is a *Sun Fire X4600*™ with 16 CPU-cores at a clock rate of 2.6 GHz and 32 GB memory. This system was used as GT4 server at the Center for Information Services and High Performance Computing. During the usage we could observe that a submission of 500 to 1000 jobs could reproducibly overload the GT4 execution system. Even a more powerful system, the frontend of a cluster with an eight core *Intel*® *Xeon*® *X5365* at 3.0 GHz and 16 GB *DDR2* main memory was overloaded with about 1000 jobs, but the cluster offered enough nodes to generate load for our experiments without getting a bottleneck.

4.2 Principles of the Measurement

We started with performance measurements of the STS server. One job with eight hours runtime was excuted on the cluster with the job-centric monitoring enabled. We recorded the sampled monitoring data and their transfer to the STS server including the exact tim-

	GT4 server	load generation (clients)
CPU typ	*AMD® Opteron™ 256*	*Intel® Xeon® X5365*
CPU clock	$3.0\,GHz$	$3.0\,GHz$
cors per CPU	2	4
cors per node	2	8
RAM typ	*DDR*	*DDR2*
RAM per node	$8\,GB$	$16\,GB$
nodes per system	1	64
number of nodes used	1	up to 8 (only moderate load)
network	*Gigabit Ethernet*	*10 Gigabit Ethernet*

Table 1: Hardware used for the test system (GT4 server) and the system used for load generation.

ing. Based on the fact that the data volume is independent of the values of measurements (it depends on the number of measurements or runtime) a replayed job causes the same behaviour on the server like a real one. To test the load on the STS server, we needed to increase the amount of monitoring data sets sent to the STS server. This was done by replaying the previously recorded data simultaneously with the original clients and bash scripts to coordinate the timing. In this way we could use one CPU-core (at moderate load) of the cluster to simulate the sending of data of multiple jobs. This allows to simulate much larger systems than we used for our experiments. For the STS server (as like for the other server types) we did multiple tests with different loads. In the following we always refer to the run with the highest load without any overload or data lost.

To analyse the performance of a LTS server the output of STS servers is needed. To avoid to operate multiple STS servers, we recorded the output of a STS server including the timing. This was done by a separate run with lower number of jobs and debugging enabled. This record can be replayed and duplicated to adjust to a predefined load as in the case of the STS above. This was done by instrumenting the STS server's source code for recording and by implementing a client to replay the record.

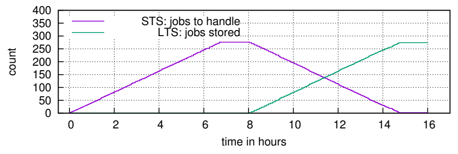
Finally, to test the MDS server the LTS server was instrumented and recording was done like for the STS server. Another client was implemented to replay the data.

4.3 STS Server

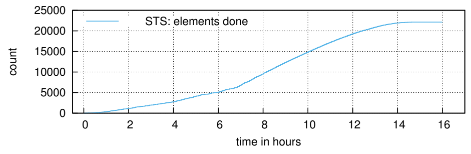
To test a STS server, the number of running jobs (jobs to handle in Fig. 2a) is increased constantly over seven hours. After eight hours the first jobs finish and the number of running jobs decreases until all jobs end after 15 hours. Thus the maximal load is reached after seven hours and holds for one hour. During this experiment it was tested that the STS server is not overloaded, which would result in a undefined state of the GT4 system and data loss.

For a practice-oriented scenario we need an LTS server at high load because this slows down the data transfer from the STS server. In preexaminations we discovered that the limitation for all our server types is the web server's CPU-load, not the network connection. So we decided to run the LTS server on the same hardware as the STS server. Based on the assumption, that the LTS can handle much more jobs than the STS server [HMP12a] we can justify this to get more practical relevant results. This is verified later on.

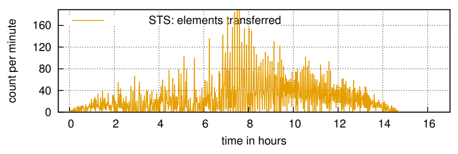
In our experiments we could use up to 276 jobs on the limited hardware described above. The result is shown in Fig. 2a. An increased job count leads to an overload of the STS server with discontinued data transfers. Also shown is the number of jobs stored on the LTS server. All job data were transferred and no data loss was seen during the experiments.



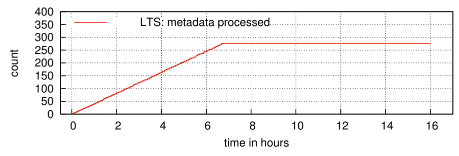
(a) Number of jobs currently using the STS server to store monitoring data and number of jobs whose monitoring data were transferred to the LTS server over time.



(b) Total number of received monitoring data sets over time.



(c) Number of monitoring data measurements processed per minute over time.



(d) Total number of transferred meta data packages over time.

Figure 2: Measurement results for the STS server

The transfer of the individual measurements (elements) to the STS server is shown in Fig. 2b. Each measurement has a size of 0.4 kB. Based on the fact that multiple measurements are done for each job, the number of measurements (22,131) is much higher then the number of jobs (276). Figure 2d shows the number of transferred meta data packages (2.4 kB each). For each job, exactly one package is sent.

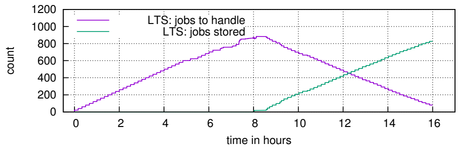
According to Fig. 2b (total number of elements) the data rate is not always constant which meets the exception of the measurement . The data rate is constant in case the number of jobs stays constant, so the number of elements sent per time is determined. According to Fig. 2a this is the case from hour seven to eight. Figure 2b shows the expected constant rate (lineare increase) for this time frame. From hour zero to seven the number of jobs rises linear (see Fig. 2a). Thus the rate of transferred elements rises linear, resulting in an quadratic increase of the number of elements in Fig. 2b. After eight hours, the number of jobs decreases linear (see Fig. 2a). Thus the rate of elements is linear lowered, resulting in an increase of the number of elements based on a square root function (the amount as integral of the rate, the rate is a linear function with negativ sign). In short, an increase of the number of jobs results in a more then linear increase of the number of elements while a decrease results in a less than linear increase. In the time frame with the maximum number of jobs (hour seven to eight) the rate of newly sent elements has its maximum and is constant.

The number of measurements/elements transferred to the STS server is also shown in Fig. 2c. For this illustration, the number of elements was added up for each minute. It shows the already mentioned global change of the data rate as a fine granular fluctuation which is not obvious in the last diagrams. There are even minutes without any data transfer. The reason for the high variance of the transfer rate is the absence of a coordination of the data transfer.

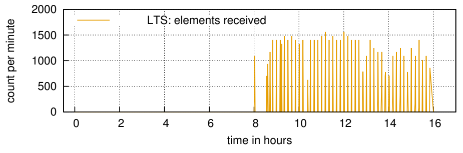
4.4 LTS Server

The LTS server test is based on the outcomes of the analysis of the STS server. The monitoring data is replayed by clients on nodes of the cluster to simulate the STS servers. For each job, two communications are realised. The first one creates an object on the LTS server to hold all data of one job and to transfer the meta data. This is done directly after starting the job and enables users to find the job's information. The second transmission is done after the job has finished. It sends all monitoring data in one package and updates the end time as well as the exit state of the job which is part of the meta data. During the two communications the job is in state "to handle".

The measurement of the jobs handled by the LTS server is shown in Fig. 3a. It shows a behaviour similar to the STS server (Fig. 2a). During the first 7.5 hours the number of handled jobs increases constantly. Afterwards the value is maximal for about one hour, then it starts to decrease.



(a) Number of jobs in processing (meta data received but no monitoring data) and number of jobs persistently stored (meta data and monitoring data received) over time.



(b) Number of received measurements per minute over time.

Figure 3: Measurement results for the LTS server

The number of jobs which could be handled simultaneously by an LTS server at the test conditions is 826. In Fig. 2a the maximum is slightly larger. This is based on the fact that during the first minutes of the experiment (while the Java execution environment still optimises the execution of the LTS server) some data transfers are aborted and redone later. The jobs with the delayed transfer are not counted for the measurement but they show that the server is under high load. A further increase of the number of jobs results in a crash of the GT4 components and data loss.

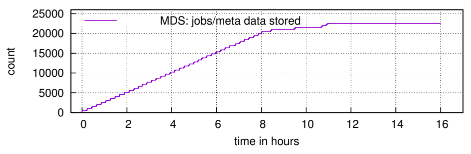
That the performance of an LTS exceeds the one of an STS server meets the expectations from the previous section and the theoretical analyse of SLAte in [HMP12a]. Based on the experiments we can also calculate the slow down factor[9] for the usage scenario $k_{N\_slow} = \frac{276}{826} = 0.33$ caused by using small packages to transfer the monitoring data.

Similar to the STS server measurements we visualise the transfer of the single measurements, even if they are sent as batches for each job of 112.5 kB each. Figure 3b shows that the data transfer starts after the first job has finished. A much higher transfer rate can be reached in comparison to the STS, but the fluctuation of the transfer rate is similar.

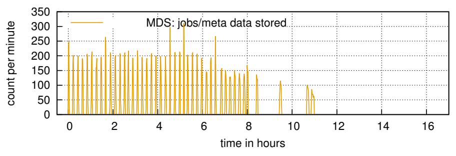
4.5 MDS Server

The MDS server only receives the meta data which are transferred as packages of 2.4 kB per job, monitoring data are not transferred. Thus a state like "to handle" is not present.

The number of jobs for which meta data are stored is shown by Fig. 4a. In the first eight hours the number is rising up to 20,470. This is similar to the behaviour of the LTS and STS server but the number of processed jobs is much higher. Afterwards the number of jobs should be constant, but the measurement show a slight increase. This is caused by transfers of meta data which got delayed based on the high load on the MDS server, but no data loss is observed. Again, an overload which leads to data loss is achieved by a further increase of the number of jobs.



(a) Number of transferred meta data packages (one per job) over time.



(b) Number of received meta data per minute over time.

Figure 4: Measurement for the MDS server

Comparing the MDS server to the STS server, the number of handled jobs is much higher with 20,470 compared to 276. The performance achieved for individual measurements (Fig. 2b) for the STS server is very similar to the one of the meta data on the MDS server. As seen at the other server types too, the transfer rate fluctuates quite strong (Fig. 4b) from zero transmissions to high values.

[9]The slow down factor was introduced in [HMP12a] and can be calculated as $k_{N\_slow} = \frac{N_{STS\_in}}{N_{LTS\_in}}$, with the realised network bandwidths N_{STS_in} for a STS server and N_{LTS_in} for the LTS server.

5 Scalability Observations

Scalability for the SLAte architecture is provided by the separation of servers (storage locations), repacking of data and the separation of monitoring and meta data. The separation allows to test individual servers of the STS and LTS layers, based on the fact that servers of one layer do not communicate with each other. They are only connected to a dedicated number of servers in the neighbouring layers. The repacking of the data of one job to one package gives a performance increase by a factor of more than three according to our measurements (276 jobs for a STS server to 826 jobs for a LTS server). This allows to connect one LTS server with three STS servers (using the hard and software used for the measurements).

As already shown in previous work [HMP12a] the MDS server is a bottleneck and defines the maximal size of an installation. Based on the data reduction which transfers only the meta data to the MDS server, the performance can be boosted to 20,470 jobs, which is a suitable number for monitoring a D-Grid like computing environment. For such a SLAte installation, an MDS server can be equipped with 25 LTS server and 75 STS servers. A single STS or LTS server is clearly not able to manage the monitor data of such a Grid infrastructure, this was already forecasted in [HMP10].

With a performance gain of a factor of 3 due to the repacking of the monitoring data and of a factor of 25 due to the distinction of monitoring and meta data (which allows to use multiple LTS servers for a installation), the overall performance increase is 75 compared to a centralised implementation with a performance similar to a STS server.

6 Conclusion and Future Work

This paper presents measurements to confirm the theoretical predication of our architectural concept described in [HMP12a]. We showed that the performance behaviour of the different layers allows to build up of a distributed monitoring infrastructure with a unified view on the data. The performance will be much higher than using a single storage location.

The presented experiments were not based on the most recent hardware. We even expect a higher performance when switching to more recent and more powerful hardware.

An interesting observation is the high fluctuation of the transfer rates. We have to analyse how to better coordinate the transfers to get a more constant and thus higher data rate. This also includes a discussion about replacing the underlying GT4-based web service we used for the current implementation for better performance.

We also observed the predicted bottleneck in the MDS server for storing data, which limits the growth of the infrastructure. In future developments we will address this shortcoming with respect to the scalable data access mechanism in a conceptional way. An additional task is to transfer the Grid implementation we have tested to Cluster and Cloud environments.

References

[ACC+10] J. Andreeva, M. Calloni, D. Colling, F. Fanzago, J. D'Hondt, J. Klem, G. Maier, J. Letts, J. Maes, S. Padhi, S. Sarkar, D.Spiga, P. Van Mulders, and I. Villella. CMS Analysis Operations. *Journal of Physics: Conference Series*, 219(7):072007, 2010.

[BBK+09] Timo Baur, Rebecca Breu, Tibor Klmn, Tobias Lindinger, Anne Milbert, Gevorg Poghosyan, Helmut Reiser, and Mathilde Romberg. An Interoperable Grid Information System for Integrated Resource Monitoring Based on Virtual Organizations. *Journal of Grid Computing*, 7:319–333, 2009. 10.1007/s10723-009-9134-3.

[BHJR10] Holger Brunst, Daniel Hackenberg, Guido Juckeland, and Heide Rohling. Comprehensive Performance Tracking with Vampir 7. In Matthias S. Müller, Michael M. Resch, Alexander Schulz, and Wolfgang E. Nagel, editors, *Tools for High Performance Computing 2009*, pages 17–29. Springer Berlin Heidelberg, 2010. 10.1007/978-3-642-11261-4_2.

[EGM+] Erik Elmroth, Peter Gardfjäll, Olle Mulmo, Åke Sandgren, and Thomas Sandholm. A Coordinated Accounting Solution for SweGrid Version: Draft 0.1.3 Date: October 7, 2003.

[Fos05] Ian Foster. Globus Toolkit Version 4: Software for Service-Oriented Systems. In Hai Jin, Daniel Reed, and Wenbin Jiang, editors, *Network and Parallel Computing*, volume 3779 of *Lecture Notes in Computer Science*, pages 2–13. Springer Berlin / Heidelberg, 2005.

[Hil14] Marcus Hilbrich. *Jobzentrisches Monitoring in verteilten heterogenen Umgebungen mit Hilfe innovativer skalierbarer Methoden*. Dissertation, Fakultät Informatik der Technischen Universität Dresden, Germany, September 2014.

[HMP10] Marcus Hilbrich and Ralph Müller-Pfefferkorn. A Scalable Infrastructure for Job-Centric Monitoring Data from Distributed Systems. In Marian Bubak, Michal Turala, and Kazimierz Wiatr, editors, *Proceedings Cracow Grid Workshop '09*, pages 120–125, ul. Nawojki 11, 30-950 Krakow 61, P.O. Box 386, Poland, February 2010. ACC CYFRONET AGH.

[HMP12a] Marcus Hilbrich and Ralph Müller-Pfefferkorn. Achieving scalability for job centric monitoring in a distributed infrastructure. In Gero Mühl, Jan Richling, and Andreas Herkersdorf, editors, *ARCS Workshops*, volume 200 of *LNI*, pages 481–492. GI, 2012.

[HMP12b] Marcus Hilbrich and Ralph Müller-Pfefferkorn. Identifying Limits of Scalability in Distributed, Heterogeneous, Layer Based Monitoring Concepts like SLAte. *Computer Science*, 13(3):23–33, 2012.

[HWT13] Marcus Hilbrich, Matthias Weber, and Ronny Tschüter. Automatic Analysis of Large Data Sets: A Walk-Through on Methods from Different Perspectives. In *Cloud Computing and Big Data (CloudCom-Asia), 2013 International Conference on*, pages 373–380, Dec 2013.

[MBB+12] Dieteran Mey, Scott Biersdorf, Christian Bischof, Kai Diethelm, Dominic Eschweiler, Michael Gerndt, Andreas Knüpfer, Daniel Lorenz, Allen Malony, Wolfgang E. Nagel, Yury Oleynik, Christian Rössel, Pavel Saviankou, Dirk Schmidl, Sameer Shende, Michael Wagner, Bert Wesarg, and Felix Wolf. Score-P: A Unified Performance Measurement System for Petascale Applications. In Christian Bischof, Heinz-Gerd Hegering, Wolfgang E. Nagel, and Gabriel Wittum, editors, *Competence in High Performance Computing 2010*, pages 85–97. Springer Berlin Heidelberg, 2012.

Automatic Sorting of Bulk Material

Georg Maier

Fraunhofer-Institut für Optronik, Systemtechnik und Bildauswertung IOSB
Abteilung Sichtprüfsysteme (SPR)
Fraunhoferstraße 1
76131 Karlsruhe, Germany

Das Geschäftsfeld Inspektion und Sichtprüfung des Fraunhofer IOSB beschäftigt sich u.a. mit der Entwicklung optischer Prüfsysteme für die industrielle Verarbeitung von Schüttgütern. Häufig beinhaltet die Aufgabenstellung eines solchen Systems, dass nach einem Sortiervorgang ein entsprechendes Produkt in möglichst reiner Form vorliegt, also Fremdkörper und Schlechtmaterial aussortiert werden. Ein Systemaufbau besteht i.d.R. aus einem Transportmedium, von welchem das entsprechende Produkt kurz vor der Inspektionslinie abgeworfen wird. Spezielle Beleuchtungen erleichtern die spätere Bildverarbeitung in Software. Ein PC empfängt die Bilddaten von einer Kamera. Durch die Ansteuerung von Luftdruckdüsen wird das Material nach der Auswertung getrennt. Durch die begrenzte Zeit, in welcher sich ein Objekt von der Inspektionslinie zur Ausblaslinie bewegt, entsteht ein Echtzeitkriterium. Der Kundenanspruch besteht darin, bei hohem Durchsatz präzise Objektklassifikationen durchzuführen. Sequentielle Bildverarbeitungsroutinen werden diesem Anspruch nicht mehr gerecht.

Die Abtastung des Bilds zur Erkennung von Objekten kann auch bei hohem Durchsatz sequentiell durchgeführt werden. Dies ermöglicht im Weiteren die parallele Verarbeitung einzelner Objekte. Zu den Berechnungen gehört beispielsweise die Trennung sich berührender Objekte, die Berechnung diverser Deskriptoren, sowie die Klassifikation. Obwohl auf diese Weise aufwendige Operationen parallel durchgeführt werden können, generiert die nebenläufige Verarbeitung einzelner Objekte einen großen Overhead, etwa durch das Starten von Threads. Durch das Erzeugen von Gruppen von Objekten kann durch ein Kriterium, wie beispielsweise die Anzahl der gepufferten Objekte, jedoch mehr Rechenaufwand auf einzelne Threads ausgelagert werden.

Dieser Ansatz bringt zwei Vorteile. Zum einen können durch die parallele Verarbeitung mehr Objekte verarbeitet werden. Zum anderen trägt der Ansatz auch zur Stabilität des Systems bei. Bei Verschmutzungen oder Fremdkörpern, für welche bestimmte Operationen z.B. aufgrund der Größe des vermeintlichen Objekts deutlich mehr Rechenaufwand im Vergleich zu dem erwarteten Produkt mit sich bringen, werden weniger andere Objekte ausgebremst, da nur jene Verarbeitungs-Threads betroffen sind, in denen sich ein entsprechendes Objekt zur Verarbeitung befindet.

Bei den beschriebenen Sortiersystemen kommt komplexe Software zum Einsatz. Somit ergibt sich die Herausforderung, bestehenden Code möglichst einfach zu parallelisieren. Durch den hier beschriebenen Ansatz konnte dies für einen der rechenintensivsten Prozesse der Software durchgeführt werden.

1. Aktuelle und zukünftige Aktivitäten (Bericht des Sprechers)

Die 32. Ausgabe der PARS-Mitteilungen enthält die Beiträge des 26. PARS-Workshops, der die wesentliche Aktivität der Fachgruppe im Jahr 2015 darstellt.

Der 26. PARS-Workshop fand am 7. und 8. Mai 2015 in Potsdam statt. Der Workshop war mit 30 Teilnehmern gut besucht. Am Morgen des ersten Tages präsentierte Professor Burkhart (Univ. Basel) ein Tutorial zum Thema „Reproducability in High Performance Computing". Bis zum Mittag des zweiten Tages folgten 13 Vorträge, ein eingeladener Vortrag und eine Industrie-Session mit 3 weiteren Vorträgen, die zusammen ein umfangreiches Themenspektrum abdeckten. Frau Professor Schnor (Univ. Potsdam) und ihrer Gruppe sei für die Organisation des Workshops gedankt.

Den zum zehnten Mal ausgeschriebenen und mit 500 € dotierten Nachwuchspreis erhielt in diesem Jahr Frau Jutta Fitzek (GSI Darmstadt). Herr Sunil Ramanarayanan (TU Berlin) erhielt einen Buchpreis. Die Übergabe des Preises erfolgte im Rahmen der Abendveranstaltung während einer Havel-Schiffsfahrt.

v.l.n.r.: Preisträgerin J. Fitzek, W. Karl (stellv. FG-Sprecher). Bild: S. Christgau

Während des PARS-Workshops fand auch eine Sitzung des PARS-Leitungsgremiums statt. Dort wurde ein neues Leitungsgremium für die Periode 2016 bis 2018 gewählt. Drei langjährige Mitglieder des Leitungsgremiums, die Herren Prof. Dr. H. Burkhart, Prof. Dr. H. Schmeck, und Prof. Dr. H. Weberpals, stellten sich nicht mehr zur Wahl. Der stellv. Fachgruppensprecher Prof. Karl dankte ihnen für ihr langjähriges Engagement. Neben den weiteren aktuellen Mitgliedern (s. Seite 2) stellten sich auch fünf „neue“ Personen zur Wahl: Herr Prof. Dr. Norbert Eicker (FZ Jülich), Herr Prof. Dr. Thomas Fahringer (U Innsbruck), Herr Prof. Dr. V. Heuveline (U Heidelberg), Herr Prof. Dr. Ben Juurlink (TU Berlin) und Frau Prof. Dr. B. Schnor (U Potsdam). Alle Kandidaten wurden ins Leitungsgremium gewählt. Herr Professor Karl (KIT) und Herr Professor Keller (FU Hagen) wurden erneut zu stellv. Sprecher und Sprecher der Fachgruppe gewählt.

Unser nächster Workshop ist der

12. PASA-Workshop vorauss. am 4. und 5. April 2016 in Nürnberg.

Der Workshop wird wie in den vergangenen „geraden“ Jahren gemeinsam mit der Fachgruppe ALGO im Rahmen der Tagung ARCS 2016 durchgeführt.

Aktuelle Informationen finden Sie auch auf der PARS-Webpage

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

Anregungen und Beiträge für die Mitteilungen können an den Sprecher (joerg.keller@FernUni-Hagen.de) gesendet werden.

Ich wünsche Ihnen einen guten Start ins Wintersemester und schon jetzt ein gesundes und erfolgreiches Jahr 2016.

Hagen, im September 2015
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

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 und 2014 in Lübeck (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)

4. Mitteilungen (ISSN 0177-0454)

Bisher sind 32 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.

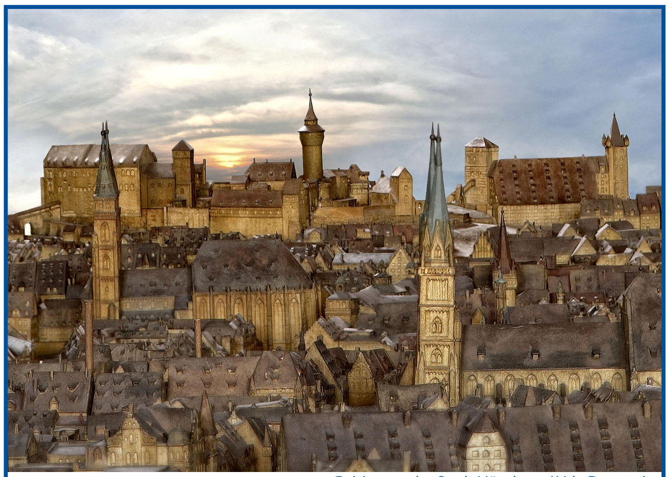
29th GI/ITG International Conference on
Architecture of Computing Systems
Nuremberg, Germany, April 4–7, 2016

ARCS 2016

Heterogeneity in Architectures and Systems – From Embedded to HPC

Call for Papers

The ARCS series of conferences has a long tradition reporting top notch results in computer architecture and operating systems research. The focus of the 2016 conference will be on **Heterogeneity in Architectures and Systems – From Embedded to HPC**. Like the previous conferences in this series, ARCS 2016 intends to be an important forum for computer architecture research. In 2016, ARCS will be organized by the Department of Computer Science at the Friedrich-Alexander University Erlangen-Nürnberg (FAU). ARCS was founded in 1970 by the German and European computer pioneer Prof. Wolfgang Händler who founded also the Computer Science Department at FAU in 1966. In 2016, the CS Department celebrates its 50th anniversary and it is a privilege to have ARCS back to its roots.
The ARCS 2016 proceedings will be published in the Springer Lecture Notes on Computer Science (LNCS) series. After the conference, it is planned that the authors of selected papers will be invited to submit an extended version of their contribution for publication in a special issue of Elsevier's Journal of Systems Architecture (JSA). In addition, the best paper and the best presentation will be awarded during the conference.


© Museen der Stadt Nürnberg / Udo Bernstein

Important Dates

Paper submission deadline:	**Oct. 26, 2015**
Workshop & tutorial proposals:	**Nov. 30, 2015**
Notification of acceptance:	**Dec. 21, 2015**
Camera-ready papers:	**Jan. 11, 2016**

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

- Architectures and design methods/tools for robust, fault-tolerant, real-time embedded systems
- Generic and application-specific accelerators in heterogeneous architectures, heterogeneous image systems
- Distributed computing architectures, high-performance computing (HPC), and cloud computing
- Cyber-physical systems, internet of things (IoT), Industrie 4.0, and their applications
- Multi-/manycore architectures, memory systems, and interconnection networks
- Programming models, runtime systems, middleware, verification, and tool support for manycore systems
- Operating systems including but not limited to scheduling, memory management, power management, and RTOS
- Adaptive system architectures such as reconfigurable systems in hardware and software
- Organic and autonomic computing including both theoretical and practical results on Self-X techniques
- Energy awareness and green computing
- 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
- Performance modeling and engineering
- Approximate computing

Submissions must be done through the link provided at the conference website. Papers have to be submitted in PDF format. They must be formatted according to Springer LNCS style (see: http://www.springer.de/comp/lncs/authors.html) and must not exceed 12 pages.

Workshop and Tutorial Proposals within the technical scope of the conference are solicited. Submissions should be sent by email directly to the Workshop and Tutorial Chair.

http://www3.cs.fau.de/arcs2016/

29th GI/ITG International Conference on
Architecture of Computing Systems
Nuremberg, Germany, April 4–7, 2016

Organizing Committee

General Co-Chairs

- Dietmar Fey, *Friedrich-Alexander University Erlangen-Nürnberg (FAU), Germany*
- Wolfgang Schröder-Preikschat, *Friedrich-Alexander University Erlangen-Nürnberg (FAU), Germany*
- Jürgen Teich, *Friedrich-Alexander University Erlangen-Nürnberg (FAU), Germany*

Program Co-Chairs

- Frank Hannig, *Friedrich-Alexander University Erlangen-Nürnberg (FAU), Germany*
- João M. P. Cardoso, *University of Porto, Portugal*

Workshop and Tutorial Chair

- Ana Lucia Varbanescu, *University of Amsterdam, The Netherlands*

Publication Chair

- Thilo Pionteck, *Universität zu Lübeck, Germany*

Program Committee

- Michael Beigl, *Karlsruhe Institute of Technology, Germany*
- Mladen Berekovic, *TU Braunschweig, Germany*
- Simon Bliudze, *EPFL, Switzerland*
- Florian Brandner, *ENSTA ParisTech, France*
- Jürgen Brehm, *University of Hannover, Germany*
- Uwe Brinkschulte, *Goethe University Frankfurt, Germany*
- Luigi Carro, *UFRGS, Brasil*
- Albert Cohen, *INRIA, France*
- Nikitas Dimopoulos, *University of Victoria, Canada*
- Ahmed El-Mahdy, *E-JUST, Egypt*
- Fabrizio Ferrandi, *Politecnico di Milano, Italy*
- Dietmar Fey, *FAU, Germany*
- Björn Franke, *University of Edinburgh, UK*
- Roberto Giorgi, *University of Siena, Italy*
- Daniel Gracia Pérez, *Thales Research & Technology, France*
- Jan Haase, *Helmut Schmidt University, Germany*
- Jörg Hähner, *Augsburg University, Germany*
- Jörg Henkel, *Karlsruhe Institute of Technology, Germany*
- Andreas Herkersdorf, *TU München, Germany*
- Christian Hochberger, *TU Darmstadt, Germany*
- Michael Hübner, *Ruhr-Universität Bochum, Germany*
- Gert Jervan, *Tallinn University of Technology, Estonia*
- Ben Juurlink, *TU Berlin, Germany*
- Wolfgang Karl, *Karlsruhe Institute of Technology, Germany*
- Christos Kartsaklis, *Oak Ridge National Laboratory, USA*
- Jörg Keller, *Fernuniversität Hagen, Germany*
- Raimund Kirner, *University of Hertfordshire, UK*
- Andreas Koch, *TU Darmstadt, Germany*
- Hana Kubátová, *FIT CTU, Prague, Czech Republic*
- Olaf Landsiedel, *Chalmers Univ. of Technology, Sweden*
- Paul Lukowicz, *University of Passau, Germany*
- Erik Maehle, *Universität zu Lübeck, Germany*
- Christian Müller-Schloer, *University of Hannover, Germany*
- Alex Orailoglu, *UC San Diego, USA*
- Carlos Eduardo Pereira, *UFRGS, Brazil*
- Luis Pinho, *CISTER, ISEP, Portugal*
- Thilo Pionteck, *Universität zu Lübeck, Germany*
- Pascal Sainrat, *IRIT – Université de Toulouse, France*
- Toshinori Sato, *Fukuoka University, Japan*
- Wolfgang Schröder-Preikschat, *FAU, Germany*
- Martin Schulz, *Lawerence Livermore National Lab., USA*
- 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 München, Germany*
- Djamshid Tavangarian, *Rostock University, Germany*
- Jürgen Teich, *FAU, Germany*
- Martin Törngren, *KTH, Sweden*
- Eduardo Tovar, *ISEP-IPP, Portugal*
- Pedro Trancoso, *University of Cyprus, Cyprus*
- Carsten Trinitis, *TU München, Germany*
- Sascha Uhrig, *Airbus, Germany*
- Theo Ungerer, *University of Augsburg, Germany*
- Hans Vandierendonck, *Queen's University Belfast, UK*
- Stephane Vialle, *SUPELEC, France*
- Lucian Vintan, *"Lucian Blaga" University of Sibiu, Romania*
- Klaus Waldschmidt, *Goethe University Frankfurt, Germany*

http://www3.cs.fau.de/arcs2016/

Preliminary 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

5th 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: http://www.easychair.org/pasa2016/
28th January 2016: Notification of the authors
10th 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

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

Prof. Dr. Helmar Burkhart, Univ. Basel
Dr. Andreas Döring, IBM Zürich
Prof. Dr. Dietmar Fey, Univ. Erlangen
Prof. Dr. Wolfgang Karl, stellv. Sprecher, KIT
Prof. Dr. Jörg Keller, Sprecher, FernUniversität 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
Prof. Dr. Hartmut Schmeck, KIT
Prof. Dr. Peter Sobe, HTW Dresden
Prof. Dr. Theo Ungerer, Univ. Augsburg
Prof. Dr. Rolf Wanka, Univ. Erlangen
Prof. Dr. Helmut Weberpals, TU Hamburg-Harburg

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/