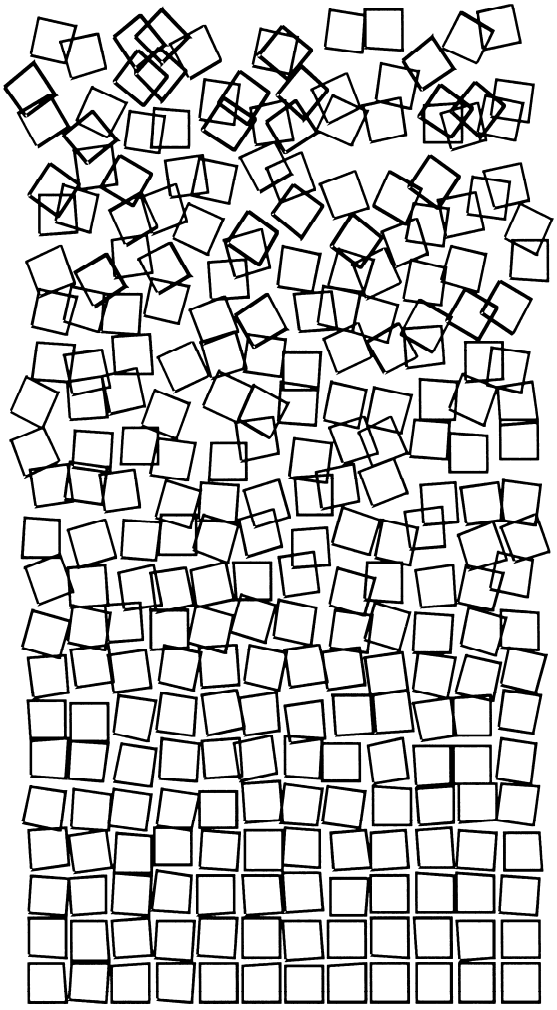
GESELLSCHAFT FÜR INFORMATIK E.V.

PARALLEL-ALGORITHMEN, -RECHNERSTRUKTUREN UND -SYSTEMSOFTWARE

PARS

INFORMATIONSTECHNISCHE GESELLSCHAFT IM VDE



Computergraphik von: Georg Nees, Generative Computergraphik

I n h a l t

Aktuelle PARS-Aktivitäten unter:

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

ISSN 0177-0454

PARS-Mitteilungen

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

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

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

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

Die Homepage

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

vermittelt aktuelle Informationen über PARS.

CALL FOR PAPERS

11th Workshop on Parallel Systems and Algorithms
PASA 2014

in conjunction with

International Conference on Architecture of Computing Systems (ARCS 2014)
Luebeck, Germany, February 25-28, 2014

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, grid and cloud 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*
- *parallel algorithms for big data*

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

28th October 2013: Deadline for submission of papers of about 10 pages (in English, using LNI style, see http://www.gi.de/service/publikationen/lni.html) under: http://www.easychair.org/conferences/?conf=pasa2014
25th November 2013: Notification of the authors
10th December 2013: Final version for workshop proceedings

Program Committee

S. Albers (Berlin), H. Burkhart (Basel), M. Dietzfelbinger (Ilmenau), A. Döring (Zürich), D. Fey (Erlangen)
W. Heenes (Darmstadt), R. Hoffmann (Darmstadt), J. Hromkovic (Zürich), K. Jansen (Kiel), W. Karl (Karlsruhe)
J. Keller (Hagen), Ch. Lengauer (Passau), E. Maehle (Lübeck), E. W. Mayr (München), U. Meyer (Frankfurt)
F. Meyer auf der Heide (Paderborn), W. Nagel (Dresden), M. Philippsen (Erlangen), K. D. Reinartz (Höchstadt)
P. Sanders (Karlsruhe), Ch. Scheideler (Paderborn), H. Schmeck (Karlsruhe), U. Schwiegelshohn (Dortmund)
P. Sobe (Dresden), T. Ungerer (Augsburg), B. Vöcking (Aachen), R. Wanka (Erlangen)
H. Weberpals (Hamburg-Harburg)

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 +49-9131-85-25152/25149, E-Mail rwanka at cs.fau.de

11. PASA-Workshop (Full Papers)

Hybrid parallelization of a seeded region growing segmentation of brain images for a GPU cluster

Anna M. Westhoff*†
*Simulation Lab Neuroscience - Bernstein Facility for Simulation and Database Technology
Institute for Advanced Simulation
Jülich Aachen Research Alliance
Forschungszentrum Jülich
Jülich, Germany
†Postal address: Forschungszentrum Jülich GmbH
Jülich Supercomputing Centre
52425 Jülich, Germany
Email: a.westhoff@fz-juelich.de

***Abstract*—The introduction of novel imaging technologies always carries new challenges regarding the processing of the captured images. Polarized Light Imaging (PLI) is such a new technique. It enables the mapping of single nerve fibers in postmortem human brains in unprecedented detail. Due to the very high resolution at sub-millimeter scale, an immense amount of image data has to be reconstructed three-dimensionally before it can be analyzed. Some of the steps in the reconstruction pipeline require a previous segmentation of the large images. This task of image processing creates black-and-white masks indicating the object and background pixels of the original images. It has turned out that a seeded region growing approach achieves segmentation masks of the desired quality. To be able to process the immense number of images acquired with PLI, the region growing has to be parallelized for a supercomputer. However, the choice of the seeds has to be automated in order to enable a parallel execution. A hybrid parallelization has been applied to the automated seeded region growing to exploit the architecture of a GPU cluster. The hybridity consists of an MPI parallelization and the execution of some well-chosen, data-parallel subtasks on GPUs. This approach achieves a linear speedup behavior so that the runtime can be reduced to a reasonable amount.**

I. Introduction

To understand the function of the human brain, it is necessary to study also its structure. An evolving imaging technique is Polarized Light Imaging [1], [2], [3]. It is applied to sections (slices) of postmortem human brain tissue and allows to analyze the course of nerve fibers between different brain regions at sub-millimeter scale.

Before the PLI data can be analyzed, some image processing steps have to be applied beforehand. The major step is a three-dimensional reconstruction of the stack of sections. An important prerequisite of this task is a prior segmentation of the images identifying the brain and non-brain regions. The main challenge of segmenting the PLI images is their immense number and the size of each image as there are terabytes of data per human brain which makes a parallel approach indispensable.

Since segmentation is a task of image processing which is required by a lot of use cases, a bunch of different approaches already exists such as thresholding, (seeded) region growing, neural networks, level sets, classification based methods or graph cuts. Although some parallelizations of segmentation approaches have been published, e.g. [4], [5], [6], [7], [8], [9], [10], [11], [12], [13], they cannot be adopted without modifications for the PLI data because all algorithms are optimized for the present image characteristics like color mode, contrast or textures. A large variety of algorithm classes is also observable from region growing and level sets to neural networks, graph based algorithms and random walks. Some of them suffer from over- or under-segmentation in these cases where it is a typical problem of the algorithms.

The authors use different parallelization approaches based on shared or distributed memory, some also use GPU(s). Most approaches distribute parts of an image between the calculation units, so the threads or tasks. In the publications [12] and [13], the solver is parallelized instead, i.e. the algorithm as such. Depending on the chosen type of parallelization, the achieved speedups vary from a weak improvement compared to the sequential implementation up to a linear speedup behavior.

Since none of the mentioned approaches can be used for the PLI data without further enhancements and the variety on all levels of yet parallelized algorithms seems to be large, another procedure has been chosen to find an appropriate, parallelizable segmentation algorithm. Existing sequential segmentation tools have been applied to a representative small subset of the PLI images. The one achieving the best results, thus the best black-and-white masks, has then been adopted and parallelized.

This paper focuses on the hybrid parallelization of a segmentation in form of a seeded region growing. It describes the parallelization for the GPU cluster JUDGE (**Ju**elich **D**edicated **G**PU **E**nvironment) hosted by Jülich Supercomputing Centre (JSC), Forschungszentrum Jülich. Since a parallelization does not make sense without a previous automation of the choice of seeds, this aspect is also briefly discussed.

II. Material & Methods

A. Human Brain Data

The human brain tissue measured with PLI is from body donor programs of German universities. The measurement is

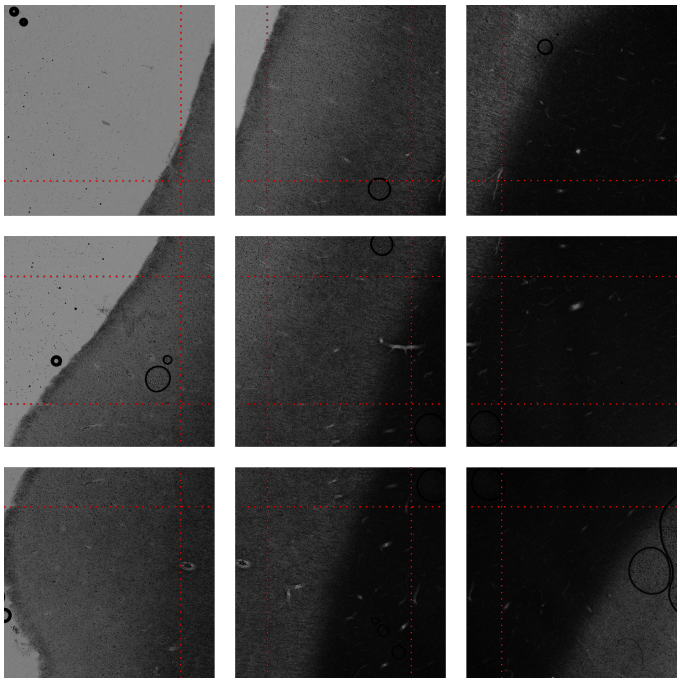


Fig. 1. These PLI image tiles demonstrate the overlapping of neighboring tiles. About 30% of the image information of a tile is also contained in the neighboring ones.

done according to [3] using a Polarizing Microscope developed by Taorad GmbH, Aachen, Germany. This technique takes advantage of the myelin sheaths that envelope the axons of nerve fibers. Myelin exhibits the optical property referred to as birefringence. This reaction to the incident polarized light is used to extract the nerve fiber orientations.

To image a post-mortem brain with PLI, it is cut into sections with a thickness of $70\mu m$. A section is imaged 18 times under linearly polarized light whereby the orientation of the polarized light is rotated by ten degree for every shot. In order to reach a high resolution, a section is not captured once for each light configuration but several times. This way a mosaic of image tiles is captured for each section. The tiles have a resolution of $1.6\mu m \times 1.6\mu m$ per pixel and an image size of 2048×2048 pixel. Neighboring tiles are imaged with a large overlapping as it is illustrated in figure 1. The outer edges of a section are only included in one tile but these edges do not show brain tissue. In total, there are at minimum 1500 sections per human brain, each consisting of about 25×30 tiles. This results in at minimum $1500 \cdot 25 \cdot 30 = 1,125,000$ images per human brain that have to be segmented.

B. Basic Seeded Region Growing Algorithm

The seeded region growing algorithm presented in [14] produces good and reasonable masks if applied to PLI images. The quality of this algorithm can also be guessed since it has been re-used in several other cases like [15], [16]. The algorithm is known as the first published region growing using seeds which has a significant effect concerning the quality of the segmentation mask because the number of classes in the mask can be directly controlled. In this way over- and under-segmentation can be avoided.

The algorithm is originally designed for intensity, i.e. gray-value, images. It divides the image into n classes $A_1, \ldots, A_n$ which contain the seeds in the beginning. Afterward, the algorithm assigns all remaining pixels iteratively to one of the $A_i, i \in [1,n]$. In each iteration, one pixel is assigned to one of the sets A_i. Therefore, the set T of all non-labeled pixels which directly border on the already labeled regions is defined as in equation (1). It contains all pixels which have not yet been added to any of the A_i but which are directly adjacent to one of the A_i. $N(x)$ is the set of all direct neighbors of a pixel at the one-dimensional position x.

$$T = \left\{ x \notin \bigcup_{i=1}^{n} A_i \mid N(x) \cap \bigcup_{i=1}^{n} A_i \neq \varnothing \right\} \tag{1}$$

If pixel $x \in T$ borders on exactly one of the A_i, let $i(x)$ be the index for which $N(x) \cap A_{i(x)} \neq \varnothing$. Otherwise, if x is neighbor of two or more of the A_i, $i(x)$ is defined as the index for which $N(x) \cap A_{i(x)} \neq \varnothing$ and a measure $\delta(x)$ is minimized. This measure $\delta(x)$ defines how similar a pixel x is compared to the region it adjoins. It may be defined as follows with $g(x)$ being the gray value of pixel x:

$$\delta(x) = \left| g(x) - \underset{y \in A_{i(x)}}{\text{mean}} \left[g(y) \right] \right| \tag{2}$$

Alternatively, pixels adjacent to two or more of the A_i can be assigned to a new set B containing all boundary pixels. In the end of each iteration, the pixel $z \in T$ with

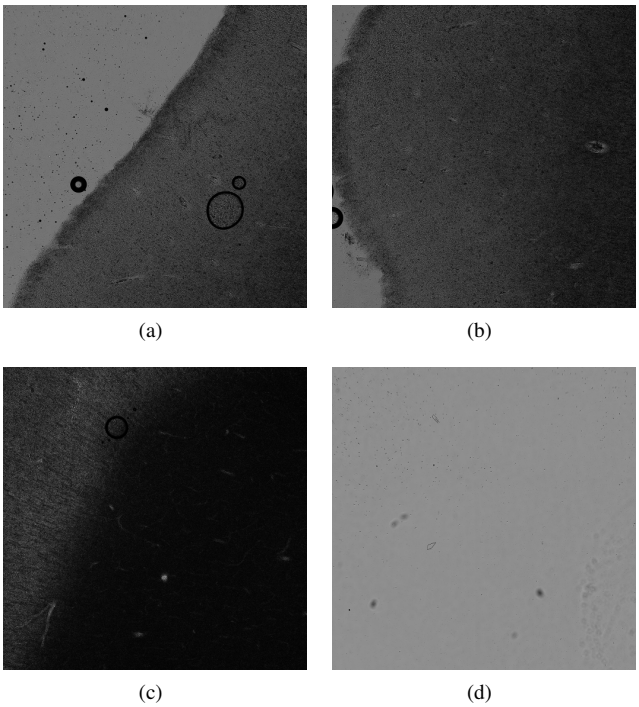
$$\delta(z) = \min_{x \in T} \{ \delta(x) \} \tag{3}$$

is labeled corresponding to $A_{i(z)}$ and appended to this set. The definitions (2) and (3) assure that the regions A_i are as homogeneous as possible and, by the use of $N(x)$, that each of the regions is connected.

C. Automating the Choice of Seeds

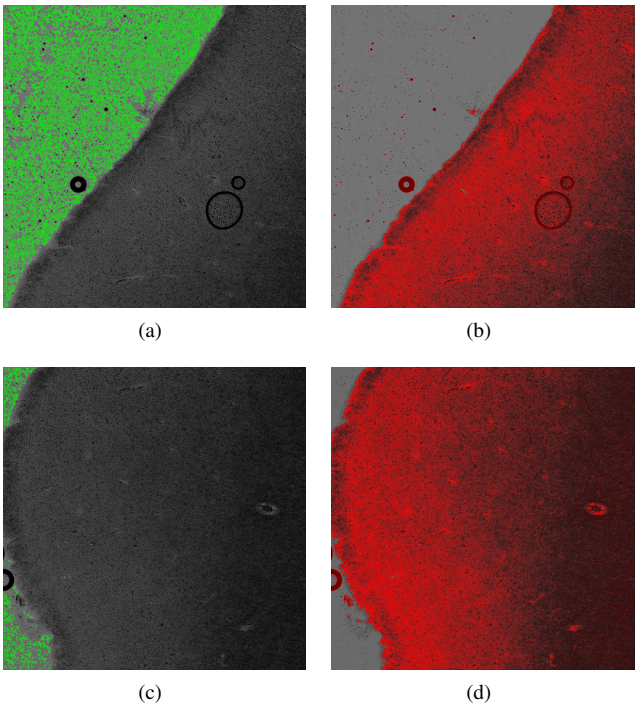
The first major challenge in the development process of a segmentation for microscopic images acquired with PLI was the adoption to the immense number of images. The choice of the seeds had to be brought into focus. A definition by hand results in an unacceptable effort because at least one seed for each of the hundreds of thousands of images per brain has to be chosen. Hence, an automation of this step was needed to get away from an interactive processing of the images and thus enable a parallelization. This step has been done based on high level knowledge of the images.

It has been decided to use the so-called transmittance of the 18 different shots of the same brain region for the segmentation because this modality allows a clear distinction between brain tissue and background. Automating the choice of seeds step for the PLI data, it had to be kept in mind that there are tiles showing both brain tissue and background but that some only show one of the two classes as it is visible in figure 2. It is observable that the brain is in principle brighter than the rest of the image. Therefore it was possible to utilize an intensity histogram based choice of seeds. Taking the different image contents into account, it was not reasonable to use separate histograms of the different tiles because they might not contain information about both brain and background



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

Fig. 2. These four images are representative examples for PLI microscopic tiles. 2(a) and 2(b) show both the dark brain with a large intensity variance on a brighter background. 2(c) contains only brain tissue and 2(d) only background.



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

Fig. 3. These images demonstrate the measure m_{cand} for two of the four examples. The original images are visible in the background. The measure values are illustrated as overlays; the background seeds are green, brain seeds are red. Green resp. red pixels correspond to measure value 0, i.e. reliable candidate seeds. For measure values in $]0,1]$, a gradient from green resp. red to transparent is used. All pixels with measure values larger than 1 have a transparent color in the overlay.

intensities. Instead, the joint histogram of all available mosaic tiles of all sections has been calculated.

The user has to define a rough threshold differentiating between brain and background intensities based on the joint intensity histogram. The effort of this manual step is independent of the number of images to be processed, so setting one value per brain. So the segmentation has been split into two independent tools, the first one calculates the joint intensity histogram, the second one performs the region growing as follows. In between, the manual choice of the threshold takes place.

The user-defined threshold divides the intensities into two intervals. For both intervals within the histogram, the median intensity $q_{0.5}$ and the quantiles q_α and $q_{1-\alpha}$ have been calculated. Based on this information, a measure m_{cand} has been calculated that defines how well suited a pixel is to be used as a seed for the respective class. (x,y) denotes the two-dimensional coordinates of a pixel and $g(x,y)$ the intensity of this pixel.

$$m_{cand}(x,y) = \begin{cases} \frac{g(x,y)-q_{0.5}}{q_{0.5}-q_\alpha}, & g(x,y) \leq q_{0.5} \\ \frac{q_{0.5}-g(x,y)}{q_{1-\alpha}-q_{0.5}}, & g(x,y) > q_{0.5} \end{cases} \tag{4}$$

$$= \max\left(\frac{g(x,y)-q_{0.5}}{q_{0.5}-q_\alpha}, \frac{q_{0.5}-g(x,y)}{q_{1-\alpha}-q_{0.5}}\right) \tag{5}$$

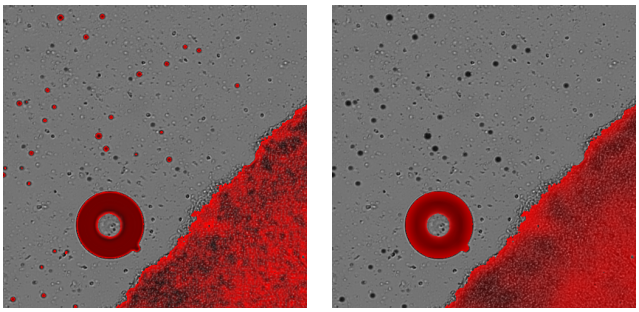
This measure can be computed independently for each pixel and is stored in a measure image. With this definition, every pixel is a candidate seed given that $m_{cand}(x,y) \leq 1$. The candidates with the smallest values are the comparably best seeds. Pixel with intensities similar to the threshold are not seeds for any of the regions because the intensity ranges of brain and background overlap.

In fact, there are two measure images, one based on the brain and one based on the background intensity interval. If candidate seeds would be chosen using these measures m_{cand}, also noise pixels were seed candidates. Image noise denotes pixels with a variation in intensity or color compared to the neighboring ones that bears no reference to the captured object and which mostly appears as isolated mis-colored pixels. Therefore, also spatial information has been included in the definition of the measure δ in form of a linear smoothing filter.

The calculation of the initial measure and the following smoothing operation could be combined to a single discrete convolution, i.e. a linear image filter. The final seed measure images m_{final} have been computed using convolution equation (6).

$$m_{final}(x,y) = w(x,y) * m_{cand}(x,y)$$
$$= \sum_{i=-m}^{m}\sum_{k=-n}^{n} w(i,k) \cdot m_{cand}(x+i, y+k) \tag{6}$$

It turned out that a weighted averaging smoothing filter w works best for the PLI tiles, if for a central value p all other entries have the same value $\frac{1-p}{(2m+1)\cdot(2n+1)-1}$. The central weight has been chosen to $p = 0.1$ and $m = n = 4$. Figure 3 illustrates the final measure images m_{final} for brain and



(a) (b)

Fig. 4. These images demonstrate the elimination of small isles of brain seeds. The side effect of this step is a smoothing of the remaining measure values.

background seeds of the example tiles showing both brain and background.

In this figure, it is observable that some of the dirt particles in the background region of the image tiles are erroneously labeled as seeds for the brain region. Since the dirt particles result in isles of seeds that are much smaller than any brain region, they could be erased out of m_{final}:

For all brain seeds, the number N of brain seeds within a square neighbor region with a size of $(\text{radius} \cdot 2 + 1)^2$ has been determined. Furthermore, the sum S of the inverted measure values of these neighboring seeds has been computed. ($\mathbb{1}$ denotes the indicator function.)

$$S(x,y) = \sum_{i,k=-\text{radius}}^{\text{radius}} \{(1.0 - m_{final}(x+i, y+k)) \cdot \mathbb{1}_{\{x \leq 1\}}(m_{final}(x+i, y+k))\} \quad (7)$$

Using N and S as in equation (8), seeds are rated higher, i.e. they are more reliable, if they have other comparably good seeds in their surroundings. Otherwise, they are downgraded as in case of the seeds isles caused by the dirt particles in the background. The effect of the elimination of small brain seed isles is illustrated in figure 4.

$$\hat{m}_{final}(x,y) = \begin{cases} m_1, & \frac{N}{(\text{radius} \cdot 2+1)^2} \geq \text{threshold} \\ m_2, & \frac{N}{(\text{radius} \cdot 2+1)^2} < \text{threshold} \end{cases}, \quad (8)$$

$$m_1 = 1.0 - \frac{S(x,y)}{(\text{radius} \cdot 2 + 1)^2},$$

$$m_2 = 1.0 + (1.0 - \text{threshold}) \cdot \frac{S(x,y)}{(\text{radius} \cdot 2 + 1)^2}$$

A deeper explanation and analysis of the automated choice of seeds can be found in the Master's thesis [17].

D. Parallelization for a GPU cluster

As it has been mentioned above, the enormous number of PLI microscopic tiles can only be handled if the segmentation

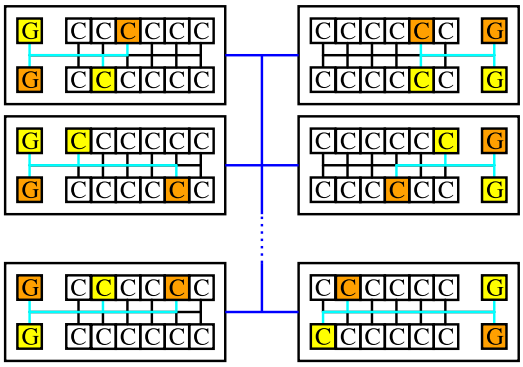


Fig. 5. Schematic illustration of the architecture of JUDGE. Each node consists of two NVIDIA GPUs and two 6-core Intel Xeon processors. For the present application, two pairs of CPUs and GPUs (marked in yellow and orange) are used per node. This one-to-one assignment is used because there are some parts which are not ported to the GPUs but which cause a significant load on the CPUs. So the most suitable assignment is one GPU to one CPU. The CPUs may communicate using the network (blue lines). The remaining ten CPUs stay idle and may be used by other applications.

software is parallelized. JSC hosts among other supercomputers the GPU cluster JUDGE. Each of its 206 compute nodes consists of two Intel Xeon X5650 (Westmere) six-core processors and two NVIDIA GPUs as illustrated in figure 5. 54 of the nodes contain Tesla M2050 (Fermi) GPUs, the others Tesla M2070 (Fermi). These two GPUs only differ regarding the available memory which is with 3GB respectively 6GB sufficient for the present application. There are 96GB of main memory per node available. The network is an InfiniBand QDR HBA.

To exploit this architecture, the parallelization has been done on two levels. A multi-core approach using distributed memory has been combined with a transfer of some algorithmic steps on the GPUs. The two levels can be used in combination or separately so that the software can be ported to supercomputers with a different architecture without much effort.

1) Multi-core Approach: Since neighboring tiles of a section are captured with a sufficiently large overlapping, it is possible to process all tiles independently of the others because all required information is already present in the respective tile.

To port the seeded region growing segmentation to a multi-core platform, the tiles of a section have been cyclically distributed between all available processes. This distribution has been used for the computation of the joint histogram as well as for the seeded region growing. In case of the joint histogram, every process first calculates the joint histogram of the tiles assigned to it. Thus, each process has got a part of the global joint histogram that covers the whole intensity range but only a part of the tiles. Afterward, these distributed histograms are collected by a master process using the Message Passing Interface (MPI) and combined to the global joint histogram using the `MPI_Reduce` method with `MPI_SUM` as the reduction operation.

The seeded region growing of a tile can be done completely independent of the other tiles so that no MPI communication is required. This is possible because the overlapping region of

TABLE I. PROPORTION OF RUNTIME REQUIRED BY THE DIFFERENT STEPS OF THE SEGMENTATION.

Step of the Algorithm	Proportion of Runtime
Calculation of the measure image m_{final}	48.46%
Elimination of the seed isles out of m_{final}	48.21%
Labeling of the seeds in the masks	0.11%
Seeded region growing	3.21%

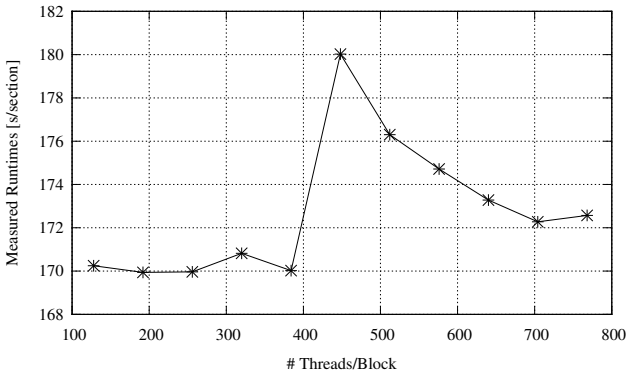


Fig. 6. The runtime of all GPU-parallelized steps of the seeded region growing executing the tool with thread block sizes from 128 to 768 in steps of 64. The shortest runtime is reached using block sizes of 192, 256 and 384 threads. The jump from 384 to 448 threads can be explained by analyzing the achieved occupancy of the device. For up to 384 threads per block, four blocks can be executed simultaneously on the same multiprocessor of the GPU. For 448 and more threads, only three blocks are possible. This reduces the achieved occupancy of the device drastically, i.e. the runtime is increased. For larger numbers of threads per block, the enlarging of the block size fills the multiprocessor again with threads so that the occupancy grows.

two neighboring tiles is much larger than all used neighborhood sizes. Furthermore, the seeded region growing algorithm is stable in respect of the choice of seeds, i.e. a moderate change of the seeds does not result in a different segmentation mask.

2) GPU Approach: For the second level of parallelism, the required runtime per section of the different region growing steps has been analyzed in a sequential execution. The measured proportions of runtime of the different main steps on the total runtime are listed in table I. The two steps required to obtain the seeds for the region growing consume together 96.67% of the total runtime, i.e. the calculation of m_{final} and the elimination of seed isles. Therefore, it has been worthwhile to revise these steps. The runtime of the calculation of the joint intensity histogram does not have to be considered with respect to a GPU parallelization because it consumes only 0.6% of the runtime per section compared to the in table I presented joint runtime of seed determination and region growing.

An analysis of the algorithms of these two steps revealed that both are data parallel operations, i.e. the operations can be computed independently for each pixel. This can be directly extracted from the definitions (6) and (8) which refer only to a single pixel (x, y). Nevertheless, information of a defined amount of neighboring pixels is required to compute these measures.

GPUs are processors that can exploit data parallelism since they have thousands of parallel processing units designed to execute the same instruction on thousands of pixels in the same processor cycle. So they are fitting to the two steps of the

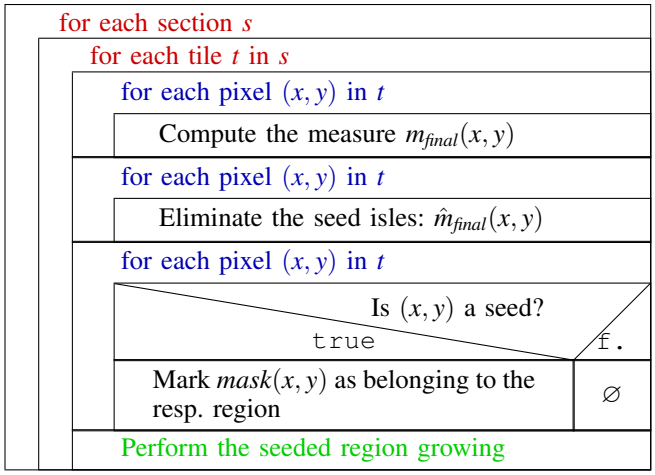


Fig. 7. This figure demonstrates how the two levels of the hybrid parallelization of the seeded region growing work together. The red loops are parallelized using the multi-core approach with MPI, the blue ones are executed on the GPUs using CUDA.

algorithm mentioned above. Thus, for each of the available CPUs, an additional GPU has been used so that a one-to-one assignment has been achieved as it is illustrated in figure 5 by the yellow and orange colored processing units.

For a re-implementation of these two steps on a GPU, the framework CUDA, version 4.1, has been used. Since the pixels of the images can be processed independently, a one-to-one assignment of pixels to CUDA threads has been applied. It has been decided to use one-dimensional thread blocks in form of rows because this shape exploits the data locality of the operations best: The images are stored in C order, so row-wise. For the processing of two neighboring pixels, the required other pixels are nearly the same which implies this one-dimensional shape for a thread block.

In order to find out the optimal size of the thread blocks, the region growing has been executed with thread block sizes varying from 128 to 768 in steps of 64 threads per block. The results are illustrated in figure 6. It figured out that the optimal thread block size is at 256 threads per block and thus an amount of 16384 thread blocks for the standard tile size of PLI.

3) Hybrid Parallelization: Figure 7 demonstrates how the two levels of the hybrid parallelization are combined. The seeded region growing contains outer loops iterating over all available image tiles, i.e. over all sections and all tiles within a section. These loops marked with the red color have been parallelized in the multi-core approach. Using the section number and the tile coordinates within the section, a consecutive index for every tile has been computed. Iterating over this index idx using P processes, a process P_i gets every Pth tile.

Inside these outer loops iterating over all indexes, the seeded region growing is performed tile by tile. This segmentation starts with three blue marked loops iterating over all pixels of the respective tile. Inside these loops, the final measure m_{final} is computed, the brain seed isles are eliminated and the obtained seeds are marked in the segmentation mask. Since all three operations are data-parallel, they have been ported to the

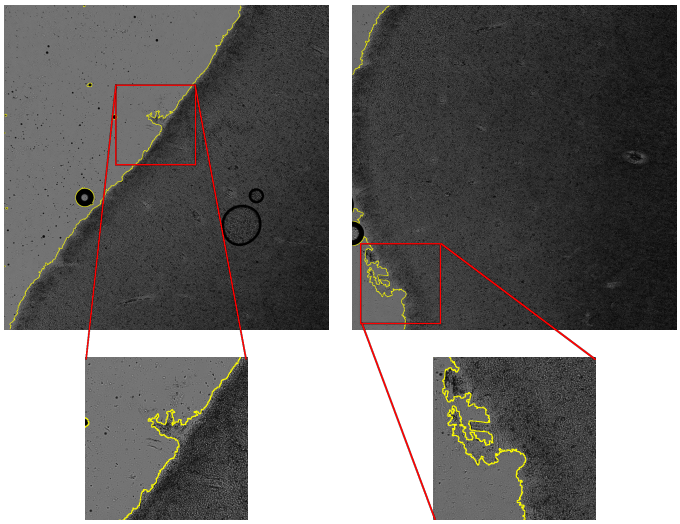


Fig. 8. The segmentation result is illustrated as edges between the classes within the masks. The detail images demonstrate that the edges follow the real outer edges of the brain tissue. The extracted regions are connected and do not contain holes.

GPU(s) using CUDA.

This schematic picture of the algorithm structure demonstrates not only how the two levels of the hybrid parallelization can be used in combination, but also that they can be omitted independent of each other. It has to be noted that the only non-parallelized part of the algorithm is the green marked line which is the seeded region growing.

III. Results

A. Segmentation Masks

The first important step to evaluate the results of the developed region growing is to have a closer look at the segmentation masks. If the masks are considered separately from the original images, it is difficult to see if the extracted mask regions fit the actual regions. A good alternative is to display the original image together with the edges between the mask regions drawn in as thin lines as it has been done for figure 8.

In this illustration, it is visible that the brain tissue is identified very well by the presented enhancement of the seeded region growing. The segmentation masks present the desired quality which is only possible because the developed automated choice of seeds works correctly. In particular, no seed for one of the regions is placed into the other region. The dirt particles in the background are ignored during the seed determination. The black rings visible in figure 8 are air bubbles which occur during the preparation of the brain tissue. They are always added to the brain region since they may occur in both the brain and the background region of the image and this assignment is important for other steps in the reconstruction pipeline.

B. Runtime and Speedup

To evaluate the hybrid parallelization of the seeded region growing, the runtime respectively speedup behavior of both

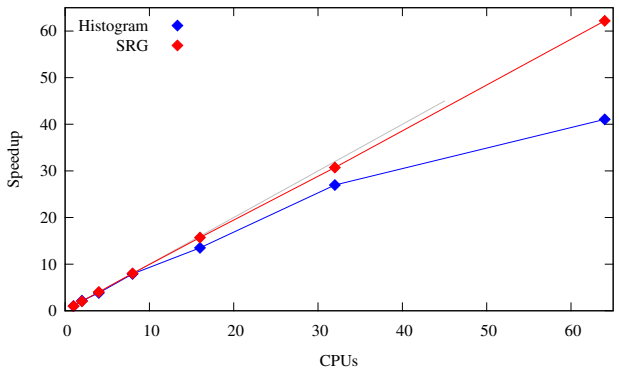


Fig. 9. These speedup curves per section have been measured on JUDGE with up to 64 processes. The blue curve belongs to the calculation of the joint intensity histogram of all image tiles. It flattens out as a result of the collection of the distributed computed histograms. The red curve shows the speedup behavior of the region growing part of the segmentation. Since there is no inter-process communication needed, an optimal, linear speedup is reached.

TABLE II. Proportion of runtime required by the different steps of the segmentation with and without the GPU parallelization.

Step of the Algorithm	**Proportion of Runtime**	
	sequential	***GPU parallel***
Calculation of the measure image m_{final}	48.46%	9.87%
Elimination of the seed isles out of m_{final}	48.21%	5.58%
Labeling of the seeds in the masks	0.11%	3.31%
Seeded region growing	3.21%	81.24%

levels has to be analyzed. Figure 9 shows the speedup curves of the multi-core approach. In particular, there is a speedup curve of the calculation of the joint histogram and another one of the region growing itself. These two parts of the segmentation tool are interrupted by the user input of the intensity threshold.

The speedup curve of the calculation of the joint intensity histogram is linear for small numbers of processes but flattens out for larger ones. This is justifiable by the increasing effort for the inter-process communication to collect the distributed histograms. In case of the seeded region growing, an optimal and thus linear speedup is reached since no communication is needed.

To evaluate the effect of the GPU parallelization approach, the runtime required by the different steps of the algorithm has once again been measured as it is visible in table II. The summed up proportions of runtime of the seed determination steps is reduced from 96.67% to 15.45% using the CUDA version instead of the sequential one. This results in an increase of the proportion of the (not modified) region growing from 3.21% to 81.24% so that the runtime behavior is dominated by the effort for the actual growing process. This is also the reason for the one-to-one assignment of GPUs and CPUs. Figure 10 illustrates the influence of the CUDA parallelization on the runtime behavior of the segmentation.

We have executed the segmentation using on the one hand up to 64 CPUs and on the other hand up to 64 pairs of CPUs and GPUs. The total runtime per section of the region growing has been measured. It is observable that the trend of the runtime curve is for both cases the same. The additional use of a GPU per process results in a shift down of the runtime curve, i.e. the region growing is accelerated by a factor of about 20.

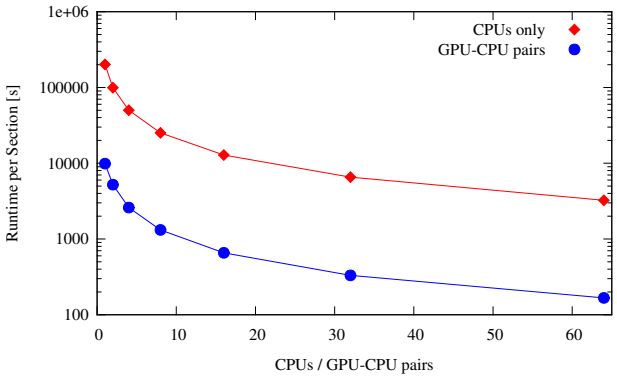


Fig. 10. This diagram compares the scaling behavior of the multi-core approach with and without the additional use of the GPU parallelization, i.e. up to 64 processes respectively 64 pairs of CPUs and GPUs are used. It is observable that the trend of the runtime per section is the same, the use of CUDA results in a shift down of the original curve. The segmentation is accelerated by a factor of 20 due to the additional use of a GPU per process.

IV. Discussion

This paper focuses on the development and hybrid parallelization of a segmentation for PLI image tiles. The additionally presented automation of the choice of seeds was necessary because of the immense number of images to be processed in case of PLI and to make a parallel execution possible. The achieved level of automation is sufficiently high since only a single value, i.e. the threshold within the intensity histogram, has to be defined manually to segment an entire human brain. The developed algorithm works well for PLI but is no solution for every possible use case because it is adopted to the characteristics of the present data. Of course, it is possible to define an analog similarity measure δ using different image characteristics for other use cases. In this sense, the developed automated choice of seeds is generally applicable if the image characteristics are well analyzed and the definition of the measure is adopted to them.

The presented algorithm provides good results for PLI as demonstrated in figure 8 given that the images are sufficiently homogeneous. The variances of the section thickness and of the illumination must not be too high since both would result in a distortion of the joint intensity histogram. The extracted edges between the brain and non-brain regions provide an exactness which is adequate for the processing in the subsequent reconstruction steps.

Also the parallelization of the region growing segmentation is worth discussing. Two comparably simple levels of parallelism have been applied to the seeded region growing. The tiles are distributed among the available processes on the first level. For every process on a CPU, an additional GPU is used to compute the data-parallel parts of the algorithm as a second level of parallelism. Both levels can be used separately or in combination. The use of one level does not influence the scaling behavior of the other one, i.e. the additional use of GPUs does not weaken the linear speedup of the multi-core approach. The other way round, distributing the tiles to different CPU-GPU pairs instead of using only one does not have any influence on the acceleration achieved by the use of the GPU. This independence of the two levels of the hybrid parallelization allows an easy porting from the GPU cluster JUDGE to another supercomputer or cluster. It is also possible to use the tool for small parts of a brain on a normal workstation. Nevertheless, it is important to adjust parameters like the thread block size to the respective available hardware.

The linear scaling behavior of the multi-core approach allows to process a given amount of data as fast as required given that enough processes are available in parallel. Let us assume that it would take about 295 days to process a whole human brain using the sequential variant of the algorithm. The additional use of a GPU reduces the runtime to 15 days. In combination with the multi-core implementation using 64 pairs of CPUs and GPUs, it is even reduced to 5.6 hours. This short example demonstrates how well the two parts of the hybrid parallelization complement each other. The multi-core level copes with the immense number of images, the GPU level addresses the large size of the images consisting of about 4,000,000 pixels per tile.

The optimal linear speedup of the multi-core approach is only achieved if an equal load distribution between the processes is provided. This does not only mean that every process has a work package containing the same number of images but also that the average runtime per tile and core is equally distributed. Images that show only one of the regions, i.e. brain tissue or background, are almost entirely covered with seeds. Since all seeds are directly applied to one of the regions in the mask, the number of remaining pixels for the region growing is much smaller compared to “mixed” images. So these one-region images are segmented much faster than mixed ones. If some processes obtain mainly the one-region images and other mainly mixed ones, the first group of processes will be finished much earlier than the second resulting in long idle parts of runtime. Therefore, it is indispensable to find a clever assignment of image tiles to processes to minimize the idle parts of runtime for every brain to be analyzed.

Other parallelizations of segmentations achieve a smaller acceleration per GPU than the presented factor 20, e.g. in [6] an acceleration by a factor of 4.9 to 6.8 depending on the GPU has been reached. In [7], a multithreaded OpenMP parallelization has been described resulting in a speedup of 2.6 for 4 processes, so an efficiency of 0.65. [5] has reached an almost linear speedup like our presented approach.

Since the growing process consumes more than 80% of the total runtime in the GPU parallelized variant, it should be reconsidered to parallelize this step of the algorithm on a third, additional level. Possible approaches might base on a division of each tile in sub-tiles as it is done in [7]-[11]. These sub-tiles may then be processed either using again a multi-core variant, so distributed memory, or a multi-threaded version with shared memory. Nevertheless, this step always includes an additional communication or synchronization effort so that the perfect linear speedup achieved by the presented multi-core approach will not be maintained. Thus, it would have to be evaluated if the runtime gain due to the new level of parallelism is compensated by the induced communication or synchronization effort.

Another approach for a parallelization of the region growing would be a GPU parallel version of this step since this is the only one which is still computed on the CPU(s). Indeed,

this step bases on an intrinsically sequential algorithm. There have been trials for a GPU parallel region growing like [6] but the achieved speedups are not as high as what can be reached in case of other algorithms.

All in all, a further parallelization of the growing process has to be kept in mind but it has to be weighed out if the additional effort is reasonable since with the already implemented two levels of parallelization, the required amount of runtime can be reduced to the desired range given that the needed infrastructure is available. In addition, the time needed to section and image a whole brain is much larger than the few hours or days required to segment the images, i.e. the segmentation is faster than the images are available for access. From this point of view, another enhancement of the parallelization is not required for the PLI data.

V. Conclusion

We presented a seeded region growing segmentation specialized for images of the human brain acquired with the technique of Polarized Light Imaging. The choice of seed pixels has been automated so that only a single point of manual interaction is required in order to process an entire human brain, i.e. a threshold within the joint intensity histogram of all images has to be set. The tool has been parallelized for the GPU cluster JUDGE achieving a linear speedup due to a multicore approach, and another acceleration by a factor of 20 using GPUs in addition to the CPUs. Both levels of parallelism can be used in combination or as alternatives, depending on the available hardware. The segmentation masks are sufficiently accurate for our purposes and the required runtime is fast enough.

Acknowledgement

Partially funded by the Helmholtz Association through the Helmholtz Portfolio Theme “Supercomputing and Modeling for the Human Brain”.

References

[1] M. Axer, K. Amunts, D. Gräßel, C. Palm, J. Dammers, H. Axer, U. Pietrzyk, and K. Zilles, “A novel approach to the human connectome: ultra-high resolution mapping of fiber tracts in the brain,” *NeuroImage*, vol. 54, pp. 1091 – 1101, 2011. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S105381191001178X#

[2] C. Palm, M. Axer, D. Gräßel, J. Dammers, J. Lindemeyer, K. Zilles, U. Pietrzyk, and K. Amunts, “Towards ultra-high resolution fibre tract mapping of the human brain - registration of polarised light images and reorientation of fibre vectors,” *Frontiers in Human Neuroscience*, vol. 4, pp. 1–16, 2010. [Online]. Available: http://www.frontiersin.org/human_neuroscience/10.3389/neuro.09.009.2010/abstract

[3] M. Axer, D. Gräßel, M. Kleiner, J. Dammers, T. Dickscheid, J. Reckfort, T. Hütz, B. Eiben, U. Pietrzyk, K. Zilles, and K. Amunts, “High-resolution fiber tract reconstruction in the human brain by means of three-dimensional polarized light imaging (3D-PLI),” *Frontiers in Neuroinformatics*, vol. 5, no. 34, 2011. [Online]. Available: http://www.frontiersin.org/neuroinformatics/10.3389/fninf.2011.00034/abstract

[4] C.-L. Huang, “Parallel image segmentation using modified hopfield model,” *Pattern Recognition Letters*, vol. 13, no. 5, pp. 345–353, 1992. [Online]. Available: http://www.sciencedirect.com/science/article/pii/016786559290032U

[5] A. Khotanzad and A. Bouarfa, “Image segmentation by a parallel, non-parametric histogram based clustering algorithm,” *Pattern Recognition*, vol. 23, no. 9, pp. 961–973, 1990. [Online]. Available: http://www.sciencedirect.com/science/article/pii/003132039090105T

[6] P. N. Happ, R. Q. Feitosa, C. Bentes, and R. Farias, “A parallel image segmentation algorithm on gpus,” in *Proceedings of the 4th GEOBIA*, may 2012, pp. 580–585.

[7] P. N. Happ, R. S. Ferreira, C. Bentes, G. A. O. P. Costa, and R. Q. Feitosa, “Multiresolution segmentation: a parallel approach for high resolution image segmentation in multicore architectures,” *The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, vol. XXXVIII-4/C7, pp. 28–32, 2005.

[8] D. A. Bader, J. Jájá, D. Harwood, and L. S. Davis, “Parallel algorithms for image enhancement and segmentation by region growing, with an experimental study,” *The Journal of Supercomputing*, vol. 10, no. 2, pp. 141–168, 1996. [Online]. Available: http://dx.doi.org/10.1007/BF00130707

[9] A. Hagan and Y. Zhao, “Parallel 3D Image Segmentation of Large Data Sets on a GPU Cluster,” in *Advances in Visual Computing*, ser. Lecture Notes in Computer Science, G. Bebis, R. Boyle, B. Parvin, D. Koracin, Y. Kuno, J. Wang, R. Pajarola, P. Lindstrom, A. Hinkenjann, M. L. Encarnação, C. Silva, and D. Coming, Eds. Springer Berlin Heidelberg, 2009, vol. 5876, pp. 960–969. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-10520-3_92

[10] Y. Zhuge, Y. Cao, J. K. Udupa, and R. W. Miller, “Parallel fuzzy connected image segmentation on GPU,” *Medical Physics*, vol. 38, no. 7, pp. 4365–4371, 2011. [Online]. Available: http://link.aip.org/link/?MPH/38/4365/1

[11] A. N. Moga and M. Gabbouj, “Parallel Marker-Based Image Segmentation with Watershed Transformation,” *Journal of Parallel and Distributed Computing*, vol. 51, no. 1, pp. 27 – 45, 1998. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0743731598914484

[12] L. Grady, T. Schiwietz, S. Aharon, and R. Westermann, “Random Walks for Interactive Organ Segmentation in Two and Three Dimensions: Implementation and Validation,” in *Medical Image Computing and Computer-Assisted Intervention MICCAI 2005*, ser. Lecture Notes in Computer Science, J. Duncan and G. Gerig, Eds. Springer Berlin Heidelberg, 2005, vol. 3750, pp. 773–780. [Online]. Available: http://dx.doi.org/10.1007/11566489_95

[13] J. Wassenberg, W. Middelmann, and P. Sanders, “An Efficient Parallel Algorithm for Graph-Based Image Segmentation,” in *Computer Analysis of Images and Patterns*, ser. Lecture Notes in Computer Science, X. Jiang and N. Petkov, Eds. Springer Berlin Heidelberg, 2009, vol. 5702, pp. 1003–1010. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-03767-2_122

[14] R. Adams and L. Bischof, “Seeded Region Growing,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 16, no. 6, pp. 641–647, 1994.

[15] O. Gómez, J. A. González, and E. F. Morales, “Image Segmentation Using Automatic Seeded Region Growing and Instance-Based Learning,” in *Progress in Pattern Recognition, Image Analysis and Applications*, ser. Lecture Notes in Computer Science, L. Rueda, D. Mery, and J. Kittler, Eds. Springer Berlin Heidelberg, 2007, vol. 4756, pp. 192–201. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-76725-1_21

[16] I. Sanders, “Seeded region growing using multiple seed points,” in *16th Annual Pattern Recognition Association of South Africa Annual Symposium*. PRASA, 2005, pp. 177–182.

[17] A. Westhoff, “GPU-accelerated Segmentation of high-resolution Human Brain Images acquired with Polarized Light Imaging,” Jülich, 2013, FH Aachen-Jülich, Masterarbeit, 2013. [Online]. Available: http://juser.fz-juelich.de/record/138037

Performance Engineering for a Medical Imaging Application on the Intel Xeon Phi Accelerator

Johannes Hofmann
Chair of Computer Architecture
University Erlangen–Nuremberg
Email: johannes.hofmann@fau.de

Jan Treibig, Georg Hager, Gerhard Wellein
Erlangen Regional Computing Center
University Erlangen–Nuremberg
Email: jan.treibig@rrze.fau.de

***Abstract*—We examine the Xeon Phi, which is based on Intel's Many Integrated Cores architecture, for its suitability to run the FDK algorithm—the most commonly used algorithm to perform the 3D image reconstruction in cone-beam computed tomography. We study the challenges of efficiently parallelizing the application and means to enable sensible data sharing between threads despite the lack of a shared last level cache. Apart from parallelization, SIMD vectorization is critical for good performance on the Xeon Phi; we perform various micro-benchmarks to investigate the platform's new set of vector instructions and put a special emphasis on the newly introduced vector gather capability. We refine a previous performance model for the application and adapt it for the Xeon Phi to validate the performance of our optimized hand-written assembly implementation, as well as the performance of several different auto-vectorization approaches.**

I. Introduction

The computational effort of 3D image reconstruction in Computed Tomography (CT) has required special purpose hardware for a long time. Systems such as custom-built FPGA-systems [1] and GPUs [2], [3] are still widely-used today, in particular in interventional settings, where radiologists require a hard time constraint for reconstruction. However, recently it has been shown that today even commodity CPUs are capable of performing the reconstruction within the imposed time-constraint [4]. In comparison to traditional CPUs the Xeon Phi accelerator, which focuses on numerical applications, is expected to deliver higher performance using the same programming models such as C, C++, and Fortran. Intel first began developing the many-core design (then codenamed Larrabee) back in 2006—initially as an alternative to existing graphics processors. In 2010 the original concept was abandoned and the design was eventually re-targeted as an accelerator card for numerical applications. The Xeon Phi is the first product based on this design and has been available since early 2013 with 60 cores, a new 512 bit wide SIMD instruction set, and 8 GiB of main memory. This paper studies the challenges of optimizing the Feldkamp-Davis-Kress (FDK) algorithm for the Intel Xeon Phi accelerator. The fastest available CPU implementation from Treibig *et al.* [4] served as starting point for the Xeon Phi implementation. To produce meaningful and comparable results all measurements are performed using the RabbitCT benchmarking framework [5].

The paper is structured as follows. Section 2 will give an overview of previous work about the performance optimization of this algorithm. A short introduction to computed tomography is given in Section 3. Section 4 introduces the RabbitCT benchmark and motivates its use for this study. Next we provide a hardware description of the Xeon Phi accelerator together with the results of various micro-benchmarks in Section 5. In Section 6 we give an overview of the implementation and the optimizations employed for the accelerator card. Section 7 contains a detailed performance model for our application on the Xeon Phi. The results of our performance engineering efforts are presented in Section 8; for the sake of completeness we also present the results obtained with compiler-generated code. Finally we compare our results with the fastest published GPU implementation and give a conclusion in Section 9.

II. Related Work

Due to its medical relevance, reconstruction in computed tomography is a well-examined problem. As vendors for CT devices are constantly on the lookout for ways to speed up the reconstruction time, many computer architectures have been evaluated over time. Initially products in this field used special purpose hardware based on FPGA and DSP designs [1]. The Cell Broadband Engine, which at the time of its release provided unrivaled memory bandwidth, was also subject to experimentation [6], [7]. It is noteworthy that CT reconstruction was among the first non-graphics applications that were run graphics processors [2].

However, the use of varying data sets and reconstruction parameters limited the comparability of all these implementations. In an attempt to remedy this problem, the RabbitCT framework [5] provides a standardized, freely available CT scan data set and a uniform benchmarking interface that evaluates both reconstruction performance and accuracy. Current entries in the RabbitCT ranking worth mentioning include *Thumper* by Zinsser and Keck [3], a Kepler-based implementation which currently dominates all other implementations, and *fastrabbit* by Treibig *et al.* [4], a highly optimized CPU-based implementation.

III. Computed Tomography

In diagnostic and interventional computed tomography an X-ray source and a flat-panel detector positioned on opposing ends of a gantry move along a defined trajectory—mostly a circle or helix—around the patient; along the way X-ray images are taken at regular angular increments. In general 3D image reconstruction works by back projecting the information recorded in the individual X-ray images (also called projection images) into a 3D volume, which is made up of individual

voxels (volume elements). In medical applications, the volume almost always has an extent of 512^3 voxels. To obtain the intensity value for a particular voxel of the volume from one of the recorded projection images we forward project a ray originating from the X-ray source through the isocenter of the voxel to the detector; the intensity value at the resulting detector coordinates is then read from the recorded projection image and added to the voxel. This process is performed for each voxel of the volume and all recorded projection images, yielding the reconstructed 3D volume as the result.

IV. RabbitCT Benchmarking Framework

Comparing different optimized FDK implementations found in the literature with respect to their performance can be difficult, because of variations in data acquisition and preprocessing, as well as different geometry conversions and the use of proprietary data sets. The RabbitCT framework [5] was designed as an open platform that tries to remedy the previously mentioned problems. It features a benchmarking interface, a prototype back projection implementation, and a filtered, high resolution CT dataset of a rabbit; also included is a reference volume that is used to derive various image quality measures. The preprocessed dataset consists of 496 projection images that were acquired using a commercial C-arm CT system. Each projection is 1248×960 pixels wide and stores the X-ray intensity values as single-precision floating-point numbers. In addition, each projection comes with a projection matrix $A \in \mathbb{R}^{3\times4}$, which is used to perform the forward projection. The framework takes care of all required steps to set up the benchmark, so the programmer can focus entirely on the actual back projection implementation, which is provided as a module (shared library) to the framework.

A slightly compressed version of the unoptimized reference implementation that comes with RabbitCT is shown in Listing 1. This code is called once for every projection image. The three outer `for` loops (lines 2–4) are used to iterate over all voxels in the volume; note that we refer to the innermost `x`-loop, which updates one "line" of voxels in the volume, as line update kernel. The loop variables `x`, `y`, and `z` are used to logically address all voxels in memory. To perform the forward projection these logical coordinates used for addressing must first be converted to the World Coordinate System (WCS), whose origin coincides with the isocenter of the voxel volume; this conversion happens in lines 6–8. The variables `O` and `MM` that are required to perform this conversion are precalculated by the RabbitCT framework and made available to the back projection implementation in a `struct` pointer that is passed to the back projection function as a parameter. After this the forward projection is performed using the projection matrix A in lines 10–12. In order to transform the affine mapping that implements the forward projection into a linear mapping homogeneous coordinates are used. Thus the detector coordinates are obtained in lines 14 and 15 by dehomogenization.

In the next step a bilinear interpolation is performed. In order to do so, detector coordinates are converted from floating-point to integer type (lines 17 and 18), because integral values are required for addressing the projection image buffer `I`. The interpolation weights `scalex` and `scaley` are calculated in lines 20 and 21. The four values needed for the bilinear interpolation are fetched from the buffer containing the intensity values in lines 25–36. The `if` statements make sure, that the detector coordinates lie inside of the projection image; for the case where the ray doesn't hit the detector, i.e. the coordinates lie outside the projection image an intensity value of zero is assumed (lines 23 and 24). Note that the two-dimensional projection image is linearized, which is why we need the projection image width in the variable `width`—also made available by the framework via the `struct` pointer passed to the function—to correctly address data inside the buffer. The actual bilinear interpolation is performed in lines 38–40.

Before the result is written back into the volume (line 42), it is weighed according to the inverse-square law. The variable `w`, which holds the homogeneous coordinate w, contains an approximation of the distance from X-ray source to the voxel

```
1  // iterate over all voxels in the volume
2  for (z = 0; z < L; ++z) {
3    for (y = 0; y < L; ++y) {
4      for (x = 0; x < L; ++x) {
5        // convert to WCS
6        float wx = O+x*MM;
7        float wy = O+y*MM;
8        float wz = O+z*MM;
9        // forward projection
10       float u = wx*A[0]+wy*A[3]+wz*A[6]+A[9];
11       float v = wx*A[1]+wy*A[4]+wz*A[7]+A[10];
12       float w = wx*A[2]+wy*A[5]+wz*A[8]+A[11];
13       // dehomogenize
14       float ix = u/w;
15       float iy = v/w;
16       // convert to integer
17       int iix = (int)ix;
18       int iiy = (int)iy;
19       // calculate interpolation weights
20       float scalex = ix-iix;
21       float scaley = iy-iiy;
22       // load values for biliean interpolation
23       float valbl = 0.0f; float valbr = 0.0f;
24       float valtr = 0.0f; float valtl = 0.0f;
25       if (iiy >= 0 && iiy < width &&
26           iix >= 0 && iix < height)
27         valbl = I[iiy * width + iix];
28       if (iiy >= 0 && iiy < width &&
29           iix+1 >= 0 && iix+1 < height)
30         valbr = I[iiy * width + iix + 1];
31       if (iiy+1 >= 0 && iiy+1 < width &&
32           iix >= 0 && iix < height)
33         valtl = I[(iiy + 1) * width + iix];
34       if (iiy+1 >= 0 && iiy+1 < width &&
35           iix+1 >= 0 && iix+1 < height)
36         valtr = I[(iiy + 1)* width + iix + 1];
37       // perform bilinear interpolation
38       float valb =(1-scalex)*valbl+scalex*valbr;
39       float valt =(1-scalex)*valtl+scalex*valtr;
40       float val = (1-scaley)*valb+scaley*valt;
41       // add distance-weighted results to voxel
42       VOL[z*L*L+y*L+x] += val/(w*w);
43     } // x-loop
44   } // y-loop
45 } // z-loop
```

Listing 1. Unoptimized Reference Back Projection Implementation Processing a Single Projection Image.

under consideration and can be used to perform the weighting.

V. Intel Xeon Phi

An overview of the Xeon Phi 5110P is provided in Figure 1. The main components making up the accelerator are the 60 cores connected to the high bandwidth ring interconnect through their Core–Ring Interconnects (CRI); interlaced with the ring is a total of eight memory controllers that connect the processing cores to main memory as well as PCIe logic that communicates with the host system.

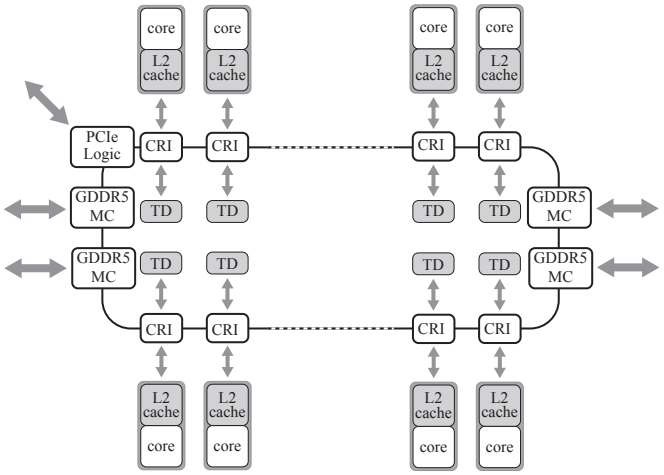


Fig. 1. Schematic Overview of the Xeon Phi 5110P Accelerator.

The cores are based on a modified version of the P54C design used in the original Pentium released in 1995. Each core is clocked at 1.05 GHz and is a fully functional, in-order core, which supports fetch and decode instructions from four hardware thread execution contexts—twice the amount used in recent x86 CPUs. The superscalar cores feature a scalar pipeline (V-pipe) and a vector pipeline (U-pipe). Connected to the U-pipe is the Vector Processing Unit (VPU), which implements the new Initial Many Core Instructions (IMCI) vector extensions.

A. Core Pipeline

The cores used in the Xeon Phi are in-order, lacking all of the necessary logic to manage out-of-order execution, making the individual cores less complex than their traditional CPU counterparts. A core can execute two instructions per clock cycle: one on the V-pipe, which executes scalar instructions, prefetches, loads, and stores; and one on the U-pipe, which can only execute vector instructions.[1] The decode unit is shared by all hardware contexts of a core and is a pipelined two-cycle unit to increase throughput. This means it takes the unit two cycles to decode one instruction bundle (i.e. one micro-op for the U- and one for the V-pipe); however, due to its pipelined design the unit can deliver decoded bundles to *different* hardware threads each cycle. As a consequence, at least two hardware threads must be run on each core to achieve peak performance; using only one thread per core will in the best case result in 50% of peak performance. We found, however, that it is good practise to always use all four hardware threads of a core because most vector instructions have a latency of four clock cycles and data hazards can be avoided without instruction reordering when using four threads.

[1] Actual simultaneous execution is governed by a set of non-trivial pairing rules [8].

B. Cache Organization, Core Interconnect, and Memory

Most of Intel's cache concepts were adopted into the Xeon Phi: the Cache Line (CL) size is 64 bytes and cache coherency is is implemented across all caches using the MESI protocol with the help of the distributed Tag Directory (TD). Each core includes a 32 KiB L1 instruction cache, a 32 KiB L1 data cache, and a unified 512 KiB L2 cache.

The L1 cache is 8-way associative and has a 1 cycle latency for scalar loads and a 3 cycle latency for vector loads. Its bandwidth has been increased to 64 bytes per cycle, which corresponds exactly to the vector register width of 512 bits. In contrast to recent Intel x86 CPUs which contain two hardware prefetching units for the L1 data cache (streaming prefetcher and stride prefetcher), there exist no hardware prefetchers for the L1 cache on the Xeon Phi. As a consequence, the compiler/programmer has to make heavy use of software prefetching instructions—which are available in various flavors (cf. Table I)—to make sure data is present in the caches whenever needed.

TABLE I. Available Scalar Prefetch Instructions for the Intel Xeon Phi.

Instruction	Cache Level	Non-temporal	Exclusive
`vprefetchnta`	L1	Yes	No
`vprefetch0`	L1	No	No
`vprefetch1`	L2	No	No
`vprefetch2`	L2	Yes	No
`vprefetchenta`	L1	Yes	Yes
`vprefetche0`	L1	No	Yes
`vprefetche1`	L2	No	Yes
`vprefetche2`	L2	Yes	Yes

Apart from standard prefetches into the L1 and L2 caches (`vprefetch0, vprefetch1`), there exist also variants that prefetch data into what Intel refers to the L1/L2 non-temporal cache (`vprefetchnta, vprefetch2`). Data prefetched into these non-temporal caches is fetched into the nth way (associativity-wise) of the cache, where n is the context id of the prefetching hardware thread and made MRU—i.e. the most recently used data will be replaced first. Prefetches can also indicate the requested CL be brought into the cache for writing, i.e. in the exclusive state of the MESI protocol (`vprefetche*`).

The L2 cache is 8-way associative and has a latency of 11 clock cycles. The size of the L2 cache is twice the size of recent Intel x86 designs, namely 512 KiB. The L2 cache contains a rudimentary streaming prefetcher that can only detect strides up to 2 CLs apart.

The Xeon Phi contains a total of eight dual-channel GDDR5 memory controllers clocked at 5 GHz, yielding a theoretical peak memory bandwidth of 320 GiB/s. To get an estimate of the attainable bandwidth for our application, we ran a streaming "Update" kernel which resembles the memory access pattern of our application (cf. Figure 2). We found that peak memory performance can only be achieved by employing

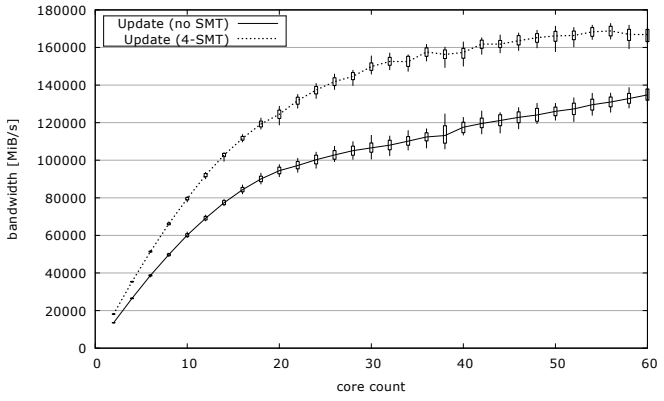


Fig. 2. MEMORY BANDWIDTH OF STREAMING UPDATE KERNEL.

SMT. The bandwidth of about 165 GiB/s corresponds to around 52% of the theoretical peak performance; this can be attributed to limited scalability of the memory system: the gradient of the graph is steeper for e.g. 10–20 cores than it is for e.g. 50–60 cores.

C. Initial Many Core Instructions

While AVX2—the latest set of vector instructions for Intel x86 CPUs—provides a total of 16 vector registers, each 256 bits wide, IMCI offers 32 register, each 512 bits wide. IMCI supports fused-multiply add operations, yielding a maximum of 16 DP (32 SP) Flops per instruction. In addition to increasing the register count and width, IMCI also introduces eight vector mask registers, which can be used to mask out SIMD lanes in vector instructions; this means that a vector operation is performed only selectively on some of the elements in a vector register. Another novelty is the support for vector scatter and gather operations.

D. Vector Gather Operation

In the FDK algorithm, a lot of data has to be loaded from different offsets inside the projection image. The Intel Xeon Phi offers a vector gather operation that enables filling of vector registers with scattered data. A major advantage over sequential loads is the fact that vector registers can be used for addressing the data; this means no detour of writing the contents of vector registers to the stack to move them into scalar registers required for sequential loads is necessary.

```
..L100: vgatherdps zmm6{k3}, [rdi+zmm13*4]
        jkzd       k3, ..L101
        vgatherdps zmm6{k3}, [rdi+zmm13*4]
        jknzd      k3, ..L100
..L101:
```

Listing 2. GATHER PRIMITIVE IN ASSEMBLY. NB PARTICULAR CODE USING TWO BRANCHES SHOWN HERE GENERATED BY C INTRINSIC.

At first glance (cf. Listing 2), a `vgatherdps` instructions looks similar to a normal load instruction. In the example `zmm6` is the vector registers in which the gathered data will be stored. The `rdi` register contains a base address, `zmm13` is a vector register holding 16 32 bit integers which serve as offsets, and `4` is the scaling factor. The 16 bits of the vector mask register act as a write mask for the operation: if the nth bit is set to 1 the gather instruction will fetch the data pointed at by the nth component of the `zmm13` register and write it into the nth component of the `zmm6` register; if the bit is set to 0 no data will be fetched and the nth component of `zmm6` is not modified. When a gather instruction is executed, only data from *one* CL is fetched. This means that when the data pointed at by the `zmm13` register is distributed over multiple CLs the gather instruction has to be executed multiple times. To determine whether all data has been fetched, the gather instruction will zero out the bits in the vector mask register whenever the corresponding data was fetched. In combination with the `jknzd` and `jkzd` instructions—which perform conditional jumps depending of the contents of the vector mask register—it is possible to form loop constructs to execute the gather instruction as long as necessary to fetch all data, i.e. until the vector mask register contains all zero bits.

TABLE II. LATENCIES IN CLOCK CYCLES OF THE VECTOR GATHER PRIMITIVE.

Distribution	L1 Cache		L2 Cache	
	Instruction	Loop	Instruction	Loop
16 per CL	9.0	9.0	13.6	13.6
8 per CL	4.2	8.4	9.4	18.8
4 per CL	3.7	14.8	9.1	36.4
2 per CL	2.9	23.2	8.6	68.8
1 per CL	2.3	36.8	8.1	129.6

A set of micro-benchmarks for likwid-bench from the likwid [9] framework were devised to measure the cycles required to fetch data using gather loop constructs; Table II shows the results, taking into account distribution of data across CLs. We find that the latency for a single gather instruction varies depending on how many elements it has to fetch from a CL. This might be taken as a hint that a single gather instruction itself is implemented as yet another loop, this time in hardware—the larger the number of elements that have to be fetched from a single CL, the higher the latency.

Table III summarizes the hardware specifications of the Xeon Phi and integrates them with two state of the art reference systems from the CPU and GPU domain.

VI. IMPLEMENTATION

Our implementation makes use of all the optimizations found in the original *fastrabbit* implementation [4]. As part of this work, we improved the original clipping mask optimization[2] by 10%. Another improvement we made was to pass function parameters inside vector registers to the kernel in accordance with the Application Binary Interface [10]: instead of replicating values from scalar registers onto the stack and then loading them into vector registers we directly pass the parameters inside vector registers.

While register spilling was a problem in the original implementation, the Xeon Phi with its 32 vector registers can handle all calculations without spilling. The number arithmetic instructions can be greatly reduced by the use of the

[2]For some projection angles several voxels are not projected onto the flat-panel detector. For these voxels a zero intensity is assumed. Such voxels can be "clipped" off by providing proper start and stop values for each `x`-loop.

TABLE III. HARDWARE SPECIFICATIONS OF THE INTEL XEON PHI AND TWO STATE OF THE ART CPU AND GPU REFERENCE SYSTEMS.

Microarchitecture	IvyBridge-EP	Knights Corner	Kepler
Model	Xeon E5-2660 v2	Xeon Phi 5110P	Tesla K20 (GK110)
Clock	2.2 GHz	1.05 GHz	0.706 GHz
Sockets/Cores/Threads per Node	2/20/40	1/60/240	1/13/–
SIMD support	8 SP/4 DP	16 SP/8 DP	192 SP/64 DP
Peak TFlop/s	0.70 SP/0.35 DP	2.02 SP/1.01 DP	3.52 SP/1.17 DP
Node L1/L2/L3 cache	20×32 KiB/20×256 KiB/20×2.5 MiB	60×32 KiB/60×512 KiB/–	13×48 KiB + 13×48 KiB (read-only)/1.5 MiB/–
Node Main Memory Configuration	2×4 ch. DDR3-1866	16 ch. GDDR5 5 GHz	10 ch. GDDR5 5.2 GHz
Node Peak Memory Bandwidth	119.4 GiB/s	320 GiB/s	208 GiB/s

fused multiply-add instructions (cf. lines 6–8, 10–12, 38–40, and 42). All divides are replaced with multiplications of the reciprocal; the reciprocal instruction on the Xeon Phi provides higher accuracy than current CPU implementations and is fully pipelined.

All projection data required for the bilinear interpolation are fetched using gather loop constructs. Several unsuccessful attempts to improve the L1 hit rate of the gather instructions were made. We found that the gather hint instruction, `vgatherpf0hintdps`, is implemented as a dummy operation—it has no effect whatsoever apart from instruction overhead. Another prefetching instruction, `vgatherpf0dps`, appeared to be implemented exactly the same as the actual gather instruction, `vgatherdps`: instead of returning control back to the hardware context after the instruction is executed, we found that control was relinquished only *after* the data has been fetched into the L1 cache, rendering the instruction useless. Finally, scalar prefetching using the `vprefetch0` instruction was evaluated. The problem with this approach is getting the $4 \cdot 16$ offsets stored inside a vector register into scalar registers. This requires storing the contents of the vector register onto the stack and sequentially loading them into general purpose registers. Obviously, 4 vector stores, as well as 64 scalar loads and prefetches, amounting to a total of 132 scalar instructions, is too much instruction overhead. As a consequence we evaluated variants in which only every second (68 instructions), fourth (36 instructions), or eighth (20 instructions) component of the vector registers was prefetched. Nevertheless, the overhead still outweighed any benefits caused by increasing the L1 hit rate.

Because the application is instruction throughput limited, dealing with the if statements (cf. lines 25–36 in Listing 1) using the zero-padding optimization[3] results in better performance than the usage of predicated instructions, which incur additional instructions to set the vector mask registers.

Despite the strictly sequential streaming access pattern inside the volume the lack of a L1 hardware prefetcher mandates the use of software prefetching. We also find that using software prefetching for the L2 cache results in a much better performance than relying on the L2 hardware prefetcher. For the volume data, we used prefetching with the exclusive hint, because the voxel data will be updated. In addition, we deliberately fetch the volume data into the non-temporal portion of the L1 and L2 caches, because we know the volume is too large[4] to fit inside the caches; this way, the volume data will not preempt cached projection data. For prefetched data be available when needed it is important to fetch the data in time. For our application, we achieved best performance when prefetching volume data four loop iterations before accessing them from main memory into the L2 cache and one loop iteration ahead from the L2 into to L1 cache.

Efficient OpenMP parallelization requires more effort on the Xeon Phi than on traditional CPUs. While even on today's high-end multi-socket CPU systems the number of hardware threads is usually below 100, the Xeon Phi features 240 hardware threads. On CPUs it was sufficient to parallelize the outermost `z`-loop (cf. line 2 in Listing 1) and use a static scheduling with chunk size of 1 to work around the imbalances created by the clipping mask. This way each thread is updating one plane of the volume a time. On the Xeon Phi this distribution of work would result in 208 of the 240 threads updating two planes and 32 of the threads updating three planes. In other words 208 threads would be idle 33% of the time. The solution is to make the amount of work more fine-granular, while at the same time ensuring the amount of work will not become so small that the overall runtime is dominated by overhead. To make the work more fine-granular the OpenMP collapse directive was used to fuse the `z` and `y` loops. The optimum chunk size was empirically determined to be 262—corresponding to about half a plane in the volume.

Another important consideration on the Xeon Phi is thread placement. The default "scatter" thread placement, in which thread 0 is run on core 0, thread 1 on core 1, etc. and SMT threads of cores are only used when all physical cores have been exhausted proves unfit for our application. With this scattered placement threads that run on the same physical core have no spatial locality in the volume; as a result they do not have a spatial locality in the projection image, which leads to preemption of projection data in the core's caches (which is shared among the hardware contexts of the core). Using "gather" thread placement in which thread 0 runs on hardware context 0 of core 0, thread 1 on context 1 of core 0, etc. we ensure spatial locality inside the volume and the projection data thus reducing cache preemptions.

VII. PERFORMANCE MODEL

Popular performance models like the Roofline model [11] reduce investigations to determining whether kernels are compute- or memory-bound, not taking runtime contributions of the cache subsystem into account.

[3]Zero-padding refers to an optimization involving allocating a buffer that is large enough to "catch" all projection rays that miss the detector; the original projection image is copied into the buffer and the remainder of the buffer if filled with zero intensity values. The `if` statements to check whether the projected rays lie inside the projection image are thus no longer necessary.

[4]The volume memory footprint is 512^3 Voxels $\cdot$ 4 bytes/Voxel = 512 MiB.

The performance model we use is based on a slightly modified version of the model used in the original *fastrabbit* publication [4], [12]. At the basis of the model is the execution time required to update the 16 voxels in a single CL, assuming all data is available in the L1 cache. In addition, the contribution of the cache and memory subsystem is modeled, which accounts for time spent transferring all data required for the update into the L1 cache and back. In the original model, designed for out-of-order CPUs, an estimation whether the cache subsystem overhead can be hidden by overlapping it with the execution time is given and the authors conclude that there exist sufficient suitable instructions[5] to hide any overhead caused by in-cache transfers. However, in their analysis, Treibig *et al.* only consider the in-cache contribution of the CLs relating to the voxel volume; all CLs pertaining to the projection images, required for the bilinear interpolation, are assumed to reside in the L1 cache. On the Intel Xeon Phi we find this simplification no longer holds true. There is a non-negligible cost for transferring the projection data from the L2 to the L1 cache that can not be overlapped with the execution time.

A. Core Execution Time

Unfortunately there exist no tools such as, e.g., the Intel Architecture Code Analyzer (IACA) [13], which is used to measure kernel execution times on Intel's CPU microarchitectures, for the Intel Xeon Phi. Therefore, we have to perform a manual estimation of the clock cycles spent in a single iteration of the line update kernel—which corresponds to the update of one CL. To complicate things, simply counting the instructions in the kernel is not an option, because the number of gather instructions varies depending on the distribution of the data to be fetched across CLs. As a consequence, we begin with an estimation of the execution time for a gather-less kernel (i.e. a version of the line update kernel in which all gather loop constructs have been commented out).

Manually counting the instructions, we arrive at 34 clock cycles for a gather-less kernel iteration. This analytical estimation was verified by measurement. For one voxel line containing 512 voxels a runtime of 2402 clock cycles was measured using a single thread. This corresponds to 75 clock cycles per kernel iteration (when one iteration updates 16 voxels). Taking into account that the single thread can only issue instruction every other clock cycle, the core execution time for one loop iteration is approximately 37.5 clock cycles—which is a close fit to the value of 34 clock cycles determined previously. For our model, we use the measured value of 37.5 clock cycles because it contains non-negligible overhead that was not accounted for in the analytical value.[6]

To estimate the contribution of the gather loop constructs we first determine how often a gather instruction is executed on average for a CL update. To get this value, we divide the total number of gather instructions issued during the reconstruction (obtained by measurement) by total number of loop iterations. We find that, on average, the gather instruction is executed 16 times in a kernel iteration. Distributing that number over the four gather loop constructs (one for each of the four values required for the bilinear interpolation) we arrive at 4 gather instructions per gather loop—indicating that the data is, on average, distributed across four CLs. From this we can infer the runtime contribution based on our previous findings (cf. Table II). The latency of each gather instruction in the situation where the data is distributed across four CLs is 3.7 clock cycles. With a total of 16 gather instructions per iteration, the contribution is 59.2 clock cycles. Together with the remaining part of one kernel loop iteration (37.5 clock cycles), the total execution time is approximately 97 clock cycles.

B. Cache and Memory Subsystem Contribution

To estimate the impact of the runtime spent transferring the data required for the CL update we first have to identify which transfers can not be overlapped with execution time. As previously established the voxel volume is too large for the caches. Thus each CL of the volume has to be brought in from main memory for the update; eventually, the updated CL will also have to be evicted. This means that a total of 2 CLs, corresponding to 128 byte, have to be transfered. Using software prefetching any latency and transferring cost from the memory and cache subsystems regarding volume data can be avoided.

As previously discussed, prefetching the projection data is not possible without serious performance penalties. Using likwid-perfctr from the likwid framework [9] we investigated the Xeon Phi's performance counters and found that 88.5% of the projection data can be serviced from the local L1 cache and the remaining 11.5% can be serviced from the local L2 cache. Since each gather is transferring a full CL, this amounts to approximately $16\,\text{CLs} \cdot 64\,\text{byte/CL} \cdot 11.5\% \approx 118\,\text{byte}$. We estimate the *effective* L2 bandwidth in conjunction with the gather instruction to be the following: the latency of a single gather instruction (when dealing with data that is distributed across four CLs) was previously measured to be 3.7 clock cycles with data in L1 cache, respectively 9.1 clock cycles with data in the L2 cache (cf. Table II). Assuming the difference of 5.4 clock cycles to be the exclusive L2 cache contribution, we arrive at an effective bandwidth of $64\,\text{byte}/5.4\,\text{cycle} = 11.85\,\text{byte/cycle}$. The average memory subsystem contribution is thus $118\,\text{byte}/11.85\,\text{byte/cycle} \approx 10\,\text{cycles}$.

Figure 3 provides an overview of the performance model. The upper part shows core and L1 cache, together with all data transfers from the cache. The lower part shows the memory hierarchy through which data has to be transfered to perform the CL update. The arrows to the left represent the CLs pertaining the voxel volume data; prefetching these CLs in time guarantees overlap of transfers with core execution. The arrow to the right between the L1 and L2 caches represents the transfers of projection data which can not be prefetched; the latency of these transfers is the determining factor for the memory subsystem contribution. This leads to a total of 107 clock cycles to perform a single CL update.

Based on the runtime of a single kernel iteration we can determine whether the memory bandwidth becomes a limiting

[5]Because the L1 cache is single-ported—i.e. it can only communicate with either the core or the L2 cache at any given clock cycle—transfers between the L1 and L2 caches can only overlap with "suitable" instructions that do not access the L1 cache such as, e.g., arithmetic instructions with register operands.

[6]The overhead includes the time it takes to call the line update kernel (backing up and later restoring callee-save registers, the stack base pointer, etc. onto the stack) as well as instructions in the kernel that are not part of the loop body, such as resetting the loop counter.

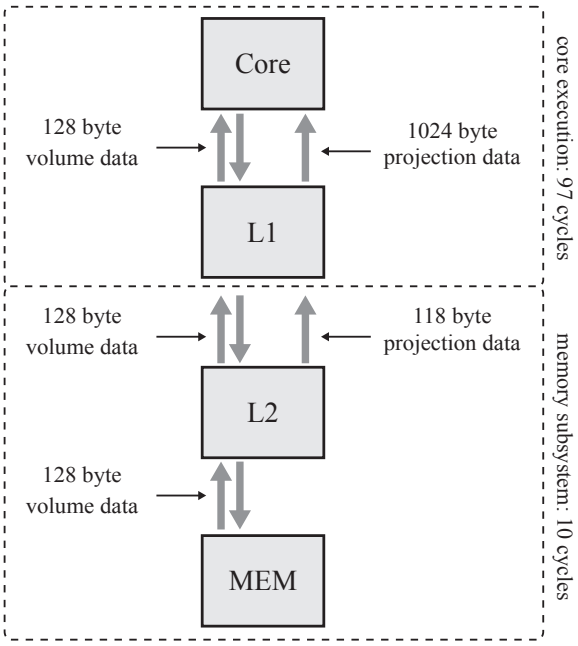


Fig. 3. OVERVIEW OF EXECUTION TIME AND MEMORY SUBSYSTEM CONTRIBUTION.

factor for our application. For each loop iteration, 128 byte (2 CLs) have to be transfered over the memory interfaces. Each of the 60 cores is clocked at 1.05 GHz; at 107 cycles per iteration, the required bandwidth is:

$$\frac{1.05\,\text{GHz/core}}{107\,\text{cycles}} \cdot 60\,\text{cores} \cdot 128\,\text{byte} = 70.0\,\text{GiB/s}.$$

The required value is well below the measured sustainable bandwidth of around 165 GiB/s (cf. Fig. 2), indicating that bandwidth is not a problem for our application.

Given the model, the total runtime contribution of the line update kernel is

$$\frac{4.39 \cdot 10^{10}\,\text{voxels}}{16\,\text{voxels/iteration}} \cdot \frac{107\,\text{cycles/iteration}}{60\,\text{cores} \cdot 1.048\,\text{GHz/core}} = 4.67\,\text{s}.^{7}$$

Unfortunately, there is a non-negligible amount of time spent outside of the line update kernel. The value obtained by measuring the runtime of the reconstruction with the call to the kernel commented out was 0.42 s. Thus, the total reconstruction time is 5.09 s. Foreclosing the runtime of the assembly implementation from the next section which is 5.16 seconds (cf. Table IV) we estimate the model error at 1.4%.

VIII. RESULTS AND DISCUSSION

In addition to our hand-written assembly implementation we also evaluated several auto-vectorization approaches for the FDK kernel: native vectorization using the Intel C Compiler, the only recently introduced vectorization directive from the latest OpenMP 4 standard [14] implemented in the Intel Compiler, and the Intel SPMD Program Compiler (ISPC) [15]. Table IV shows the runtime in seconds and the corresponding performance in Giga Voxel Updates per Second (GUp/s)—the commonly used performance metric for FDK—of all implementations. All implementations were benchmarked using static OpenMP scheduling with a chunk size of 262 voxel lines; independent of the implementation, this value resulted in the best performance.

TABLE IV. RUNTIMES AND PERFORMANCE OF ALL IMPLEMENTATIONS FOR A 512^3 VOLUME.

Version	time [s]	Performance [GUp/s]
OpenMP 4 (`#pragma simd`)	7.77	5.6
ISPC (Version 1.5.0)	7.00	6.3
Intel C Compiler (Version 13.1.3)	6.99	6.3
Assembly	5.16	8.5

We find that the performance of auto-vectorization variants can not match the speed of our manually written assembly kernel. Even the best of the three variants—the native vectorization of the Intel C Compiler—can only provide around 74% of the performance of hand-written code. We find that the performance provided by the latest version of the free, open-source ISPC almost matches that of the commercial Intel C Compiler; the original ISPC version that was used during the early stages of this work had a 10% lower performance. The result obtained with the OpenMP 4 directive was the worst; the main reason is that the standard guarantees the results obtained with this vectorization are identical to that of scalar code—thus prohibiting several optimizations, such as reordering of arithmetic instructions to increase performance.

Our performance model revealed that even in the ideal case in which all projection data resides in the L1 cache, the runtime impact of gathering the projection data (59.2 cycles) dominates the overall runtime of a kernel iteration (97 cycles). We thus identify the gather operation as the limiting factor for this application. While all other parts of the FDK kernel benefit from the increase of the vector register width at the same time the increased width counteracts the performance, because the cost of filling the vector registers with scattered data increases linearly with register width.

IX. CONCLUSION

We have presented a detailed examination of the Intel Xeon Phi accelerator by performing various benchmarks. We performed various optimizations for the FDK algorithm and devised a manually vectorized assembly implementation for the Xeon Phi and compared it to auto-vectorized code. In order to integrate our findings with today's state of the art reconstruction implementations a comparison of our implementations with an improved[8] version of the *fastrabbit* implementation, as well as the fastest currently available GPU implementation called *Thumper* [3] is shown in Table V.

We find that the Kepler-based GeForce GTX 680 by Nvidia can perform the reconstruction 7–8 times faster, depending on the volume's discretization. This discrepancy can not be explained by simply examining the platforms' specifications

[7] The total number of voxels to process is given by considdering each voxel of the clipped volume once for each of the 496 projection images.

[8] Back-porting various optimizations of the Xeon Phi implementations yielded a 25% increase in performance for the *fastrabbit* CPU implementation.

TABLE V. COMPARISON OF DIFFERENT PLATFORMS IN GUP/S.

Platform	Version	512^3	1024^3
2S-Xeon E5-2660	*improved fastrabbit*	6.2	6.7
Xeon Phi 5110P	OpenMP 4 (`#pragma simd`)	5.6	5.7
	ISPC (Version 1.5.0)	6.3	6.4
	Intel C Compiler (Version 13.1.3)	6.3	6.4
	Assembly	8.5	13.1
GeForce GTX 680	*Thumper*	67.7	88.2

such as peak Flop/s and memory bandwidth. The main causes contributing to the GPU's superior performance for this particular application are discussed in the following.

Most computations involved in the reconstruction kernel, such as the projection of voxels onto the detector panel or the bilinear interpolation, are typical for graphics applications (which GPUs are designed for). While, due to the fused multiply-add operation, the forward projection is performed efficiently on both the GPU and the Xeon Phi platform, the bilinear interpolation is not. GPUs posses additional hardware called texture units, each of which can perform a bilinear interpolations using a single instruction for data inside the texture cache. To emphasize the implications, consider that out of the total of 97 clock cycles for one loop iteration of the FDK kernel, 6 cycles are used for the computation of the detector coordinates and 3 cycles to weight the interpolated intensity value and update the voxel volume; the remaining 88 clock cycles, more than 90% of the kernel, is spent on the bilinear interpolation[9]—which is handled by a single instruction on a GPU.

Given a sufficient amount of work, Nvidia's CUDA programming model does a better job at hiding latencies. As seen before, even in the ideal case where all data can be serviced from the L1 cache, on average, each of the gather instructions has a latency of 3.7 clock cycles. Although the Intel Xeon Phi can hide the latencies of most instructions when using all four hardware contexts of a core, 4-way SMT is not sufficient to hide latencies caused by loading non-continuous data. In contrast to SMT, Nvidia's multiprocessors feature hardware that allows them to instantly switch between warps.[10] This way, every time a warp has to wait for an instruction to complete or data to arrive from the caches or main memory, the hardware simply schedules another warp in the meantime. Given a sufficient number of warps to choose from, this approach can hide much higher latencies than the 4-way SMT in-order approach.

Although we have shown that the Intel Xeon Phi accelerator can not provide the same performance as GPUs for the task of 3D reconstruction in the interventional setting, there nevertheless might be applications that can benefit from our work. One promising application seems to be the reconstruction of large CT volumes. Today, the largest industrial CT scanner, which at the time of this writing is the XXL-CT device only recently installed by the Fraunhofer Institute in Fürth [16], is capable of recording projection images with a resolution of 10000×10000 pixels, corresponding to more than 380 MiB per projection image. In this setting, it is possible for main memory capacity and bandwidth to play more important roles, potentially giving CPUs, with their high memory capacities, and the Intel Xeon Phi, with its high memory bandwidth, an advantage over GPUs. Another interesting topic of research, of course, will be to evaluate the next iteration of the Intel Xeon Phi architecture, codenamed Knights Landing, for this application once it becomes available.

[9] This includes preparing the interpolation weights, converting floating-point detector coordinates to integral values, gathering the projection data, and performing the actual interpolation.

[10] On Nvidia GPUs, the number of CUDA threads concurrently executing on a core is called warp.

REFERENCES

[1] B. Heigl and M. Kowarschik, "High-speed reconstruction for C-arm computed tomography," in *In Proceedings Fully 3D Meeting and HPIR Workshop*, July 2007, pp. 25–28.

[2] G. Pratx and L. Xing, "Gpu computing in medical physics: A review," *Medical Physics*, vol. 38, no. 5, pp. 2685–2697, 2011. [Online]. Available: http://link.aip.org/link/?MPH/38/2685/1

[3] T. Zinsser and B. Keck, "Systematic Performance Optimization of Cone-Beam Back-Projection on the Kepler Architecture," in *Proceedings of the 12th Fully Three-Dimensional Image Reconstruction in Radiology and Nuclear Medicine*, F. committee, Ed., 2013, p. 225228.

[4] J. Treibig, G. Hager, H. G. Hofmann, J. Hornegger, and G. Wellein, "Pushing the limits for medical image reconstruction on recent standard multicore processors," *International Journal of High Performance Computing Applications*, 2012, (Accepted). [Online]. Available: http://arxiv.org/abs/1104.5243

[5] C. Rohkohl, B. Keck, H. Hofmann, and J. Hornegger, "RabbitCT - an open platform for benchmarking 3D cone-beam reconstruction algorithms," *Medical Physics*, vol. 36, no. 9, pp. 3940–3944, 2009.

[6] M. Kachelriess, M. Knaup, and O. Bockenbach, "Hyperfast parallel-beam and cone-beam backprojection using the cell general purpose hardware." *Med Phys*, vol. 34, no. 4, pp. 1474–86, 2007. [Online]. Available: Http://www.biomedsearch.com/nih/Hyperfast-parallel-beam-cone-backprojection/17500478.html

[7] H. Scherl, M. Kowarschik, H. G. Hofmann, B. Keck, and J. Hornegger, "Evaluation of state-of-the-art hardware architectures for fast cone-beam ct reconstruction," *Parallel Comput.*, vol. 38, no. 3, pp. 111–124, Mar. 2012.

[8] "Intel Xeon Phi Coprocessor Vector Microarchitecture." [Online]. Available: http://software.intel.com/sites/default/files/article/393199/intel-xeon-phi-coprocessor-vector-microarchitecture.pdf

[9] J. Treibig, G. Hager, and G. Wellein, "LIKWID: A lightweight performance-oriented tool suite for x86 multicore environments," in *PSTI2010, the First International Workshop on Parallel Software Tools and Tool Infrastructures*. Los Alamitos, CA, USA: IEEE Computer Society, 2010, pp. 207–216. [Online]. Available: http://dx.doi.org/10.1109/ICPPW.2010.38

[10] Intel Corporation, *System V Application Binary Interface — K1OM Architecture Processor Supplement*, April 2012.

[11] S. W. Williams, A. Waterman, and D. A. Patterson, "Roofline: An insightful visual performance model for floating-point programs and multicore architectures," EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2008-134, Oct 2008.

[12] J. Treibig and G. Hager, "Introducing a performance model for bandwidth-limited loop kernels," in *Parallel Processing and Applied Mathematics*, ser. Lecture Notes in Computer Science, R. Wyrzykowski, J. Dongarra, K. Karczewski, and J. Wasniewski, Eds. Springer Berlin / Heidelberg, 2010, vol. 6067, pp. 615–624.

[13] "Intel architecture code analyzer." [Online]. Available: http://software.intel.com/en-us/articles/intel-architecture-code-analyzer/

[14] OpenMP Architecture Review Board, *OpenMP Application Program Interface — Version 4.0*, July 2013.

[15] M. Pharr and W. R. Mark, "ispc: A SPMD Compiler for High-Performance CPU Programming," in *In Proceedings Innovative Parallel Computing (InPar)*, San Jose, CA, May 2012.

[16] "Fraunhofer Institute, XXL-CT." [Online]. Available: http://www.iis.fraunhofer.de/de/bf/xrt/system/xxl-ct.html

PBA2CUDA - A Framework for Parallelizing Population Based Algorithms Using CUDA

Ioannis Zgeras
Institute of System- and Computer Architecture
Leibniz Universitaet Hannover
30167 Hannover
Email: zgeras@sra.uni-hannover.de

Jürgen Brehm
Institute of System- and Computer Architecture
Leibniz Universitaet Hannover
30167 Hannover
Email: brehm@sra.uni-hannover.de

Michael Knoppik
Institute of System- and Computer Architecture
Leibniz Universitaet Hannover
30167 Hannover

***Abstract*—To increase the performance of a program, developers have to parallelize their code due to trends in modern hardware development. Since the parallelization of source code is paired with additional programming effort, it is desirable to provide developers with tools to help them by parallelizing source code. PBA2CUDA is a framework for semi-automatically parallelization of source code specialized in the algorithm class of Population Based Algorithms.**

I. Introduction

Modern computer architectures consist of a variety of hardware like multi-core CPUs, General-Purpose Graphics Processing Units (GPGPUs) and also specialized hardware like Field Programmable Gate Arrays (FPGAs). With this powerful parallel hardware, new challenges for software developers emerge. Serial programs do not benefit from parallel architectures as the program code is executed sequentially on a single computation node. The program code has to be parallelized to use the computation capabilities of these hardware. This process is not trivial, many constraints have to be fulfilled, like data dependencies and synchronization barriers. Furthermore, the developers have to get used to new frameworks and APIs like OpenMP [1] for multi-core CPUs or CUDA (s. Sec. II-C) for GPGPUs. The memory architecture of modern parallel hardware is complex as well. Particularly, the memories of GPGPUs consist of many different layers with different characteristics and sizes. Using the correct memory with the correct access method is crucial for fast execution of a parallel program.

This paper presents a novel approach for semi-automatically parallelizing serial source code on GPGPUs. Thereby, our tool is specialized on *Population Based Algorithms* (PBAs), an algorithm class from the area of *Biologically-Inspired Algorithms*. The main characteristics of this algorithms are the iterative execution of special functions to solve the given problem by a large number of individuals (Sec. II-B), that makes PBAs suitable for parallelization. Furthermore, by setting the focus on a specific algorithm class, our framework can perform optimization procedures regarding this particular class of algorithms.

The paper is organized as follows: Section II contains a short survey of related work. In Section III we describe the concept and implementation of our parallelization framework. Section IV presents the evaluation results. Finally, Section V concludes with further research opportunities.

II. State of the Art

A. Parallelization Frameworks

There are many parallelization frameworks in the literature transferring serial C/C++ code into CUDA code. The general approach of all frameworks is to mark parts of source code that have be transformed by the specific framework with directives. However, the features of the available frameworks differ. We have created a list of important criteria for PBA2CUDA comparing different frameworks. The criteria for the comparison are:

1) Data transfer between CPU and GPGPU is influenceable.
2) Memory on GPGPU can be allocated and freed manually.
3) Every memory hierarchy of the GPGPU is usable.
4) Loop optimization possible.
5) C++ support
6) Free to use

The results of the comparison are shown in Table I. HMPP and OpenMPC are both supporting the most features. While HMMP supports C++, it is not free to use. We have decided to use OpenMPC, as we wanted to implement an open framework and important C++ features can be supported by implementing C++ to C parser transforming, for example, C++-*Vectors* to C-*Arrays* or C++-*Classes* to C-*Structs*.

B. Population Based Algorithms

PBAs are nature inspired heuristics, all PBAs have similar structures. The main part is the population that consists of a set of solutions for a given problem. These solutions are called individuals, particles or, more general, agents. These agents execute in each iteration of the algorithm different kinds of operations to improve their solution. The quality of a solution is called *fitness*. The function or problem that the agents have to optimize (or solve) is called fitness function. Two representatives of PBAs the so called *Genetic Algorithms* (GAs) and *Particle Swarm Optimization* (PSO) that are used for this paper are now described in more detail.

Criterion	PGI [2]	OpenACC [3]	HMPP [4]	OpenMPC [5]	hiCUDA [6]	R-Stream [7]
1	+	+	+	+	+	+
2	+	+	+	+	+	-
3	-	-	+	+	+	-
4	-	-	+	+	-	-
5	+	+	+	-	-	-
6	-	-	-	+	-	-

TABLE I: Comparison PBA2CUDA criteria

1) Genetic Algorithms:: GAs [8][9] are heuristics based on the idea of natural selection. The population of GAs consists of a set of individuals that represent a solution of a given problem where every solution consists of single chromosomes. As an example the solution of an individual for an *n*-dimensional function would consist of *n* values of this function. The vector representing these values is the chromosome of this individual and every value of this vector is called *gene*. The outer iteration loop of GAs consists of three main operations - *Crossover*, *Mutation* and *Selection* - that are performed after a random initialization process until a break condition (e.g. *finding the minimum*) is achieved.

The crossover operation uses two individuals (parents) to generate new individuals (children) by crossing the solutions of the two parents. There are many different crossover operations, some popular examples can be found in [8]. In the next step, some chromosomes of the individuals are randomly changed, this operation is called mutation. Again, there are different methods for implementing mutation [8]. The individuals are now ranked based on their *fitness value* that indicates the quality of their solution. In the last step, the individuals are selected to become part of the population for the next iteration. Once again, there are many different ways to select the individuals [8]. There are many parallel implementations for GAs, see [10][11][12].

2) Particle Swarm Optimization:: PSO algorithms [13][14] have similarities to GAs but the approach is different. PSO is based on the natural behavior of birds. The population of a PSO algorithm is called *swarm* and the individuals are called *particles*. Every particle is represented by a position and a velocity where the position represents a solution of the problem and velocity the speed and direction this particle changes its position.

In each iteration step, the particles try to approximate better solutions by detecting the best neighbor and updating their velocity and position value taking into account the velocity and position values of the best neighbor. The iteration loop is repeated until a break condition is met. Like for GAs, different parallel PSO algorithms can be found in the literature [15][16].

C. CUDA

As GPUs became more and more complex, their use was no longer limited to tasks belonging to graphical programming. Different programming languages were developed to facilitate the GPUs towards more general purpose processing. The company NVIDIA published CUDA (Compute Unified Device Architecture) as a programming environment for their GPUs. CUDA became a common programming language extending the C language by a small set of instructions, allowing the programmer to develop code running in parallel. The parallel code is executed on so-called CUDA kernels. More details about CUDA can be found in [17][18].

D. ROSE

For our code analysis approach, we have used the ROSE Compiler [19], a source- to-source transformation and analysis tool. ROSE transforms the source code into an Abstract syntax tree (AST) that can be modified and transformed back into compilable source code.

III. PBA2CUDA Framework

This section describes the architecture of the parallelization framework PBA2CUDA. It is divided in three parts, in the first part the general concept of PBA2CUDA is shown while in the second part the actual implementation of the framework is described. Finally, in the third part, the pre-conditions that are necessary for the use of PBA2CUDA are shown.

A. PBA2CUDA Concept

PBA2CUDA is a framework for parallelizing serial PBAs. The generic approach is shown in Fig. 1, showing three main modules, the *Parallelization*-module, the *PBA Optimization*-module and the *CUDA Optimization*-module.

The Parallelization-module is the main module converting a serial PBA into a parallel form that can be executed on a GPGPU. The procedure of the module is semi-automatic and needs information about the parts of the code that have to be converted into CUDA-code. These areas have to be marked by pre-defined directives by the developer. Also, the Parallelization-module provides pre- and post-processing tools for the source-code that are needed by the framework. The PBA-Optimization module gets the parallelized source code as input and executes optimization operations dedicated to PBA. Finally, the CUDA Optimization module executes optimization operations focused on the CUDA specific part to accelerate execution of the code and increase the precision of the results.

B. PBA2CUDA Implementation

Here, the implementation of the generic modules shown in Section III-A is described. The concrete modules representing

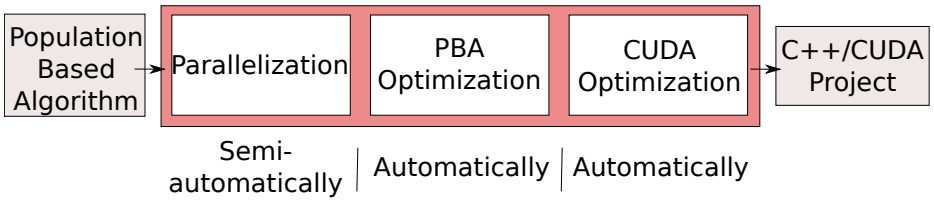


Fig. 1: PBA2CUDA Generic Framework

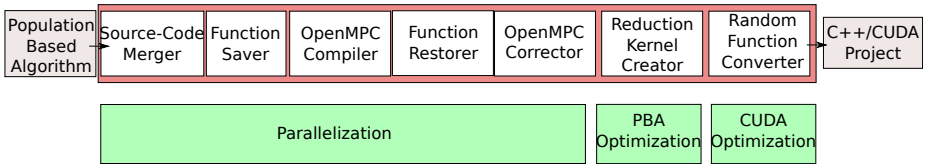


Fig. 2: PBA2CUDA Framework

parts of the generic modules are described and discussed. As shown in Fig. 2, the generic modules consist of several concrete modules specialized on single tasks of PBA2CUDA.

1) Source-Code Merger: A limitation of PBA2CUDA is the necessity to use only a single file when parallelizing source-code, which is unfavorable for larger programs. This limitation raises from the use of ROSE (s. II-D) as our main code parsing tool. To face this limitation we have implemented the *Source-Code Merger* (SCM) module to merge the single files. The SCM module searches for all available header and source files available. Next, the main method is identified and all functions of the remaining source code files are copied in the associated header files. All source-code/header files combinations that belong together are stored in a single header file, respectively. In the last step, the merged header files have to be copied into the main file in the correct order to avoid dependencies. To resolve the dependencies, the main file is parsed recursively and every header include declaration is exchanged by the dedicated merged header file. The included header files are stored to avoid multiple declarations.

2) Function Saver: The used parallelization framework OpenMPC executes optimization operations that raise errors in the source code. As an example, OpenMPC deletes functions that are referenced only as function pointers. However, the references themselves remain in the source code which leads to incorrect code. The *Function Saver* (FS) marks the referenced functions with directives to face this problem. In the following, the marked functions are stored into a separate file and deleted from the source file. Furthermore, the references are removed to avoid errors when parallelizing the code.

3) OpenMPC: We have used OpenMPC (s. II-A) to transform the serial C-code into CUDA code. This step is done fully automatically by OpenMPC and has no further improvements by our framework.

4) Function Restorer: The *Function Restorer* module restores the functions that were removed by the FS module and the parallelization procedure. Subsequently, the FS restores the missing functions and the function pointer references are restored in the source file.

5) OpenMPC Corrector: The OpenMPC compiler generates different errors that can not be corrected by the previous modules and have to be corrected by the *OpenMPC Corrector* (OC) module. In the following, the possible errors are enumerated and the used correction method is described.

1) Sometimes OpenMPC deletes the __host__ declaration in front of a function. This declaration indicates the compiler that the corresponding function can be executed on the host (CPU) system. All __device__ device expressions, that determine a device function, therefore a function that is executed on the GPGPU, are replaced by __host__ __device__ expressions. Functions with this markings can be executed on the GPGPU as well as on the CPU.
2) The NULL pointer is replaced by the expression *(void*)0*. This expression is readable for most compilers but raises an error when parsed by ROSE. Therefore, the OC module transforms the (void*)0 expression back to a standard NULL expression.

6) Reduction Kernel Creator: One of the main tasks in PBAs (s. II-B) is to find the best individual after each iteration. Comparisons between single threads on the GPGPU are not trivial due to synchronization issues and wasted clock cycles while waiting for the last thread to end its task. Generally, a simple search algorithm to find the best value is within the complexity $\mathcal{O}(n)$, where n is the number of different values. Every single value has to be compared iteratively with its neighbours until all values have been compared. As this operation is essential for PBAs, we have implemented the *Reduction Kernel Creator* (RKC) module to speed up comparison operations on GPGPUs. The reduction method is a common approach for finding the best value within a set of values. The main approach is shown in Fig. 3. $n/2$ threads have to be executed in the first iteration step to compare n values. Every thread compares two values and saves the best value for the next iteration step. The next $(n/2)/2$ threads are started and execute the same operation. The algorithm stops when the best value is found.

This approach speeds up the PBA and is in the complexity class $\mathcal{O}(log(n))$. A drawback of this approach is the use

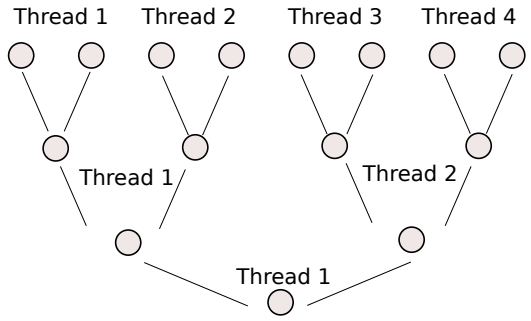


Fig. 3: Reduction procedure

of a small number of threads, that leads to an underutilized GPGPU.

7) Random Function Converter: Random calls are an essential part in PBAs for different operations like mutation or crossover (s. II-B). Unfortunately, the OpenMPC compiler does not consider external function calls like the C *rand()*. We have implemented the *Random Function Converter* (RFC) module to transfer standard C random calls into CUDA random calls using the CUDA library *thrust*. Two options have to be considered: The random call can be found directly in the kernel function or the random call is nested in a device function called by the kernel function. RFC parses the functions recursively to detect every random call and exchanges the call with a thrust random call. A seed is transferred from the host to the device function and concatenated with the thread id of the threads executed on the GPGPU to obtain independent random numbers

C. Preconditions for automatic parallelization

The serial source code that is to be transferred by PBA2CUDA has to meet some preconditions. These preconditions arise out of the used frameworks like ROSE or OpenMPC. The parts of the source code that are to be parallelized have to be written in C without C++ constructs. Furthermore, the parts that are to be parallelized have be marked using OpenMPC directives. We are working on methods to extend PBA2CUDA. Some of our work in progress can be found in Sec. V.

IV. Evaluation

To evaluate to quality of the parallel souce-code generated by PBA2CUDA we have implemented different PBAs (GA and PSO) solving common benchmark functions that can be found in [20]. We then compared the achieved speedup of the automatically parallelized algorithms, manually parallelized implementations and parallelization on the CPU (OpenMP [1]) in relation to serial versions of the algorithms. Due to page limitations, we present in this paper two functions, the *Euclid* function and the more complex *Griewank* function:

1) Euclid: $f(\vec{x}) = \sqrt{\sum_{i=1}^{n}(x_i - 500)^2}$

2) Griewank: $f(\vec{x}) = \frac{1}{4000}(\sum_{i=0}^{n-1} x_i^2) + (\prod_{i=0}^{n-1} \cos(\frac{x_i}{\sqrt{i+1}})) + 1$

We have chosen the suggested parameters from [20] for the dimension size (30 and 100) and have also evaluated a larger dimension size (500). The parameters for the population size are 100 and 500, respectively, covering medium and large population sizes. The test bench consists of an Intel Core-i7-2960XM with 4 cores and a NVIDIA GeForce GTX 580M GPGPU with 384 CUDA-cores.

A. Results: Genetic Algorithm

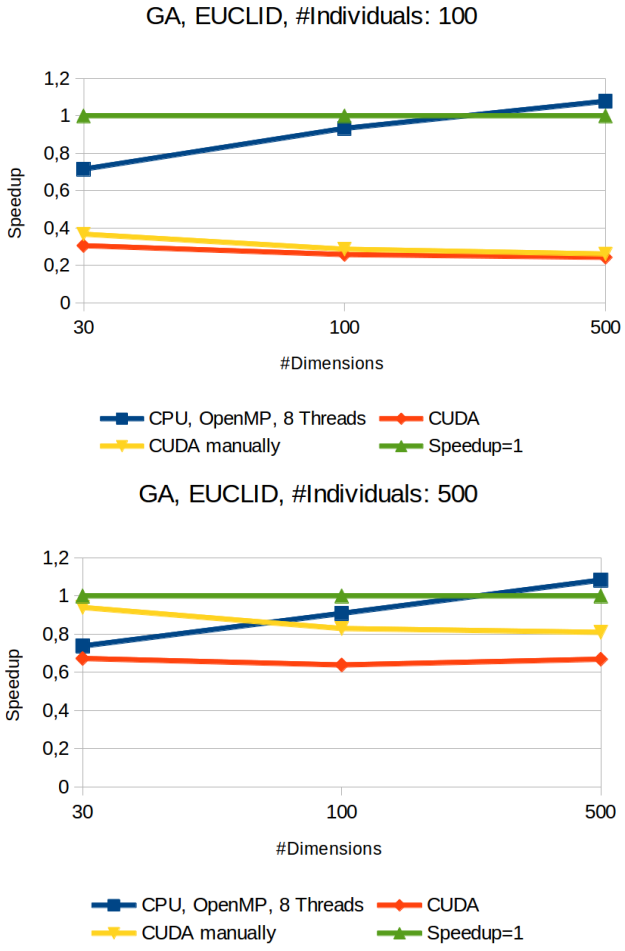


Fig. 4: Genetic Algorithm - Euclid function

1) Euclid Function:: In Fig. 4, the results for the GA algorithm solving the Euclid function are shown. On the left figure, the results using an individual size of 100 are shown while on the right figure, the results for 500 individuals are shown. The parallel versions of the algorithm do not perform well compared to the serial version (Speedup=1 - green curve) due to the simplicity of the Euclid function. The parallelization overhead is much larger than the speedup gained by parallelizing the algorithm. Comparing the parallel versions, the OpenMP version (CPU, OpenMP,8 Threads - blue curve) shows the best results following by the manually implemented CUDA version (CUDA manually - yellow curve) and the PBA2CUDA version (CUDA - red curve).

2) Griewank Function:: The Griewank function is more complex to compute than the Euclid function. In Fig. 5

the results of the GA computing the Griewank function are shown. Even for smaller population sizes (upper figure) a speedup using the parallel versions of the program is achieved. Whereby, the OpenMP version outperforms both the manually implemented CUDA version and the automatically parallelized version by PBA2CUDA. However, for larger population sizes (bottom figure) the CUDA versions outperform the OpenMP version of the program confirming the general assumption that a parallelization of GPGPUs is worthwhile only for large problems. Here again, the manual CUDA implementation shows better results than the PBA2CUDA version. This is achieved by implementing all optimization techniques of PBA2CUDA in our manually optimized implementation of the algorithm.

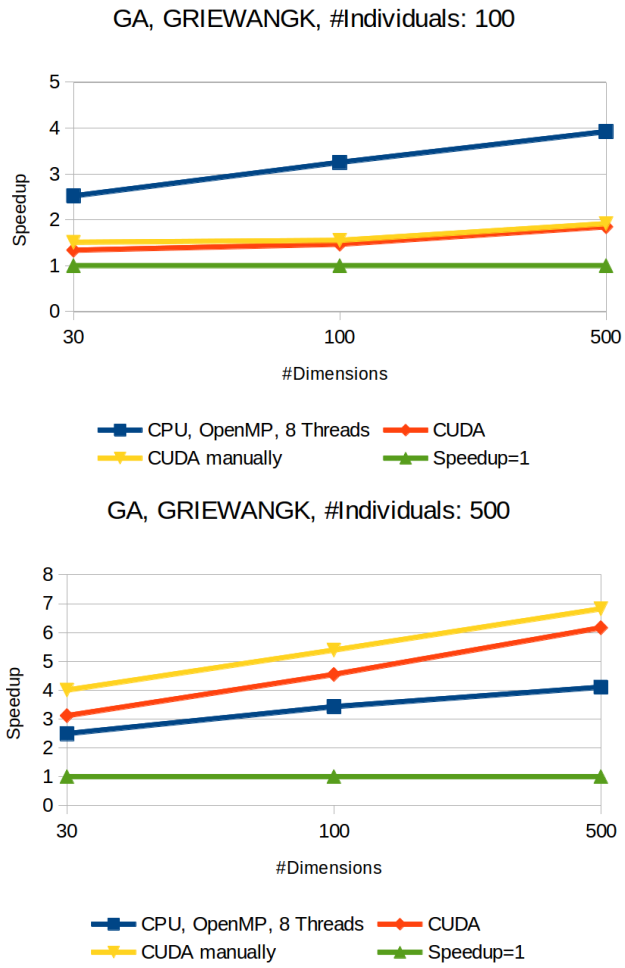


Fig. 5: Genetic Algorithm - Griewank function

B. Results: Particle Swarm Optimization

1) Euclid Function:: The results of the PSO solving the Euclid function show slightly different characteristic than the GA. While the OpenMP version again performs better than the CUDA versions, all parallelized versions perform better than the serial version of the PSO in larger problem sizes (right figure). The reason for this performance is the implemented Reduction Kernel (RK), that is only used for the PSO and can not be used by the GA implementation. The RK is also the reason for the sharp bend in the curves for population sizes greater than 100. While the RK speeds up the calculation of the best particle, it does not utilize the GPGPU very well leaving idle cores (s. Sec. III-B). For larger problem sizes, this leads to a reduction of the RK performance.

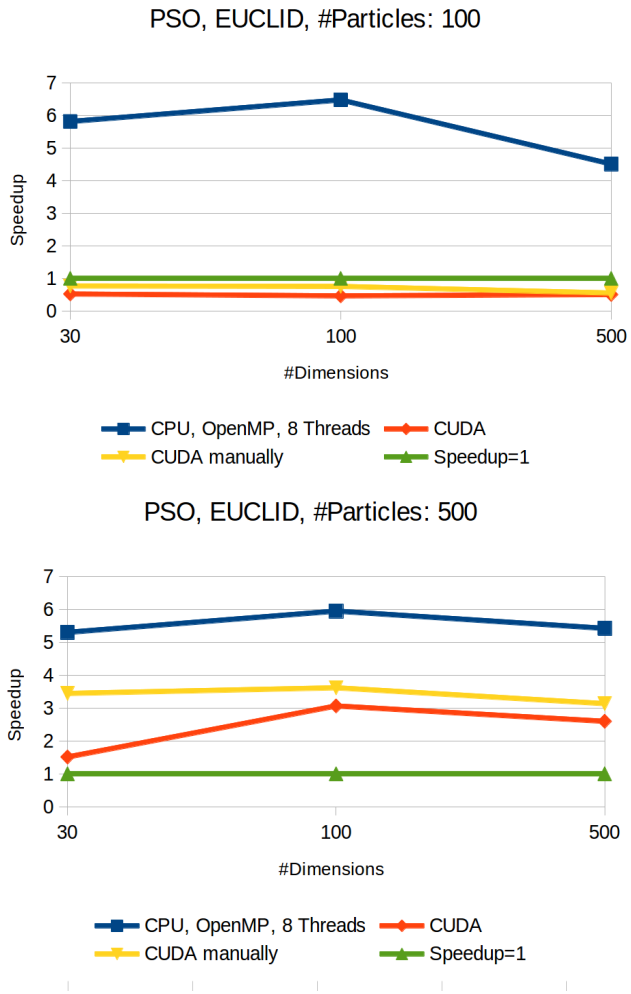


Fig. 6: Particle Swarm Optimization - Euclid function

2) Griewank Function:: Similar to the Euclid function, all parallel versions perform better than the serial version of the PSO for larger problems (right figure). Furthermore, the CUDA versions, both the manually implemented and the PBA2CUDA version, perform better than the OpenMP version. The computation time of the RK has a smaller impact in the general execution time due to the relatively large compute time of the other PSO functions and by this the CUDA versions have no quality fall-off in their performances for population sizes greater 100.

The evaluation results show similar performance of the versions parallelized by PBA2CUDA and the manually parallelized versions of the PBAs. Considering the minimal effort parallelizing a PBA with PBA2CUDA the results look promising and we want to expand the build-in optimization procedures to achieve even better results and support more parallelization patterns.

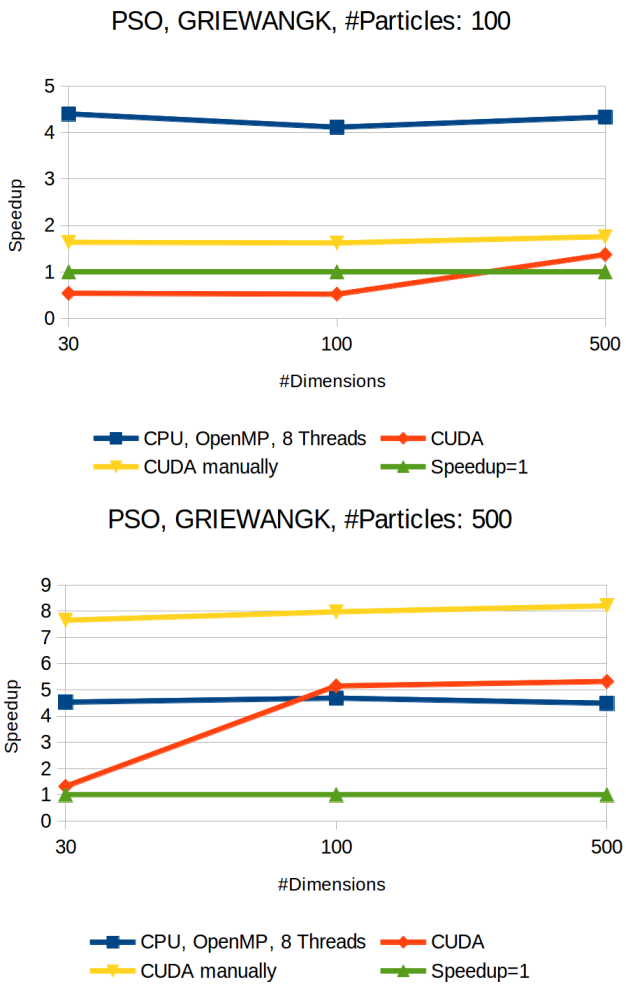


Fig. 7: Particle Swarm Optimization - Griewank Function

V. Conclusion and Future Work

We have shown in our paper a novel approach for half-automatically parallelization of PBAs using CUDA. The PBA2CUDA framework performed in our evaluation scenarios almost as well as a manually optimized CUDA version. We are working on expanding the framework and on making it more user friendly. At the moment, OpenMPC directives have to be inserted in the source code defining the parallel parts and determining on which GPGPU memory the data has to be transferred. We are working on a procedure to simplify these directives and to automatically determine the correct memory for the data. Furthermore, we are working on a decision process based upon speedup analysis mapping the source code on the correct hardware (multi-core/GPGPU) based upon the best predicted speedup automatically. Additionally, we are working on further automatically performed optimization techniques for PBAs similar to the Reduction Kernel shown in this paper.

References

[1] L. Dagum and R. Menon, "Openmp: an industry standard api for shared-memory programming," *Computational Science & Engineering, IEEE*, vol. 5, no. 1, pp. 46–55, 1998.

[2] M. Wolfe, "Implementing the pgi accelerator model," in *Proceedings of the 3rd Workshop on General-Purpose Computation on Graphics Processing Units*. ACM, 2010, pp. 43–50.

[3] R. Reyes, I. López-Rodríguez, J. J. Fumero, and F. de Sande, "accull: An openacc implementation with cuda and opencl support," in *Euro-Par 2012 Parallel Processing*. Springer, 2012, pp. 871–882.

[4] S. Bihan, G.-E. Moulard, R. Dolbeau, H. Calandra, and R. Abdelkhalek, "Directive-based heterogeneous programming–a gpu-accelerated rtm use case," in *Proceedings of the 7th International Conference on Computing, Communications and Control Technologies*, 2009.

[5] S. Lee and R. Eigenmann, "Openmpc: Extended openmp programming and tuning for gpus," in *Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE Computer Society, 2010, pp. 1–11.

[6] T. D. Han and T. S. Abdelrahman, "hi cuda: a high-level directive-based language for gpu programming," in *Proceedings of 2nd Workshop on General Purpose Processing on Graphics Processing Units*. ACM, 2009, pp. 52–61.

[7] A. Leung, N. Vasilache, B. Meister, M. Baskaran, D. Wohlford, C. Bastoul, and R. Lethin, "A mapping path for multi-gpgpu accelerated computers from a portable high level programming abstraction," in *Proceedings of the 3rd Workshop on General-Purpose Computation on Graphics Processing Units*. ACM, 2010, pp. 51–61.

[8] M. Mitchell, *An Introduction to Genetic Algorithms (Complex Adaptive Systems)*, third printing ed. A Bradford Book, Feb. 1998. [Online]. Available: http://www.amazon.com/exec/obidos/redirect?tag=citeulike07-20\&path=ASIN/0262631857

[9] D. E. Goldberg and J. H. Holland, "Genetic algorithms and machine learning," *Machine Learning*, vol. 3, pp. 95–99, 1988, 10.1023/A:1022602019183. [Online]. Available: http://dx.doi.org/10.1023/A:1022602019183

[10] T. V. Luong, N. Melab, and E.-G. Talbi, "Gpu-based island model for evolutionary algorithms," in *Proceedings of the 12th annual conference on Genetic and evolutionary computation - GECCO '10*. New York, New York, USA: ACM Press, Jul. 2010, p. 1089. [Online]. Available: http://dl.acm.org/citation.cfm?id=1830483.1830685

[11] ——, "Parallel hybrid evolutionary algorithms on gpu," *IEEE Congress on Evolutionary Computation CEC*, 2010. [Online]. Available: http://hal.inria.fr/inria-00520466/en/

[12] M. Parrilla, J. Ar, and S. Dormido-canto, "Parallel evolutionary computation: Application of an ea to controller design."

[13] J. Kennedy and R. Eberhart, "Particle swarm optimization," in *Neural Networks, 1995. Proceedings., IEEE International Conference on*, vol. 4, nov/dec 1995, pp. 1942 –1948 vol.4.

[14] R. Poli, J. Kennedy, and T. Blackwell, "Particle swarm optimization," *Swarm Intelligence*, vol. 1, pp. 33–57, 2007, 10.1007/s11721-007-0002-0. [Online]. Available: http://dx.doi.org/10.1007/s11721-007-0002-0

[15] Z.-h. Zhan and J. Zhang, "An parallel particle swarm optimization approach for multiobjective optimization problems," in *Proceedings of the 12th annual conference on Genetic and evolutionary computation - GECCO '10*. New York, New York, USA: ACM Press, Jul. 2010, p. 81. [Online]. Available: http://dl.acm.org/citation.cfm?id=1830483.1830497

[16] Y. Zhou and Y. Tan, "Gpu-based parallel particle swarm optimization," in *Evolutionary Computation, 2009. CEC '09. IEEE Congress on*, may 2009, pp. 1493 –1500.

[17] NVIDIA, *NVIDIA CUDA Compute Unified Device Architecture - Programming Guide*, 2007.

[18] E. Lindholm, J. Nickolls, S. Oberman, and J. Montrym, "Nvidia tesla: A unified graphics and computing architecture," in *IEEE Micro*, vol. 28, no. 2. Los Alamitos, CA, USA: IEEE Computer Society Press, 2008, pp. 39–55.

[19] D. J. Quinlan, "Rose: Compiler support for object-oriented frameworks," *Parallel Processing Letters*, vol. 10, no. 2/3, pp. 215–226, 2000.

[20] J. Vesterstrom and R. Thomsen, "A comparative study of differential evolution, particle swarm optimization, and evolutionary algorithms on numerical benchmark problems," in *Evolutionary Computation, 2004. CEC2004. Congress on*, vol. 2, june 2004, pp. 1980 – 1987 Vol.2.

A Quantitative Comparison of PRAM based Emulated Shared Memory Architectures to Current Multicore CPUs and GPUs

Erik Hansson*, Erik Alnervik*, Christoph Kessler* and Martti Forsell†
* Linköping University, Sweden
Email: erik.hansson@liu.se, erial674@student.liu.se
christoph.kessler@liu.se
† VTT, Oulu, Finland
Email: martti.forsell@vtt.fi

***Abstract*—The performance of current multicore CPUs and GPUs is limited in computations making frequent use of communication/synchronization between the subtasks executed in parallel. This is because the directory-based cache systems scale weakly and/or the cost of synchronization is high. The Emulated Shared Memory (ESM) architectures relying on multithreading and efficient synchronization mechanisms have been developed to solve these problems affecting both performance and programmability of current machines. In this paper, we compare preliminarily the performance of three hardware implemented ESM architectures with state-of-the-art multicore CPUs and GPUs. The benchmarks are selected to cover different patterns of parallel computation and therefore reveal the performance potential of ESM architectures with respect to current multicores.**

I. Introduction

To keep up with the never decreasing demand of more computation power both researchers and hardware manufacturers have turned to parallel computing [1]. One reason is that higher clock frequencies is a dead end with current technology due to power and heat constraints. For example high-end personal computers had already in 2003 clock frequencies up to 4GHz, today, ten years later they are about the same [2]. Still the current technology evolves so that more transistors, and hence more logic, can be built on a single chip, following Moore's law [2]. This has lead to the multicore trend that we face today – processor chips contain multiple processor cores that execute computational threads in parallel. Another approach used for speeding up sequential programs is to exploit *instruction level parallelism* (ILP) by executing multiple instructions in multiple functional units. Unfortunately, the amount of easily extractable ILP is limited in most applications. As the step towards exploiting massive *thread-level parallelism* (TLP) in applications now must be taken, the trade-off between ILP and TLP must be balanced cost-effectively.

It is claimed [3], [2] that current CPUs are inefficient in certain types of parallel functionalities making use of frequent inter-thread communication/synchronization and that so-called *Emulated Shared Memory* (ESM) architectures would solve these problems. ESMs use multithreading to hide the latency of the shared memory subsystem instead of caches and provide highly cost-efficient synchronization mechanisms. The motivation for shared memory emulation comes from the theoretical *Parallel Random Access Machine* (PRAM) model [4] that helps a programmer to focus on the essence of parallel computation without having to worry about data locality and various architecture/implementation-dependent properties of the machine, see [4] for more information about ESM. We show that an advanced ESM architecture, REPLICA, performs very well compared to the Intel CPU and Nvidia GPUs, even though it has not yet been provided with latest performance enhancement techniques such as SIMD instructions. In several cases the REPLICA architecture only needs to be clocked at a few hundred Megahertz to match them.

In this paper, we compare preliminarily the performance of three hardware implemented ESM architectures with a state-of-the-art CPU and GPU. The benchmarks are selected to cover different patterns of parallel computation. For the CPU and GPU we used fast library implementations of the benchmark functionalities while most of ESM the implementations were written by us and therefore potentially less optimized, see Table III.

Some related work includes [5] and [6]for SB-PRAM, [7] where XMT is compared to GPUs, and [8] compares GPUs and CPUs. Vuduc et al. in [9] also compares CPUs to GPUs for problems that are irregular, and states that a GPU is in this case roughly comparable to one or two CPUs (including the on-chip cores). [10] investigates CPU programming using OpenCL.

The remainder of this article is organized as follows; in section II we give a brief overview of the hardware architectures we compare, in section III we go through the selected benchmarks and setup we use and discuss the experimental results, and finally in section IV we state some conclusions.

II. Hardware Architectures

In order to figure out the performance potential of ESM architectures with respect to current CPUs and GPUs, we selected five different architectures for comparison – REPLICA (ESM), SB-PRAM (ESM), XMT (asynchronous ESM), Intel Xeon X5660 CPU, and Nvidia Tesla M2050 GPU.

A. REPLICA

REPLICA is a family of *configurable emulated shared memory machines* (CESM) developed by VTT Oulu, Fin-

land [3]. It is a *chip multiprocessor* (CMP) with massively hardware-multithreaded *very long instruction word* (VLIW) cores with chained functional units [11].

The chained functional units make it possible to use the result of one VLIW-subinstruction as an input to another in the same step. This allows dependencies between the subinstructions and use of explicit forwarding, and reduces pressure on general purpose registers.

Different configurations of the CMP have different numbers of processor cores, arithmetical logical units (ALUs), and memory units (MUs). The cores are interconnected to shared memory modules via a 2D multimesh network. In this paper we use a fixed number of ALUs and MUs.

REPLICA is based on the *Concurrent Read Concurrent Write* (CRCW) PRAM model; it defines an easy-to-use synchronous and deterministic model of programming capable of concurrent memory access and strict memory consistency [4]. The parallelism is homogeneous and explicit. REPLICA also has support for multiprefix operations [12]. If several threads in a group execute a multiprefix operation they all contribute to calculating the result together in parallel.

REPLICA can be switched at runtime to run in NUMA mode [13]. This can be useful for legacy programs or programs with low thread-level parallelism, however we will not evaluate it in this paper.

B. SB-PRAM

SB-PRAM was a research project to implement the PRAM model in hardware during the nineties [4]. The last hardware prototype [5] had 64 processors with 2048 hardware threads in total and 4GB of shared memory, following the *multiple instruction multiple data* (MIMD) paradigm and had uniform memory access time for all processors. The processors are connected to the memory modules using a butterfly network. The processors have an extended Berkeley-RISC instruction set. The routers in the butterfly network support concurrent reads and writes and also parallel prefix operations.

The processor was clocked at 8MHz which is a modest frequency in today's perspective. Still it would be possible to have a higher clock frequency if a higher number of threads was used to hide memory latency, this compensates for the relative gap between memory and processor speed [14].

C. XMT

XMT (eXplicit Multi Threading) architecture has some similarities with both REPLICA and SB-PRAM since it is also inspired from the PRAM model and will be implemented in hardware, at the moment there exists a FPGA based prototype [2]. The XMT can be seen as master-slave architecture, with a larger core called *master thread control unit* (MTCU) and several smaller *thread control units* (TCUs). The MTCU runs in serial and can spawn threads that will be executed on the TCUs [15]. The TCUs are divided into different physical clusters; from our perspective we see a cluster as processor or core that has several hardware threads. Each cluster has a shared set of functional units and a single port to shared memory. This implies that each thread can only run at speed c/t where c is the global clock frequency and t the number of threads per cluster.

The clusters are connected via a high-bandwidth low-latency network that is globally asynchronous locally synchronous [2]. Inter-thread communication between clusters is asynchronous and therefore does not realize the PRAM model.

The programmer can spawn more software threads than there are hardware threads, leading to both an elegant programming style and less overhead than if the application software had to explicitly make use of the fixed number of hardware threads. Unfortunately, this happens with the cost of synchronicity, making programs of using frequent synchronization expensive to execute.

D. Intel Xeon CPU

Xeon X5660 is a 2.8 GHz commercially off-the-shelf available Intel CPU with 6 dual-threaded cores with SIMD units. It has three levels of cache; 32 kB data cache, 256 kB mid-level cache per core and 12 MB last-level cache shared between the cores [16].

E. NVidia Tesla GPGPU

Tesla M2050 is a 575 MHz commercially off-the-shelf available NVidia GPU with 14 streaming multiprocessors, with totally 448 CUDA cores and 3 GB memory in total. Each streaming multiprocessor can handle up to 1536 resident threads, i.e., the maximal number of resident threads is 14×1536=21504. GPUs can profit from multithreading to hide memory latency. However, we do not consider it in this paper; instead we refer to [17].

III. Evaluation

We evaluated the performance of the architectures of Section II by measuring the execution time of four benchmarks representing different types of parallel computing patterns (see Table I) on seven configurations of the architectures (see Table II). We motivate our selection of benchmarks as follows: three of the benchmarks that we use, namely matrix matrix multiply, sparse matrix multiply and breadth first search, belong to the well-known Berkeley dwarfs [18]. Moreover, we have two irregular memory access algorithms; sparse matrix vector multiply and breadth first search, which would suit ESM well. On the other hand, prefix sum and matrix matrix multiply are regular memory access algorithms and therefore suit cache based systems better. The control flows of sparse matrix vector multiply and breadth first search are input dependent, which is not the case for prefix sum and matrix matrix multiply.

A. Setup

REPLICA benchmarks were written in the baseline REPLICA language [19], compiled in our LLVM 3.0 based REPLICA compiler capable of generating and optimizing target code for the different hardware configurations [11] and executed in an in-house developed cycle-accurate simulator modeling processors, interconnect and on-chip memories.

For SB-PRAM we used the Fork95 compiler (fcc version 2.0) and SB-PRAM simulator pramsim 2.0 assuming an ideal memory system.

Benchmark	Description	Mem. access	Control flow input dependent	Berkeley dwarfs [18]
prefix sum	Prefix sums of an array.	regular	No	No
matmul	Product of two matrices.	regular	No	Yes
smatvec	Product of a sparse matrix and a vector.	irregular	Yes	Yes
breadth first	Breach first search from a graph.	irregular	Yes	Yes

TABLE I: List of benchmarks

Configuration	#Cores	#Thrds	#FUs	Model
REPLICA-4	4	2048	10	ESM
REPLICA-16	16	8192	10	ESM
REPLICA-64	64	32768	10	ESM
SB-PRAM	64	2048	4	ESM
XMT	64	1024	4	AESM
Xeon CPU	6	6		SMP
TESLA GPU	448	21504		Hybrid

TABLE II: List of architectures tested. FUs stands for Functional Units.

For XMT we used the cycle-accurate XMT Simulator version 0.82.112 with 1024 TCUs and 64 clusters modeling the processors, interconnect and on-chip caches, and the xmtcc 0.82.0 compiler with O2 optimizations.

All tests for Intel Xeon X5660 were done using Debian squeeze with Linux kernel 2.6.32-5-amd64 (x86_64) and gcc version 4.4.5 with O2 optimizations with hyperthreading switched off. For the matrix matrix multiplication we used the highly optimized parallel library OpenBLAS stable release version 0.2.8 [20].

The tests for Tesla were done on ARCH Linux with Linux kernel 3.10.10-1-ARCH (x86_64) using nvcc Cuda compilation tools, release 5.0, V0.2.1221 [21]. For the prefix sum benchmark we used `Thrust::inclusive_scan` and for the matrix matrix multiplication we used the highly optimized CuBLAS library that comes with CUDA 5.0 Toolkit [21].

In Table III we show the benchmark implementations we used for Intel Xeon, Nvidia Tesla, XMT, SB-PRAM and REPLICA. Note that we used library implementations for all of the Berkeley Dwarfs listed in Table I for Intel Xeon and Nvidia Tesla.

B. Results

For breadth first search we used graphs from the Rodinia benchmark suite 2.4 [26]. For CPU and GPU implementations we used the target specific BFS algorithms provided by the Rodinia benchmark suite 2.4 [26]. Among our benchmarks BFS on graphs are the most irregular problems with respect to memory accesses.

Figure 1 shows selected performance results normalized as processed elements per execution time in clock cycles (and normalized to performance of Xeon for sparse matrix vector multiply since the number of elements is not meaningfully defined for sparse matrices), assuming the same clock frequency for CPU and ESMs and 4.8 times lower frequency for GPU[1]. From these results we can make the following observations:

Name	#Nodes	#Edges
graph4096.txt	4096	24576
graph65536.txt	65536	393216
graph1MW_6.txt	1000000	5999970

TABLE IV: Graphs from the Rodinia benchmark suite [26] used in our BFS benchmarking

Name	#Cols/Rows	#non-zeros
Internet	124651	207214
Lugn2	109460	492564
ASIC_680ks	682712	2329176
t2em	921632	4590832

TABLE V: Sparse matrices used from The University of Florida Sparse Matrix Collection [27].

The highest performance was obtained from the REPLICA-64 and XMT architectures except in matrix matrix multiply where the Tesla GPU achieved the best performance if data transportation time to GPU and back to CPU was not taken into account. The second best was the Xeon CPU for smaller matrices and REPLICA-64 for larger matrices. The lowest performance was provided by Tesla with data copying. The main reason for lower than expected ESM performance in this benchmark is that, while Xeon and Tesla are using highly assembler optimized library algorithms, ESMs use the standard triple-nested loop algorithm and no hand optimization. There exist faster algorithms also for ESM but due to the high amount of work we did not try to include them in this preliminary study. Due to the relatively high degree of parallelism, the GPU, which has the highest number of cores by a large margin, provides the best performance.

As expected, among the 64-core ESMs, REPLICA and XMT provide better performance than SB-PRAM in all tests. This is because SB-PRAM has a less optimized hardware architecture and less aggressively optimizing compiler than REPLICA and XMT. XMT outperformed REPLICA only in cases where the amount of TLP is limited in breadth first search and sparse matrix vector multiply. This is because REPLICA needs more threads to hide the latency of the interconnect and XMT has clever load balancing between the computational tasks. Unfortunately the former means that the silicon imple-

[1]This factor comes fromt the difference in clock speed between the Xeon CPU and Tesla GPU.

Benchmark	Intel Xeon	Nvidia Tesla	XMT	SB-PRAM	REPLICA
prefix sum	-	`Thrust::inclusive_scan` [21]	From [22]	-	-
matmul	OpenBLAS [20]	CuBLAS [21]	Based on [23]	-	-
smatvec	Sparse Library (SL) [24]	CUSP [25]	Based on [23]	-	-
breadth first search	Rodinia [26]	Rodinia [26]	-	-	-

TABLE III: Implementations used for the different architectures. - means that we implemented it ourselves for this study.

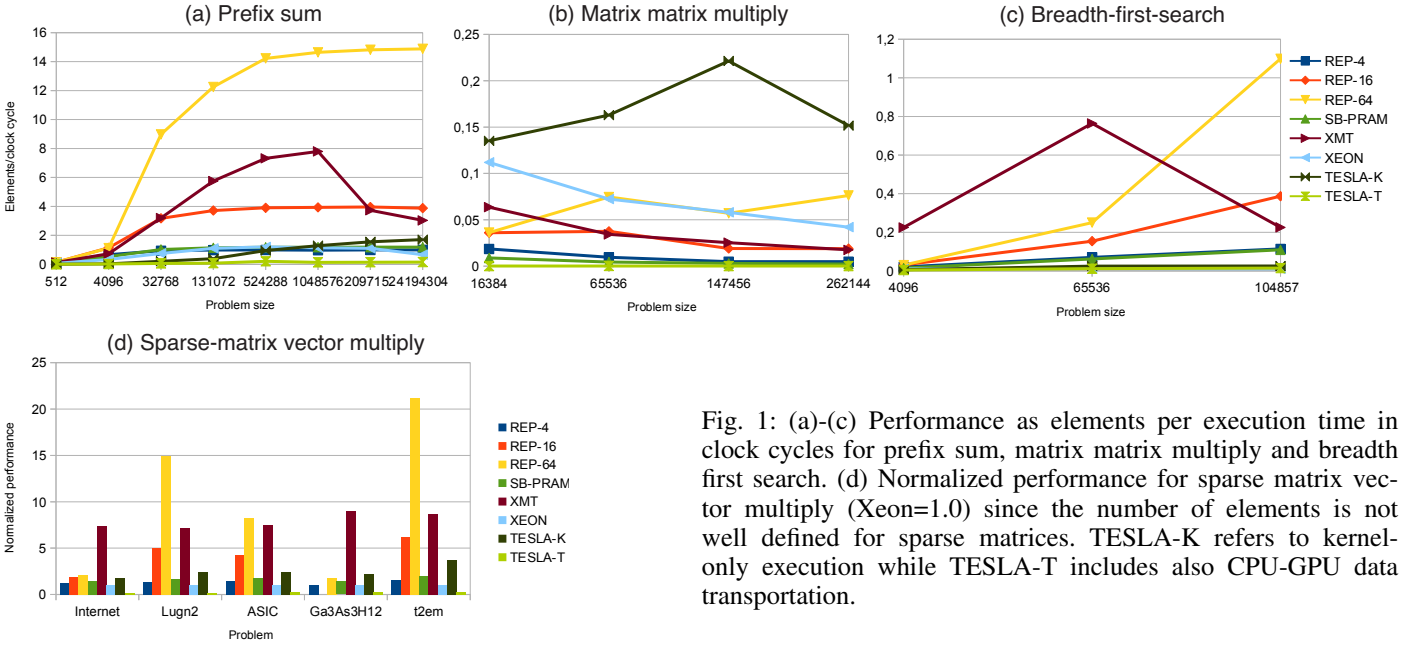


Fig. 1: (a)-(c) Performance as elements per execution time in clock cycles for prefix sum, matrix matrix multiply and breadth first search. (d) Normalized performance for sparse matrix vector multiply (Xeon=1.0) since the number of elements is not well defined for sparse matrices. TESLA-K refers to kernel-only execution while TESLA-T includes also CPU-GPU data transportation.

mentation of XMT would have lower clock rate although they are assumed to be the same in this study. Xeon CPU with only a fraction of cores of Tesla had higher performance in small prefix sum and breadth-first-search benchmarks if the plain kernel execution time is taken into account and in most benchmarks if the data transportation time to GPU and back is included.

We also considered the necessary clock rate of REPLICA architectures to match the performance of CPU and GPU (see Figure 2). From these results we can make the following observations:

The trend for prefix sum is that the larger problem sizes we have, the lower clock frequency is needed by REPLICA to match CPU and GPU, from 3.52GHz down to less than a MHz.

Matrix-matrix multiply needs around 6 GHz for a 16-core REPLICA to match the CPU for the largest problem size, and up to 22GHz to match the GPU if transfer time to the GPU is not considered. The overall lowest frequency needed to match the CPU for matrix-matrix multiply is 1.6GHz.

For sparse matrix-vector multiply, the needed clock frequencies range between 35MHz to 2.3GHz to match CPU and GPU.

For BFS the lowest clock frequency needed was 27MHz to match the Xeon CPU, using a 64-core REPLICA for the largest graph. On the smallest graph the four core REPLICA gave the worst result; 1.3GHz to match Xeon CPU.

In the case of matrix-matrix multiply we only show the comparison against the Xeon CPU in Figure 2. The reason is that if we consider the data transfer time to the GPU the needed REPLICA clock frequency is just a few Megahertz, while it is up to hundred GHz without transfer time.

In Figure 3 we show the speedups that we get if we clock the REPLICA at 2.0 GHz; for all benchmarks except matrix-matrix multiply we get a speedup for the different REPLICA versions. It ranges from 0.86x up to 102x. The lowest speedup (slowdown) among these three benchmarks was for sparse-matrix-vector multiply in the case of the smallest matrix (internet) when we compare the REPLICA four-core version to the Xeon CPU (which has 6 cores and is clocked higher).

Finally we compared REPLICA to an otherwise similar system but having an ideal shared memory (so-called PRAM configuration). The slowdown for REPLICA was less than 1% for prefix sums calculation. This consideration could not be done for the other ESM architectures since XMT does not define the PRAM model and the SB-PRAM results were already PRAM conformant since the simulator did not model the interconnection network[2].

REPLICA has a scalable multi-mesh network and has been estimated to be clockable at least at 2 GHz with an old 65 nm technology. Estimating the maximum clock frequency, needed silicon space and resulting power consumption of ESMs with current technology is beyond the scope of this paper.

[2]Note that [5] reported a similar slowdown of 1-2% measured on the SB-PRAM prototype compared to the pramsim simulated execution times.

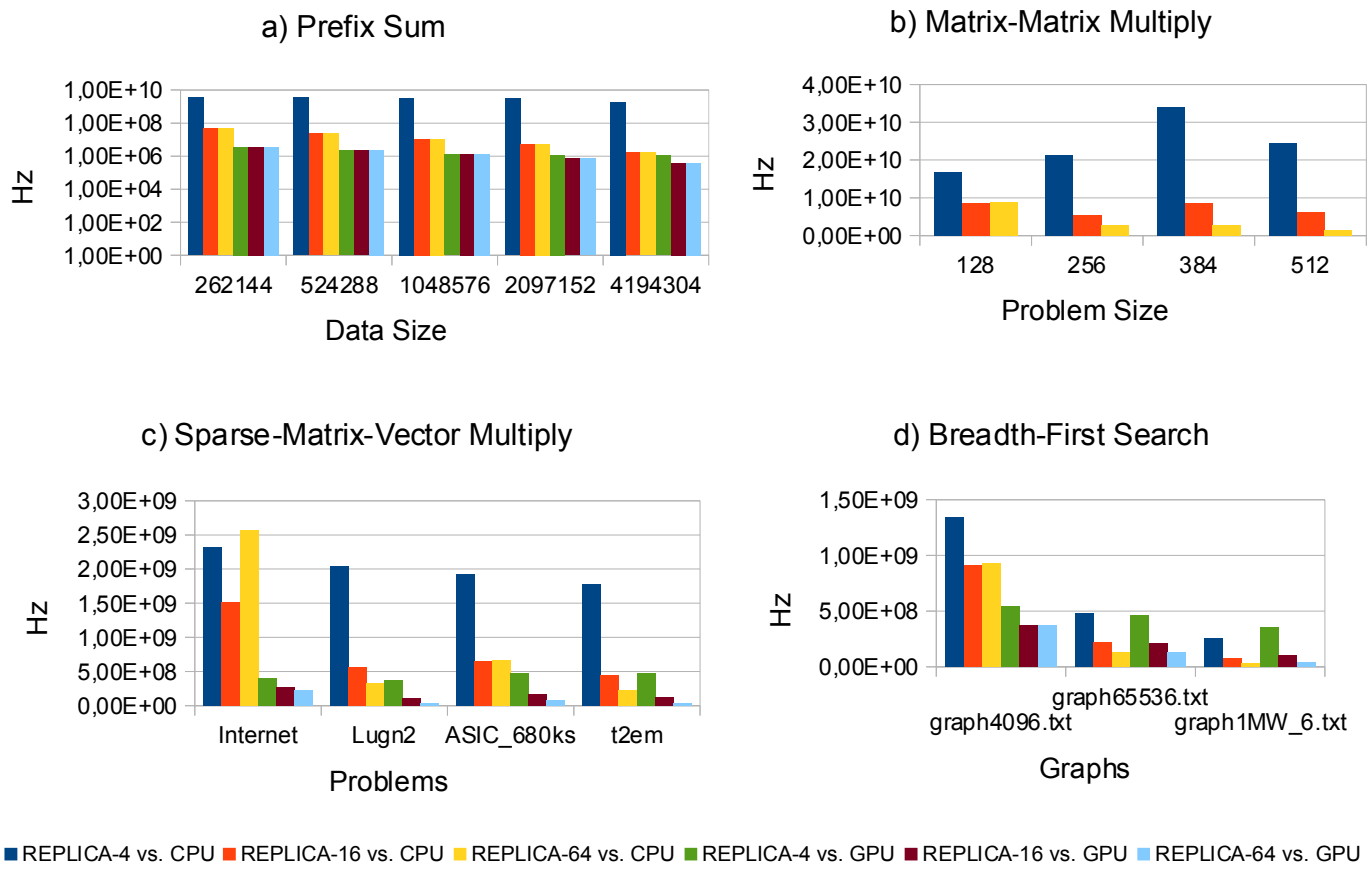


Fig. 2: Clock frequencies needed for REPLICA CMPs to match the performance of Xeon CPU and Tesla GPU. Note that Figure a) Prefix Sum has a logarithmic scale on the Y-axis.

IV. Conclusions

We have done a preliminary quantitative comparison of ESM architectures against off-the-shelf CPUs and GPUs by benchmarking. It is commonly known that some types of algorithms, based on their data access scheme, suit some type of architectures better than others. Here we have chosen different benchmarks with large variation to actually avoid biasing some architecture. Some of our tests use both data and benchmarks that are well known, also some of our implementations for the commercially available architectures are written so they make use of state-of-the-art vendor provided libraries.

Assuming that the ESM architectures execute at the same clock rate as the CPU (4.8 times faster than the GPU), the fastest ESM architectures (REPLICA and XMT) perform way better than both CPU and GPU except in matrix multiplication where the latter use asymptotically faster and more optimized algorithms. At the same time also most programs for ESM machines were significantly simpler and shorter than those of CPU and GPU. In Listing 1 we give an example of the sparse matrix vector multiply kernel implemented for REPLICA that we used in the benchmark.

The best performance was obtained by REPLICA except with some small-size breadth first searches and sparse matrix vector multiplication by XMT and with all problem sizes by

Listing 1: Sparse matrix vector multiply kernel for REPLICA. Note that variable names that ends with _ are shared. The matrix is compressed using Compressed Sparse Row (CSR) format.

```
counter_ = _number_of_threads;
r=_thread_id;
while(r<ROWS)
{
    sum = 0;
    rowStart = row_ptr_[r];
    rowEnd = row_ptr_[r+1];
    for(c=rowStart; c<rowEnd; ++c)
    {
      sum += val_[c] * x_[col_ind_[c]];
    }
    y_[r] = sum;
    aprefix(r,ADD,&counter_,1);
}
_synchronize;
```

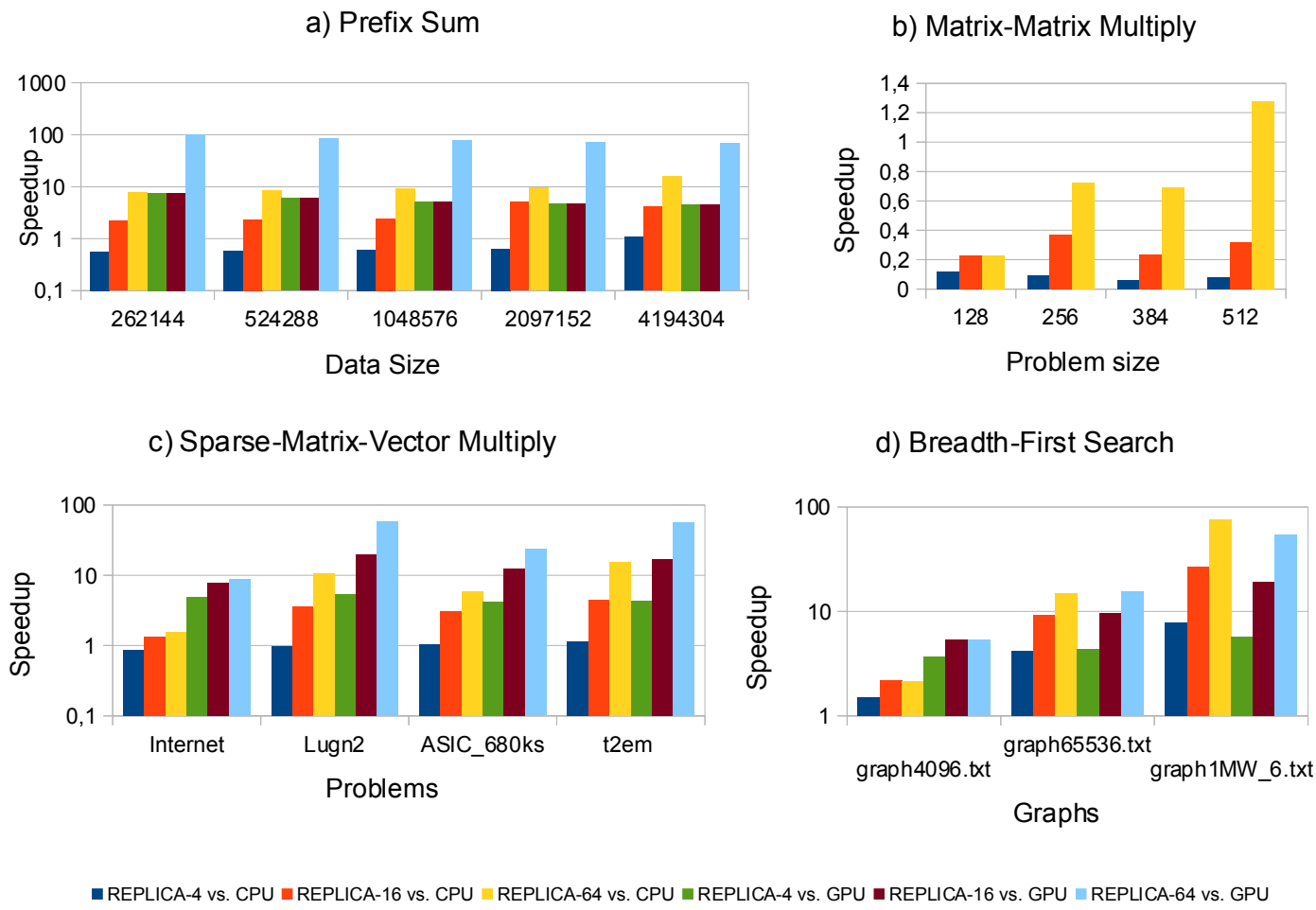


Fig. 3: Speedup for REPLICA CMPs clocked at 2.0 GHz compared to Xeon CPU and Tesla GPU. Note that Figures a), c) and d) have a logarithmic scale on the Y-axis.

Tesla GPU in matrix matrix multiplication.

This paper also reveals some potential weaknesses of the ESM architectures, including the modest architectural implementation of SB-PRAM, limited performance of XMT with large data sets, and problems of REPLICA with low-parallelism and dynamic load balancing cases. The latter ones could be potentially solved by using the NUMA mode [13] combining multiples threads to single a NUMA thread for each processor but we did not test it in this study.

Acknowledgment

This work was funded by VTT, Finland, project REPLICA. We would also like to thank Nicolas Melot and Lu Li for their support.

References

[1] H. Sutter, "The Free Lunch Is Over: A Fundamental Turn Toward Concurrency in Software," *Dr. Dobb's Journal*, vol. 30, no. 3, 2005.

[2] U. Vishkin, "Using simple abstraction to reinvent computing for parallelism," *Commun. ACM*, vol. 54, no. 1, Jan. 2011.

[3] M. Forsell, "REPLICA project site," http://www.vtt.fi/sites/replica/?lang=en, [Online; accessed 11-September-2013].

[4] J. Keller, C. Kessler, and J. L. Träff, *Practical PRAM Programming*. New York, NY, USA: John Wiley & Sons, Inc., 2000.

[5] W. Paul, P. Bach, M. Bosch, J. Fischer, C. Lichtenau, and J. Röhrig, "Real PRAM programming," in *Euro-Par 2002 Parallel Processing*, ser. LNCS, B. Monien and R. Feldmann, Eds. Springer Berlin, 2002, vol. 2400.

[6] R. Dementiev, M. Klein, and W. Paul, "Performance of mp3d on the sb-pram prototype," in *Euro-Par'02 Parallel Processing*, ser. LNCS, B. Monien and R. Feldmann, Eds. Springer Berlin Heidelberg, 2002, vol. 2400.

[7] G. Caragea, F. Keceli, A. Tzannes, and U. Vishkin, "General-purpose vs. gpu: Comparison of many-cores on irregular workloads," in *Proceedings of the Second Usenix Workshop on Hot Topics in Parallelism*, 2010.

[8] V. W. Lee, C. Kim, J. Chhugani, M. Deisher, D. Kim, A. D. Nguyen, N. Satish, M. Smelyanskiy, S. Chennupaty, P. Hammarlund, R. Singhal, and P. Dubey, "Debunking the 100x GPU vs. CPU myth: an evaluation of throughput computing on CPU and GPU," *Comput. Archit. News*, vol. 38, no. 3, Jun. 2010.

[9] R. Vuduc, A. Chandramowlishwaran, J. Choi, M. Guney, and A. Shringarpure, "On the limits of GPU acceleration," in *Proceedings of the 2Nd USENIX Conference on Hot Topics in Parallelism*, ser. HotPar'10. Berkeley, CA, USA: USENIX Association, 2010, pp. 13–13. [Online]. Available: http://dl.acm.org/citation.cfm?id=1863086.1863099

[10] A. Ali, U. Dastgeer, and C. Kessler, "OpenCL for programming shared memory multicore CPUs," in *Proceedings of the 5th Workshop on MULTIPROG, in conjunction with HiPEAC*, 2012.

[11] M. Kessler, E. Hansson, and C. Kessler, "Exploiting instruction level parallelism for REPLICA - a configurable VLIW architecture with chained functional units." in *Proc. PDPTA'12*, July 2012.

[12] M. Forsell, "Realizing multioperations for step cached MP-SOCs," in *Proc. SOC'06*, 2006.

[13] M. Forsell, E. Hansson, C. Kessler, J.-M. Mäkelä, and V. Leppänen, "Hardware and software support for NUMA computing on configurable emulated shared memory architectures," in *APDCM'13, IPDPS-2013 Workshop proceedings*, May 2013.

[14] A. Formella, J. Keller, and T. Walle, "HPP: A high performance PRAM," in *Euro-Par, Vol. II*, 1996.

[15] X. Wen, "Hardware design, prototyping and studies of the explicit multi-threading (XMT) paradigm," Ph.D. dissertation, College Park, MD, USA, 2008.

[16] Intel, "Intel Xeon Processor 5600 series, data sheet, volume 1," 2011.

[17] J. Chen, X. Tao, Z. Yang, J.-K. Peir, X. Li, and S.-L. Lu, "Guided region-based GPU scheduling: Utilizing multi-thread parallelism to hide memory latency," in *Parallel Distributed Processing (IPDPS), 2013 IEEE 27th International Symposium on*, 2013, pp. 441–451.

[18] K. Asanovic, R. Bodik, J. Demmel, T. Keaveny, K. Keutzer, J. Kubiatowicz, N. Morgan, D. Patterson, K. Sen, J. Wawrzynek, D. Wessel, and K. Yelick, "A view of the parallel computing landscape," *Commun. ACM*, vol. 52, no. 10, Oct. 2009.

[19] J.-M. Mäkelä, E. Hansson, D. Åkesson, M. Forsell, C. Kessler, and V. Leppanen, "Design of the language replica for hybrid PRAM-NUMA many-core architectures," in *Proc. ISPA'12*, Washington, DC, USA, 2012.

[20] Z. Xianyi, W. Qian, and Z. Chothia, "OpenBLAS, version 0.2.8," http://xianyi.github.io/OpenBLAS/, [Online; accessed 13-September-2013].

[21] Nvidia, "Cuda toolkit," https://developer.nvidia.com/cuda-toolkit, [Online; accessed 11-September-2013].

[22] J. A. Edwards and U. Vishkin, "Brief announcement: speedups for parallel graph triconnectivity," in *Proceedings of the 24th ACM symposium on parallelism in algorithms and architectures*, ser. SPAA '12. New York, NY, USA: ACM, 2012, pp. 190–192. [Online]. Available: http://doi.acm.org/10.1145/2312005.2312042

[23] D. Naishlos, J. Nuzman, C. W. Tseng, and U. Vishkin, "Evaluating the XMT parallel programming model," *High-Level Parallel Programming Models and Supportive Environments*, pp. 95–108, 2001.

[24] A. N. Yzelman, "Sparse library," [Online; accessed 12-December-2013]. [Online]. Available: http://people.cs.kuleuven.be/~albert-jan.yzelman/software.php

[25] S. Dalton and N. Bell, "Cusp," [Online; accessed 12-December-2013]. [Online]. Available: https://github.com/cusplibrary/cusplibrary

[26] K. Skadron, "The Rodinia benchmark suite, version 2.4," https://www.cs.virginia.edu/~skadron/wiki/rodinia/index.php/Main_Page, [Online; accessed 11-September-2013].

[27] T. Davis and Y. Hu, "The University of Florida Sparse Matrix Collection," http://www.cise.ufl.edu/research/sparse/matrices/index.html, [Accessed 20-September-2013].

Evaluation of Adaptive Memory Management Techniques on the Tilera TILE-Gx Platform

Tobias Fleig, Oliver Mattes and Wolfgang Karl
Institute of Computer Science & Engineering (ITEC)
Karlsruhe Institute of Technology (KIT)
tobias.fleig@student.kit.edu,
mattes@kit.edu, karl@kit.edu

***Abstract*—Manycore processor systems are likely to be the future system structure, and even within range for usage in desktop or mobile systems. Up to now, manycore processors like Intel SCC, Tilera TILE or KALRAY's MPPA are primarily intended to use for high performance applications, utilizing several cores with direct inter-core communication to avoid access to external memory. The spreading of these manycore systems brings up new application scenarios with multiple concurrently running high-dynamic applications, changing I/O characteristics and a not predictable memory usage. Highly dynamic workloads with varying memory usage have to be utilized.**

In this paper the memory management of various manycore platforms is addressed. In more detail the Tilera TILE-Gx platform will be explained, presenting results of own evaluations accessing its memory system. Based on that, the concept of the autonomous self-optimizing memory architecture Self-aware Memory (SaM) exemplarily was implemented as a software layer on the Tilera platform. The results show that adaptive memory management techniques can be realized without much management overhead, in return achieving higher flexibility and and simple usage of memory in future system architectures.

I. Introduction

The continuously increasing integration level of CMOS devices and the limited increase of the CPU frequencies is leading to manycore systems. Up to now, these systems are mainly designed for executing high performance applications on several cores by using direct inter-core communication over shared on-chip caches or small per-core memories. So far, the connection to the system memory commonly is realized over only a small number of controllers to just one or a few external memory components. This lack in the memory system leads to inefficient memory assignment and causes congestion [1] getting worse scaling the core count or integrating heterogeneous cores. Furthermore, with the increasing number of cores, the so called memory wall [2], the difference between the uprising CPU speed and the slow external memory, also gets more and more important.

In addition, concurrently running multiple applications calls for a dynamic memory management. In most cases, an initial optimal dynamic assignment of memory regions to tasks is often not feasible, caused by data locality issues, placement restrictions and memory regions, which are already occupied by other tasks. To be able to scale the memory with the rising core count, we propose Self-aware Memory (SaM) [3] in order to approach the optimization problem of managing and assigning memory to tasks in high-dynamic application scenarios.

In this paper the memory management of different manycore platforms is addressed. In more detail the architecture and usage of the Tilera TILE-Gx platform will be explained. We present results of own measurements accessing the memory system. The first two address the access to private and shared memory. A third evaluation shows the results of message passing based direct communications between the cores. The experiences made within these evaluations confirm the restrictions of the memory access and management of current manycore architectures as well as their thereby restricted application scenario.

Based on these evaluations, the concept of the autonomous self-optimizing memory architecture Self-aware Memory (SaM) exemplary was adapted to the Tilera platform. Since modifications in the system structure and memory management in hardware are not possible for existing systems, we implemented SaM as a software abstraction layer running on top of the existing memory management and system structure. Aside from the fact that an additional software layer cannot be faster than the original HW-based memory accesses and management, the results show that adaptive memory management techniques can be realized without much management overhead, in return achieving higher flexibility and simple usage of memory in future system architectures.

The paper is organized as follows. In Section II the architecture of the Tilera TILE-Gx platform and the results of our measurements are presented. Additionally we present other current manycore systems in Section III. A short introduction to Self-aware Memory is given in Section IV. Section V and VI cover the adaptation and implementation of SaM to the Tilera TILE-Gx platform and first results of evaluations with the self-optimization mechanism on this machine. The paper concludes in Section 6 and gives an outlook regarding ongoing research.

II. Tilera TILE

For this paper the measurements of the existing memory system as well as the exemplary adaptation of SaM was done on a system using a TILE-Gx 8036 processor. Because of that and in contrast to other manycore systems, which are presented in Section III, the Tilera TILE platform is explained in more detail in this section.

The Tilera TILE-Gx processors are a group of commercially available manycore processors and a follow-up of the MIT RAW project [4]. As well as the Intel SCC [5] the Tilera

systems use a tiled architecture and are designed to execute parallel applications like streaming applications, which mainly communicate directly between the cores, therefore using a shared cache [6]. Access to the external memory, depending on the core count of the processor, is achieved over one to four memory controllers. In 2008 the first Tilera processor TILE64 was published. Since then the manycore architecture was extended with the TILEPro and the TILE-Gx which is used in the following evaluations. There are TILE-Gx processors available with 9, 16, 36 and 72 cores up to now.

Eponymous for this architecture are the processor cores, called tiles. By using several small VLIW processors to build up identical cores, arranging them in a regular grid and connecting them over a network-on-chip (NoC), a high amount of these similar tiles could easy be combined to one processor.

The NoC connects all tiles of the system among themselves as well as to the external memory modules and further I/O. In order to achieve a high parallelization, the NoC consists out of 5 independent nets. Each tile is directly connected to its 4 surrounding adjacent tiles over a dedicated switch, which handles the redirection of the data packages over one of the 5 networks. Each of these networks has a special purpose - 3 of them are reserved for memory accesses, the cache management and coherency and their usage is transparent for the programmer. The 2 other ones are used for data transfer to the external I/O and direct inter-core connections. In the following evaluation and implementation the User Data Network (UDN) was used for the inter-core communication, because it is freely usable by the programmer.

A. Memory Architecture and Management

The memory of the TILE-Gx architecture is made both for shared memory and message passing, providing the programmer more flexibility and covering a bigger field of applications. In the standard operation mode a Linux operating system is used to provide an abstract system access with shared memory programming. In addition it is possible to pin tasks to distinct cores, directly exchanging message over the UDN. The Bare Metal Environment (BME) mode is available as a third mode, in which no normal operating system (and its libraries) runs any longer and the user has exclusive control over the whole system.

The compute cores are arranged in a grid and connected with a regular mesh network. Each tile has its own L1 and L2 cache. The memory controller are connected at the border of the grid to some individual tiles, depending on the specific processor version. For this paper we had access to a version with 36 cores and 2 memory controllers, each connected to the network on 2 positions. As can be seen in Figure 1 the left one is connected to the tiles with the numbers 6 and 24, the right one to the ones with the number 11 and 29. Thus, the next memory controller is accessible over a maximum of 3 tiles, a distinct controller with 6 hops.

A memory page is always administrated by one single tile, the so called home tile. Memory accesses of other cores to this page are first routed to the responsible home tile and answered using its L2 cache. In case of a miss the data is first updated in the L2 cache of the home tile and then sent to the requesting core. Accessing an already cached value therefore can be answered without an external memory access. But in case of cache miss the data is not directly transfered to the requesting core. This behavior is described as L3 cache by Tilera and is realized in a transparent way in hardware and the hypervisor.

Cache coherency for the pages is also guaranteed by the home tile, which keeps a list of all tiles which may have local copies in their caches. In case of a modification, invalidation messages are sent to all listed tiles.

B. Measure Methodology

The measured time periods are very short, about some hundreds CPU cycles. With that a high-level timing method is not usable, due to their limited resolution, unclear processing times or a context switch to the kernel. A method with constant run-time, high resolution and as low as possible influence on the run-time behavior was needed for our evaluations.

We achieved this with two preprocessor macros. In the first, which is called at the start of the measurement, two variables are declared, which are only used in the second macro. Then in a single machine code instruction, which is always processed in a single CPU cycle, a value of the special purpose register containing the program counter is saved in the normal register (r30). We block the register r30 using a compiler flag. This is done to abstain it from the clobber list, which holds all multiply used registers, due to efficiency reasons, so that the register value can be changed without saving/restoring it before. This is usable for our measurement, but maybe not within real systems.

After the following program code which should be measured, the second macro is called. In this, the program counter is saved into another register again with a single machine code instruction. After moving the register values to the previously declared variables the duration can be calculated. The second register doesn't have to be blocked, because the measured program code is finished at that time and an influence on the run-time is not given any more. With these two macros the measurement method has a constant run-time and no context switches or jumps are needed.

In the normal mode of the Tilera system, the operating system often disturbs the measurements e.g. by the scheduling. In addition the network is used by other tasks. To ensure that the measurement is not disturbed we used the so called dataplane modes. With it, running applications on a core are no longer interrupted, and only 1 core has to run Linux for the management and ssh connections to the systems. We used core 0 for that, because it is neighbor to a tile, which is connected to the memory, and can access the left and in this evaluation unused external memory without disturbing the rest of the network.

In the following the results of evaluations using 3 different memory usage scenarios are presented.

C. Private Memory

The first scenario covers access times to private memory. We reserved a segment as private memory in the right memory module and then access it from all cores. The segment is set

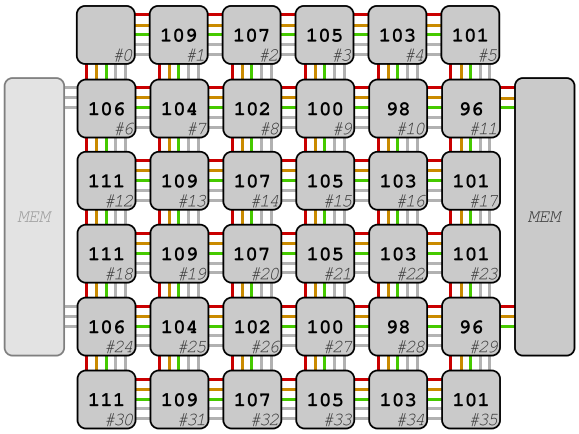


Fig. 1. Minimal access times in cycles for read accesses to private memory

as uncachable, so the data is not buffered on core side and repeated measurements always get the same results.

After the memory assignment we accessed from each core 1,000,000 times a defined memory value. Every time the duration is measured and out of all the minimum, maximum and the mean calculated. For core 0 no results are available because this core is used to run the operating system. Of special interest are the minimal access times as can be seen in Figure 1, because they represent the access times without any disturbance e.g. by other messages in the network. The repeated measurement allows us to increase the probability that for each core the minimal time was measured at least once. This minimal access time is determined by the hardware and cannot be undercut by a software based solution. The influence of the distance between the core and the memory can be seen in the printed values in Figure 1. Routing in horizontal direction always needs 2, in vertical direction 5 cycles. This forms a contrast to the official documentation, in which the transfer cost should be similar independent of the direction the data is routed to.

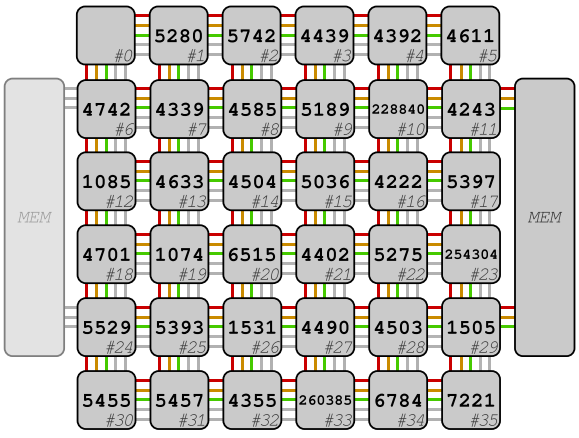


Fig. 2. Maximal access times in cycles for read accesses to private memory

The results also show that the distance to the data source is not negligible – for the core with the highest distance, 13,5 % of the access time are due to the network transfer. This percentage will increase with an increasing core count. In addition these times are only minimal access times, collected while a single core had exclusive access to the network and memory without any disturbance. Figure 2 presents the values for the maximal access times of the same evaluation. These values are collected also with only the test application running on only one core, maybe disturbed by some message of core 0. With applications running concurrently on all cores, the maximal values will be still higher. In real systems with a high load, the minimal values are not reachable and even the average and maximal values will be much higher.

D. Shared Memory

The second scenario covers access times to shared memory. This is the recommended mode of Tilera for the most jobs [7]. On these grounds the NoC is designed for high performance, providing more bandwidth to the automatically managed memory network than to the network which is freely usable by the programmer.

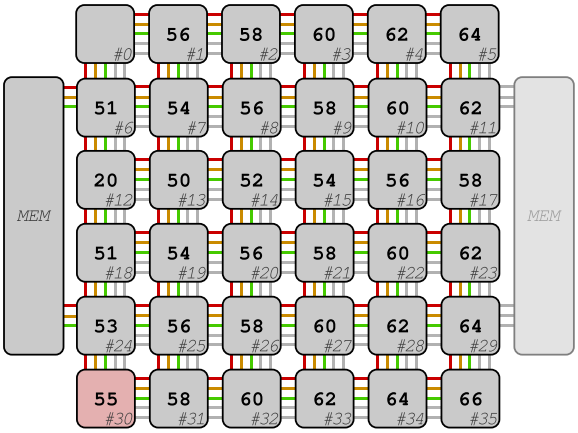


Fig. 3. Minimal access times in cycles for read accesses to shared memory

For the evaluation we used common memory, a shared memory variant of Tilera. First we assigned shared memory from a predefined tile (tile #30). Then the process forks processes to all other tiles – except core 0 with the operating system. In each measurement round the home tile changes the value in the shared memory, which invalidates the caches of the other tiles. Then all other tiles access the memory and measure their access time. To not disturb each other the tiles access the memory in a sequential order. With that, we always get comparable results, because the values cannot be taken out of the local caches but have to be received over the network. The minimal access times are shown in Figure 3, obtained from repeated measurements, in which also the maximum and mean was calculated.

As can be seen the access times to shared memory are shorter than at private memory. A reason for that is that the usage of caches cannot be completely prevented, some buffers like the TLB for the address translation are still used. According to the documentation, the data values are always loaded from the cache of the home tile of the memory segment (here tile 30). The measured values suggest, that the home tile is involved at all. In fact the tiles directly access the used left memory controller. This behavior could not be declared, accessing the value over the cache of the home tile would be much faster.

In Figure 3 tile #12 seems to be an aberration. This access time is only possible in case of a cache usage. Also the results have been the same for minimal and maximal values. A repeated evaluation showed, that sometimes some other tiles or not a single one could also have these values in a non reproducibly way.

E. Message Passing

The third scenario covers times for message passing between tiles. Such messages are sent over the previously described User Data Network (UDN). Neither the operating system nor any standard program make use of this network. Therefore, it is freely usable by the programmer and one has absolute control over every single message. This measurement is of special interest because the SaM implementation presented in Section V primarily relies on the UDN.

For these measurements, tile #1 acts as a server. Whenever a message is received, it is returned to the sender immediately. We measured the time between sending a message to tile #1 and receiving the answer on all other tiles. Again, the measurement is repeated multiple times (1,000,000) to increase the probability of finding the hardware inherent minimum. Figure 4 presents the results.

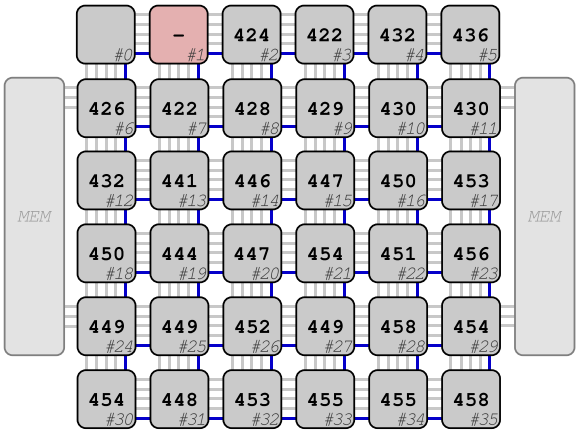


Fig. 4. Minimal round-trip-time in cycles, echo from tile 1

In general, the measured times do increase as expected with the distance to tile #1. However, there are some minor fluctuations. We expect these to be caused by the implementation of the API used to access the network. An example is active waiting for incoming messages in a loop.

F. Measurement Validation

As declared in the former sections, the measured memory access times strongly depend on the position of the tile in the grid and its assigned and accessed memory modules. As expected for currently available manycore systems, the access times are at a minimum 13,5 % higher for the farthest compared to the nearest tile. Moreover, the value for this slowdown can only achieved, when the whole system and network can be exclusively used by a single core. In real application scenarios, this actually will never be the case. With rising core count the slowdown instead will rise to a much higher level due to mutual interference using the same network or in accessing the same memory component.

III. Related Work

In addition to the already presented Tilera TILE architecture, we outline the architecture of some current manycore systems and their memory management in this section. As depicted in the following current manycore systems are not used as a scaled general purpose processor for flexible use with dynamic application scenarios. Most often they are designed to fulfill special needs and are used as co-processors or within the optimized and specialized high performance computing area using distinct parallel programming models. So, too, the memory management and accessibility often is provided in a restricted way. In the following we point out these specialization and accompanying restrictions.

A. RAMP Blue

RAMP Blue [8] is a FPGA-based multicore processor, developed at the University of California. Based on 84 Xilinx FPGAs, 1008 simulated compute cores could be achieved, running 12 MicroBlaze CPUs on each FPGA. Each core has 1 GB of exclusive memory on external memory modules, for usage with message passing. As shown in Figure 5 the system

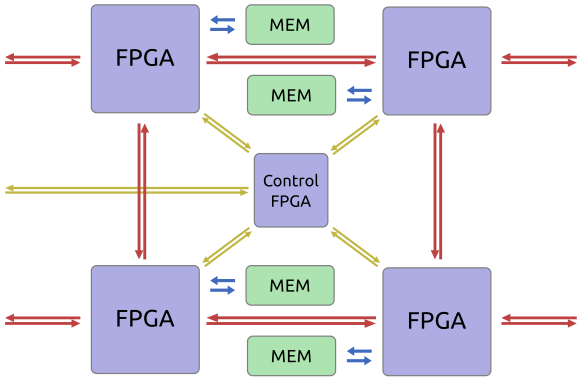


Fig. 5. Structure of a RAMP Blue board (simplified)

contains a multistage network. A control FPGA and 4 FPGAs are aggregated on a board, each connected over a 5 Gb/s ring bus. Multiple boards are connected via 10 Gb/s Ethernet in a 3D mesh network, the system uses processors without MMU running an adapted Linux (uClinux). Evaluations using benchmarks pointed out, that the overall system performance is limited by the high communication cost due to a software-based manual send and receive mechanism.

B. KALRAY MPPA

KALRAY's MPPA (Multi-Purpose Processor Array) manycore [9] is designed mainly as a dataflow architecture. The VLIW cores of the processor are grouped to clusters, containing a system core and integrated memory. The MPPA can be programmed by a c-based parallel dataflow model or with posix C/C++, which enables threading within a single cluster. The external main memory is connected over only two memory controllers, which also leads to differing memory access times.

C. Intel SCC

The Intel SCC (Single-chip Cloud Computer) [5] is the result of a project to evaluate different challenges of future

manycore architectures, like a tiled architecture, network-on-chips, communication structures and programming models. Its

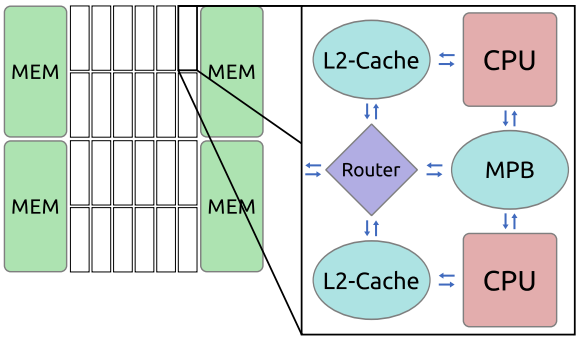


Fig. 6. Intel SCC Structure

main purpose lies in executing message passing applications, which communicate over the Message Passing Buffer (MPB), a distinct per-core cache-like memory. Access to the external connected memory is realized over four memory controllers. An initial memory assignment has to be manually done in advance of starting the system and executing applications. Depending on the locality of the used compute core, the access times to the external memory modules strongly differ. As shown in Figure 6, it consist out of 48 small Pentium 54C based cores, grouped on tiles, consisting of 2 cores, caches, a Message Passing buffer (MPB) and a router. The four external memory modules are each pre-assigned to 6 tiles and can be accessed over a connection. In addition to its main message passing purpose, an operation mode with shared memory is designated, but with no hardware cache coherency.

D. Intel Larrabee

Intel developed Larrabee as an architecture for high performance graphics cards. As Figure 7 shows, x86 processor cores were interconnected with a ring bus, recreating a pipeline structure commonly found in graphics cards. Each core has access to a part of a shared L2 cache that implements cache-coherency in hardware [10]. Larrabee was designed as a consumer-grade graphics card for computer games. Prototypes showed a near-linear speedup in frame rate with the number of cores. By using

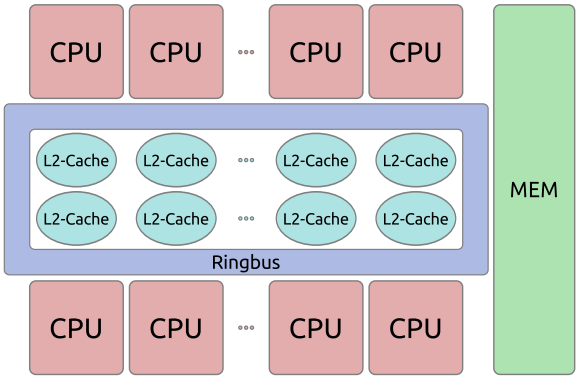


Fig. 7. Structure of Intel Larrabee

general-purpose CPUs instead of specialized vector-processing cores, Intel attempted to provide more flexibility to developers, ultimately resulting in higher frame rates. Because competitors performance could not be achieved [11], Intel stopped the graphics card project and released a Larrabee-based coprocessor card for high performance computing, named Xeon Phi.

IV. Self-aware Memory

Self-aware Memory (SaM) [3] mainly represents a memory architecture, which enables self-management of system components to build up a decentralized system architecture without a central management instance as a single point of failure. The main goal of SaM is to develop an autonomous memory subsystem for increasing the overall system reliability, flexibility and adaptability. This is crucial in upcoming computer architectures.

With SaM the individual memory modules act as independent units and are no longer directly assigned to a specific processor. SaM acts as an distributed and extended memory management unit and controls memory allocation, access rights, and ownership in a distributed manner. Figure

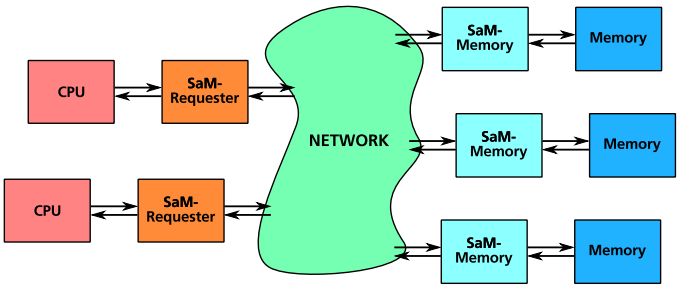


Fig. 8. Distributed SaM structure with assigned management components

8 depicts the structure of SaM. Due to this concept, SaM is constructed as a client-server architecture in which the memory modules offer their services (i.e., store and retrieve data) to client processors. The memory is divided into several autonomous self-managing memory modules, each consisting of a management component called SaM-Memory and a part of the physical memory. The SaM-Memory is responsible for handling access to its attached physical memory, administration of free and reserved space, as well as mapping to the physical address space. As a counterpart of SaM-Memory, the so called SaM-Requester augments the processor with self-management functionality. The SaM-Requester is responsible for handling memory requests, performing access rights checks, and mapping of virtual address space of the connected processor into the distributed SaM memory space. To enable and accelerate the management, tables and small caching structures are used.

The SaM components transparently realize virtual to physical addresses translation. Access to shared memory is enabled by integrating efficient synchronization techniques [12]. Memory coherency accessing shared memory regions is handled by the memory system guaranteeing the TM principles atomicity, consistency and isolation in a combined HW and SW approach. This provides the programmer an easy way for accessing shared memory and an abstract view of the memory resource.

In addition to that, research was done to establish a POSIX-like thread model allowing thread creation and management as well as allocation of compute nodes without a central management instance [13].

To increase the adaptivity of SaM to high-dynamic application scenarios, a decentralized self-optimization mechanism was integrated [14], [15]. The concurrently ongoing self-optimization is achieved with a five-stage cycle, containing the stages decentralized monitoring and data preprocessing, data analysis, optimization algorithms, decentralized consensus building, and the actual optimization. With this a decentralzed optimization approach for the memory assignment can be employed with only a small overhead of additional messages over the on-chip network.

V. Implementation

Up to now there are 3 different evaluation prototypes for Self-aware Memory. The most common prototype is a SystemC-based simulation [3], [14], [15], which easily can be parameterized and adapted to several test scenarios and system structures. With this, the memory management mechanism and the self-optimization process was evaluated and developed. In addition a coarse-grained implementation using several FPGA boards [13], [12], each representing a CPU or memory component, connected over Ethernet is available. The third prototype exists as a SW daemon, running on normal PC and redirecting memory access to other nodes, which also can be connected to the FPGA-based version via Ethernet. The goal of this work was to adapt and implement the concept of the Self-aware Memory to the Tilera platform to exemplarily demonstrate a high-dynamic and adaptive memory management on a existing manycore system.

A. Tracing

This work is motivated to enable a flexible memory management for high-dynamic application scenarios. Up to now there are no predefined benchmark scenarios for manycore systems available. To simulate these dynamic application scenario, we use a collection of memory traces we got from several benchmarks. This allows us to replay exactly the same scenario several times with changed parameters. Along with that it enables us, to easily run different evaluations with variable program and memory access phases. For each tile representing a CPU, an application scenario using a sequence of traces is provided.

B. Scenarios

The implementation on Tilera is also parameterizable to enable an easy evaluation of different application scenarios. The scenario definition is provided by a included file in which all important parameters are defined. Every tile can be configured as SamRequester or SamMemory component. The size of the memory nodes can be configured as well as the size of the used management tables. For each tile representing a CPU, an application scenario using a sequence of traces is provided. For the optimization cycle, the parameters of the decentralized monitoring are provided, e.g. the radius of the broadcast in which the status messages are exchanged. A bigger radius leads to a more global optimization knowledge, but also results in a higher overhead and additional messages. Associative counter arrays are used to arrange and pre-validate the collected information. The threshold of these arrays, which is used to launch the analysis stage of the cycle, as well as the used optimization algorithm can also be configured.

C. System Configuration

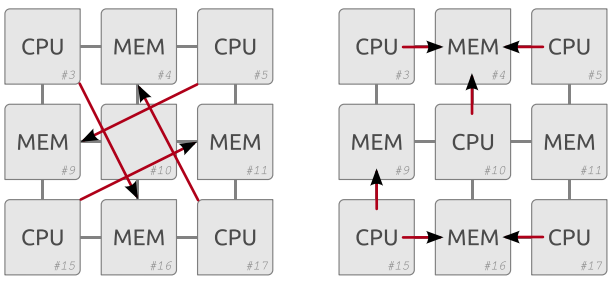
For the communication between the tiles we choose the previously addressed User Data Network (UDN). In the decentralized design of Self-aware Memory the nodes do not have prior knowledge of the system structure and are independent of a distinct network structure. Because of that, for the ongoing decentralized monitoring the status messages are exchanged using broadcasts with a distinct radius. The UDN doesn't support broadcasts, so this behavior is simulated in sending out messages to all neighbors in the regular grid. The implementation stays as software abstraction layer on top of the existing system. Therefor it adapts the existing memory management. In the presented evaluation we implemented the SaM layer as single tasks, each running exclusively on a tile. The memory accesses are blocking, so only one access can be done at one time. This restriction also affects the memory side in which only one message can be handled. So no concurrent memory accesses and monitoring exchange with neighbors are supported up to know.

D. Optimization

Every tile periodically sends out status messages to its neighbors. The content of these messages is depending on the chosen optimization goal. In the simplest way these messages are used to discover the system structure and the distance between the neighbors. The thereby discovered system view can also be resent to other neighbors, so that the total view of the system grows gradually. If a threshold of an associative counter array is exceeded, the optimization algorithm is called, which then calculates an optimization advice. An optimization advice contains a list of possibly exchanged memory segments. This advice is sent to the involved nodes which evaluate the advice and agree or reject the advice. As the last step the actual optimization, normally an exchange of memory segments, is executed.

VI. Evaluation

In this paper we present first results of the optimization cycle on the Tilera system, deploying as algorithms a locality optimization and a load compensation. Two scenarios are shown in Figure 9. In the following tables the node number correspond to the ones in these figures. During the evaluation we encountered some bigger problems with the Tilera system, which froze the total system while using the implementation on big scenarios or big memory segment sizes. This behavior could be isolated as a possible problem in the operating system.



(a) Locality Optimization (b) Load Compensation

Fig. 9. Evaluation Scenarios

Due to this problem in the following we present first results using a reduced system of 3×3 tiles and application scenarios with reduced segment sizes and run-time. Using patched operating system versions, results of additional evaluations with increasing sizes and scenarios will be subsequently provided.

A. Locality Optimization

As a first evaluation the locality optimization as optimization algorithm is presented. With this, memory segments are moved to minimize access times to different memory components. A scenario for that can be seen in Figure 9(a) in which arrows point to the initial suboptimal memory assignment.

In our implementation the threshold overflow of the associative counter array, counting the access rate to a segment, triggers the optimization algorithm. The algorithms determines if another known memory node is more narrow to the accessing CPU node. Following this, an advice is generated and sent to the possible participants. After a positive answer the segment is directly exchanged.

TABLE I. RESULTS OF THE LOCALITY OPTIMIZATION, MEMORY NODES

Optimization	activated				deactivated			
Tiles	4	9	11	16	4	9	11	16
Messages out	27512	30156	26886	26343	27623	27623	27623	27623
Words out	55271	60514	54026	52911	55263	55263	55263	55263
I/O segments	3/4	4/2	3/4	4/4	0/0	0/0	0/0	0/0

Tables I and II present the results of the evaluation with and without optimization for memory and CPU nodes. For this quicksort benchmarks are used, concurrently accessing 17 memory segments. The monitoring sends out a status message every 20 ms and uses a radius of 2 for broadcasting these messages to its neighbors.

TABLE II. RESULTS OF THE LOCALITY OPTIMIZATION, CPU NODES

Optimization	activated				deactivated			
Tiles	3	5	15	17	3	5	15	17
Duration (10^6 cycles)	154	154	155	152	129	130	131	130
Moved Segments	4	2	4	4	0	0	0	0
Access time (cycles)	5249	5260	5298	5188	4287	4303	4308	4348

The higher values with activated optimization are due to the periodical exchange of status messages. As previously mentioned, the memory nodes cannot process these status messages and memory accesses at the same time in our implementation. This overhead can be reduced in adapting the exchange frequency. As can be seen in Table III, most of the additional costs are caused by the status messages and the transfer of the moved segments. The slightly changed values compared to Table I are due a different evaluation run. With the operating system problems we had to reduce the size of the transfered data segments. Without that, the transfer time is increased while the number of status messages is kept constant. But as can be seen in Table II the access to the new location is faster after the optimization. So with more realistic application scenario sizes and without the given limitations of the Tilera system, the optimization would be profitable and outbalance the additional management costs.

TABLE III. MESSAGE IN THE STAGES OF THE OPTIMIZATION CYCLE

Tiles	4	9	11	16
regular	15129	53901	37259	4173
busy	0	8	2	0
moved	2	2	3	3
transfer	20	20	21	21
suggestion	2	2	3	3
answer	1	6	3	0
status	70	110	70	110
Messages out	15254	54049	37361	4310
Words out	30732	108269	74951	8814
I/O segments	1/2	6/2	3/3	0/3

B. Load Compensation

As second presented evaluation, a load compensation optimization was evaluated. Segments of highly occupied memory nodes are moved to less charged ones. A scenario for that can be seen in Figure 9(b) in which arrows also point to the initial suboptimal memory assignment. Here the segments of 3 CPU nodes are initially assigned to one single memory node.

TABLE IV. RESULTS OF THE LOAD COMPENSATION, MEMORY NODES

Optimization	activated				deactivated			
Tiles	4	9	11	16	4	9	11	16
Msg. out	33459	34596	33389	37246	82869	14280	0	40966
Words out	67911	69990	67314	75305	165789	28569	0	81957
I/IO segments	5/9	7/8	9/4	7/7	0/0	0/0	0/0	0/0

As in the first example, a threshold overflow of the associative counter array is used to trigger the optimization algorithm which sends out an optimization advice. In this example most of the work of the optimization algorithm is done to evaluate the advice by the participants. The algorithms therefor compares the load of its own node and the load of the initiator. In case of a higher load an exchange is done.

TABLE V. RESULTS OF THE LOAD COMPENSATION, CPU NODES

Optimization	activated					deactivated				
Tiles	3	5	10	15	17	3	5	10	15	17
Duration (10^6)	222	223	224	190	215	209	209	209	139	138
Moved Segments	3	7	4	3	11	0	0	0	0	0
Access time	7456	7440	7512	6395	7317	6984	6974	6970	4558	4581

Duration & Access time in cycles

Tables IV and V present the results of the evaluation with and without optimization for memory and CPU nodes. During the evaluation the monitoring sends out a status message every 40 ms and also uses a radius of 2 for broadcasting these message to its neighbors.

TABLE VI. MESSAGE IN THE STAGES OF THE OPTIMIZATION CYCLE

Tiles	4	9	11	16
regular	29799	34554	36428	37334
busy	0	6	3	7
moved	9	6	7	5
transfer	72	42	61	41
suggestion	12	15	15	13
answer	14	14	13	14
status	42	66	42	66
msg. out	29948	34703	36569	37480
words out	60891	69920	73942	75482
segm I/O	4/9	8/6	7/7	8/5

As can be seen in these two tables, the CPU cycle count for the execution is more balanced with activated optimization. Instead of even unused nodes in the execution with deactivated optimization, the number of messages on memory node side are good balanced.

The values in Table VI show, just as in the first evaluation, that most of the additional costs are caused by the fine-granular transfer of the moved segments. But the optimization goal of good balanced memory loads could be achieved nevertheless. So again as in the previous example in scaling the scenarios and without the given limitations of the Tilera system, the optimization would be more profitable.

VII. Conclusion and Outlook

This paper presents evaluation of the memory management of existing manycore systems. In more detail the architecture and usage of the Tilera TILE-Gx platform was addressed. First we measured the existing memory management of the available Tilera platform, second we adapted the the concept of the autonomous self-optimizing memory architecture Self-aware Memory to it.

Our measurements on the Tilera TILE-Gx system pointed out some weak spots in current manycore architectures. As expected, the measured access times to external connected memory modules strongly depend on the position of the tile in the grid. This holds true for access to private and shared memory. The access times distinguish themselves from e.g. a minimum of 13,5 % for the farthest compared to the nearest tile accessing private memory, when the whole system and network can be exclusively used by a single core. In reality, this will never be the case in using manycore systems, even less with the promised high-dynamic application scenarios. Moreover, scaling up the number of cores will significantly rise the slowdown due to mutual interference using the same network or in accessing the same memory component.

With the adaptation and implementation of the Self-aware Memory concept to the Tilera system, we demonstrated that high-dynamic and adaptive memory management techniques are feasible in combination with manycore architectures. Modifications in the system structure and memory management of the existing hardware system were not possible for the evaluations. Therefore we implemented SaM as a software abstraction layer running on top of the existing memory management and system structure. The results of the evaluation showed that adaptive memory management techniques can be realized without much additional messages, in return achieving higher flexibility and and simple usage of memory in future system architectures. To take advantage of these mechanisms, the results and experiences have to be adapted to future manycore architectures and directly integrated in the hardware. As presented, current systems often contain multiple different networks. Moving the monitoring and management messages to an additional or currently unused network, would separate the ubiquitous management from the data transfers and make the optimization profitable furthermore.

In this paper we presented first results with reduced scenario sizes, due to the repeated operating system crashes on the Tilera system. Currently, we work on fixing these problems. Additional evaluations, scaling the number of tiles, applying bigger application scenarios and increased test run-times are the next steps on our agenda.

To further improve the results of the optimization process, aspects from machine learning in combination with program phases could be examined in the future. Adapting and evaluating the presented mechanisms to new and upcoming memory connections like 3D-stacked memory or optical connections, accompanied with a change in the system structure, will be another challenging step.

References

[1] J. Duato, "Beyond the power and memory walls: The role of HyperTransport in future system architectures," in *First International Workshop on HyperTransport Research and Applications (WHTRA)*, February 2009.

[2] W. A. Wulf and S. A. McKee, "Hitting the memory wall: implications of the obvious," *SIGARCH Comput. Archit. News*, vol. 23, no. 1, pp. 20–24, Mar. 1995.

[3] R. Buchty, O. Mattes, and W. Karl, "Self-aware Memory: Managing Distributed Memory in an Autonomous Multi-master Environment," in *Architecture of Computing Systems – ARCS 2008*, ser. Lecture Notes in Computer Science, vol. 4934, February 2008, pp. 98–113.

[4] M. Taylor, J. Kim, J. Miller, D. Wentzlaff, F. Ghodrat, B. Greenwald, H. Hoffman, P. Johnson, J.-W. Lee, W. Lee, A. Ma, A. Saraf, M. Seneski, N. Shnidman, V. Strumpen, M. Frank, S. Amarasinghe, and A. Agarwal, "The Raw microprocessor: a computational fabric for software circuits and general-purpose programs," *Micro, IEEE*, vol. 22, no. 2, pp. 25–35, 2002.

[5] M. Gries, U. Hoffmann, M. Konow, and M. Riepen, "SCC: A Flexible Architecture for Many-Core Platform Research," *Computing in Science Engineering*, vol. 13, no. 6, pp. 79–83, 2011.

[6] T. Corporation, "TILE-Gx Processor Specification Brief," May 2012.

[7] Tilera Corporation, *Architecture Overview for the TILE-Gx Series, UG130*, San Jose, CA 95134 USA, 2012.

[8] A. Krasnov, A. Schultz, J. Wawrzynek, G. Gibeling, and P.-Y. Droz, "RAMP Blue: A message-passing manycore system in FPGAs," in *Field Programmable Logic and Applications, 2007. FPL 2007. International Conference on*. IEEE, 2007, pp. 54–61.

[9] Kalray, "KALRAYs MPPA (Multi-Purpose Processor Array)," http://www.kalray.eu/products/mppa-manycore/, April 2013.

[10] L. Seiler, D. Carmean, E. Sprangle, T. Forsyth, M. Abrash, P. Dubey, S. Junkins, A. Lake, J. Sugerman, R. Cavin *et al.*, "Larrabee: a many-core x86 architecture for visual computing," in *ACM Transactions on Graphics (TOG)*, vol. 27, no. 3. ACM, 2008, p. 18.

[11] Intel Corporation, "An Update On Our Graphics-related Programs." [Online]. Available: http://blogs.intel.com/technology/2010/05/an_update_on_our_graphics-rela/

[12] O. Mattes, M. Schindewolf, R. Sedler, R. Buchty, and W. Karl, "Efficient Synchronization Techniques in a Decentralized Memory Management System Enabling Shared Memory," ser. PARS Mitteilungen GI, vol. 28, no. ISSN 0177-0454. Rüschlikon, Switzerland: Gesellschaft für Informatik e.V., May 2011.

[13] M. Schindewolf, O. Mattes, and W. Karl, "Thread creation for self-aware parallel systems," in *Facing the Multicore-Challenge*, ser. Lecture Notes in Computer Science, R. Keller, D. Kramer, and J.-P. Weiss, Eds. Springer Berlin / Heidelberg, 2011, vol. 6310, pp. 42–53.

[14] O. Mattes, "An Autonomous Self-Optimizing Memory System for Upcoming Manycore Architectures," in *Proceedings of the First Organic Computing Doctoral Dissertation Colloquium (OC-DDC'13)*, ser. Reports / Technische Berichte - Herausgeber: Fakultät für Angewandte Informatik der Universität Augsburg, no. 2013-06, 2013, pp. 4–7.

[15] O. Mattes and W. Karl, "Self-aware Memory - An Autonomous Self-Optimizing Memory System for Upcoming Manycore Architectures," in *Memory Architecture and Organization Workshop (MeAOW'13)*, ser. Embedded Systems Week 2013, Montreal, Canada, 2013.

ScaFES: An Open-Source Framework for Explicit Solvers Combining High-Scalability with User-Friendliness

Martin Flehmig
Technische Universität Dresden
Center for Information Services and
High Performance Computing (ZIH)
martin.flehmig@tu-dresden.de

Kim Feldhoff
Technische Universität Dresden
Center for Information Services and
High Performance Computing (ZIH)
kim.feldhoff@tu-dresden.de

Ulf Markwardt
Technische Universität Dresden
Center for Information Services and
High Performance Computing (ZIH)
ulf.markwardt@tu-dresden.de

***Abstract*—We present ScaFES, an open-source HPC framework written in C++11 for solving initial boundary value problems using explicit numerical methods in time on structured grids. It is designed to be highly-scalable and very user-friendly, i.e. to exploit all levels of parallelism and provide easy-to-use interfaces. Besides, the numerical nomenclature is reflected in a nearly one-to-one mapping.**

We describe how the framework works internally by presenting the core components of ScaFES, which modern C++ technologies are used, which parallelization methods are employed, and how the communication can be hidden behind during the update phase of a time step.

Finally, we show how a multidimensional heat equation problem discretized via the finite difference method in space and via the explicit Euler scheme in time can be implemented and solved using ScaFES in about 60 lines. In order to demonstrate the excellent performance of ScaFES, we compare ScaFES to PETSc on the basis of the implemented heat equation example in two dimensions and present scalability results w.r.t. MPI and OpenMP achieved on HPC clusters at the ZIH.

I. Yet Another Framework?

A wide variety of phenomena like heat transport, fluid flow, and electrostatics can be described by initial boundary value problems of the following type: Given a time interval $[t_S; t_E]$ with $0 \leq t_S < t_E$, an open, bounded domain $\Omega \subset \mathbb{R}^d$ with dimension $d \in \mathbb{N}$ and boundary $\partial\Omega$, a source $f : \bar{\Omega} \times (t_S; t_E] \to \mathbb{R}^m$, a boundary condition $g : \partial\Omega \times (t_S; t_E] \to \mathbb{R}^m$, an initial condition $\widetilde{u} : \Omega \to \mathbb{R}^m$, and differential operator F, then the task is to find $u : \bar{\Omega} \times [t_S; t_E] \to \mathbb{R}^m$ such that the following system of equations is fulfilled:

$$\begin{aligned} \partial_t u + F(u, \nabla u, \ldots) &= f && \text{in } \Omega \times (t_S; t_E], \\ u &= g && \text{on } \partial\Omega \times (t_S; t_E], \\ u(\cdot, t_S) &= \widetilde{u} && \text{in } \Omega. \end{aligned}$$

Analytical solutions of such problems exist only for rare cases. Nevertheless, engineers and scientists want to have more and more detailed approximations of these problems, resulting in a significantly increase of the memory requirements as well as the computational time. These computations can only be run in parallel. There are many software packages available which can solve these problems numerically using simple methods like finite difference methods (FDM) or more complex methods like finite elements or spectral methods (AMDiS [1], PETSc [2], and DUNE [3]). So, why should it be necessary to implement yet another framework for solving initial boundary value problems?

The answer is that the software packages like the ones mentioned above, can solve these problems by combining several numerical methods, parallelization approaches and implementation languages. But they have all been designed for more general purposes and therefore provide a large and quite complex infrastructure with a lot of objects, methods and modules which lead to long learning curves. Roughly speaking, they are kind of heavyweight. And indeed, for many initial boundary value problems it is sufficient to use simple numerical methods like the explicit finite difference method on a structured grid (e.g. for solving Maxwell's equations ([4]).

Thus, we designed the framework ScaFES ("Scalable Framework for Explicit Solvers") for explicit methods in time and space on structured grids. Instead of expanding existing software but writing ScaFES from scratch we had the opportunity to clearly focus on our design principles. These are:

- High-scalability, i.e. all levels of parallelism on current multi- and many-core architectures should be efficiently used.
- User-friendliness, i.e. ScaFES should have easy-to-use interfaces such that users can implement their numerical methods as usual. In particular, the numerical nomenclature should be reflected in a nearly one-to-one mapping and knowledge about parallelization aspects should not be required. Besides, it should be easy to build and install on a wide variety of platforms.

As a consequence, ScaFES can be used as a rapid prototyping tool to evaluate and compare different numerical approaches as well as to write high quality production code without loosing scalability and efficiency.

The presented work is organized as follows: In section II, we will discuss the design concepts of the framework. How a multidimensional heat equation problem can be solved using ScaFES will be demonstrated in section III. In section IV, we will present scalability results achieved on an HPC cluster at the ZIH for the implemented problem in the three-dimensional

case. The paper concludes in section V with a summary of the results and an outlook on further work related to ScaFES.

II. Design Concepts of ScaFES

In the following, we describe the design concepts of ScaFES, i.e. the implementation of the design principles. In order to fulfill the two principles high-scalability and user-friendliness we have chosen C++11 [5] as programming language. Since C++11 contains modern programming concepts and features like constructor delegations, class and function templates as well as STL containers, the framework allows well structured development without significant performance losses. The readability and usability of the source code was improved by using additional features of the Boost C++ libraries [6]. The framework is based on the GNU autotools [7] which are available on almost all Linux based systems in order to allow the installation of ScaFES on a high-variety of different systems. This means that the build and installation process is a combination of the usual calls to `configure`, `make`, and `make install`.

ScaFES is highly modularized. It consists of the following core components: The class templates `Grid<DIM>` and `GridSub<DIM>` for the representations of structured grids resp. sub-grids, the class template `GridGlobal<DIM>` for the representation of global grids, the class template `DataField<CT,DIM>` for the representation of physical fields, and the class template `Problem<PRBLM,CT,DIM>` for the representation of initial boundary value problems. In the following subsections, we will present these class templates and their design concepts in detail.

A. Representation of Structured Grids

The considered problems should be discretized using numerical methods which are based on structured grids. In the following, we refer to a structured grid as a uniform decomposition of a given domain $D \subset \mathbb{R}^d$ into d-dimensional hypercuboids. More precise, let $D := (s_0, e_0) \times \ldots (s_{d-1}, e_{d-1})$ the given domain with $s_p < e_p \in \mathbb{R}$ for each direction $p \in \{0, 1, \ldots, d-1\}$ and $2 \leq n_p \in \mathbb{N}$ the number of grid nodes in each direction p. As the domain D is uniformly decomposed, the corresponding grid size $h_p := (e_p - s_p)/(n_p - 1)$ is constant in each direction p. The grid nodes are numbered accordingly to their positions $i = (i_0, i_1, \ldots, i_{d-1}) \in \mathbb{N}_0^d$ in the grid. Then, for a given position $(i_0, i_1, \ldots, i_{d-1})$, the corresponding real-world coordinates $x_{(i_0,i_1,\ldots,i_{d-1})} \in \mathbb{R}^d$ are given by

$$x_{(i_0,i_1,\ldots,i_{d-1})} = (s_0 + i_0 \cdot h_0, \ldots, s_{d-1} + i_{d-1} \cdot h_{d-1}).$$

The set of all coordinates $x_{(i_0,i_1,\ldots,i_{d-1})}$ is denoted by D_h and the set of the corresponding integer tuples $(i_0, i_1, \ldots, i_{d-1})$ by $\mathcal{G}(D_h)$. Furthermore, the sets $\mathcal{G}_I(D_h)$ and $\mathcal{G}_B(D_h)$ are defined as index sets of all interior resp. boundary nodes of D_h such that $\mathcal{G}(D_h) = \mathcal{G}_I(D_h) \cup \mathcal{G}_B(D_h)$, and the total number of grid nodes is denoted by $N := \prod_{p=0}^{d-1} n_p$. All quantities are also explained in Fig. 1a. The positions of the direct neighboring nodes in direction p for a given interior grid node number $i = (i_0, i_1, \ldots, i_{d-1}) \in \mathcal{G}_I(\Omega_h)$ can be accessed using the following connectivity mapping c. Fig. 1b illustrates the mapping in two dimensions.

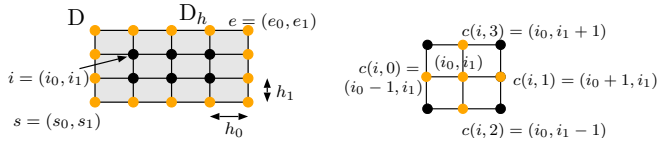


Fig. 1: (a): Two-dimensional grid D_h of a given domain $D = (s_0, e_0) \times (s_1, e_1)$ with 5×4 nodes: All inner grid nodes are colored black, all boundary grid nodes are colored orange. (b): Indices of direct neighboring nodes of a given grid node with index (i_0, i_1), accessed via the connectivity mapping c.

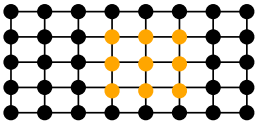


Fig. 2: Two-dimensional (base) grid of 8×5 grid nodes with a sub-grid of 3×3 grid nodes colored orange.

$$c(i; 2 \cdot p) := (i_0, i_1, \ldots, i_{p-1}, i_p - 1, i_{p+1}, \ldots, i_{d-1}),$$
$$c(i; 2 \cdot p + 1) := (i_0, i_1, \ldots, i_{p-1}, i_p + 1, i_{p+1}, \ldots, i_{d-1}).$$

Due to the regular structure of the grids, the set D_h can be completely described by the coordinates $s = (s_0, s_1, \ldots, s_{d-1})$ and $e = (e_0, e_1, \ldots, e_{d-1})$ of the domain D together with the number of grid nodes $n = (n_0, n_1, \ldots, n_{d-1})$ in each direction. In ScaFES, the set D_h is represented by the class template `Grid<DIM>`. The quantities s, e and n are given as member variables (see Listing 1). In particular, there is no need to create huge arrays storing the coordinates and the indices of all grid nodes. The space dimension d is implemented as template parameter `DIM` such that the size of all member variables is known at compile time.

```
// Number of nodes: n=(n_0,...,n_{d-1})
ScaFES::Ntuple<int, DIM> mNnodes;
// Coordinates of first node: s=(s_0,...,s_{d-1})
ScaFES::Ntuple<double, DIM> mCoordNodeFirst;
// Coordinates of last node: e=(e_0,...,e_{d-1})
ScaFES::Ntuple<double, DIM> mCoordNodeLast;
```

Listing 1: Member variables of the class template `ScaFES::Grid<DIM>`.

Subsets of the grid D_h of the following type will be referred to as so-called "sub-grids":

$$\Delta_h := \Big\{ x_{(i_0,i_1,\ldots,i_{d-1})} \in D_h : \; a \leq (i_0, i_1, \ldots, i_{d-1}) \leq b \text{ with } a, b \in \mathbb{N}_0^d \text{ as the indices of the first resp. the last node of the subset.} \Big\}.$$

Fig. 2 shows a grid and a sub-grid in two dimensions. The subset Δ_h can be described by the base grid D_h and the indices a and b of the first and last node of the sub-grid. As Δ_h is related to the (base) grid D_h, the sub-grid is represented by a sub class template named `GridSub<DIM>` of the (base) class template `Grid<DIM>` which itself represents the (base) grid D_h. The indices a and b are given as member variables (see Listing 2), the space dimension d is implemented as template parameter `DIM`.
Currently, all grids and sub-grids will be traversed lexicographically in C style, i.e. node numbers $(i_0, i_1, \ldots, i_{d-1})$ in

```
// Index of first node: a=(a_0,...,a_{d-1})
ScaFES::Ntuple<int, DIM> mIdxNodeFirstSub;
// Index of last node: b=(b_0,...,b_{d-1})
ScaFES::Ntuple<int, DIM> mIdxNodeLastSub;
```

Listing 2: Member variables of the class template `ScaFES::GridSub<DIM>`.

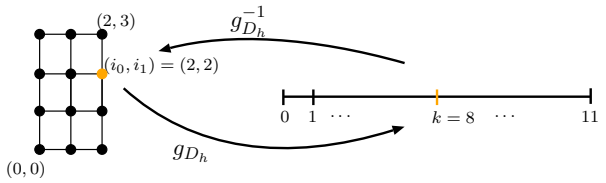


Fig. 3: Correspondents between node numbers in tuple notation (i_0, i_1) and in scalar notation k via the mappings g_{D_h} and $g_{D_h}^{-1}$ of a two-dimensional grid D_h with 3×4 grid nodes. The correspondent is explicitly shown for the node number $(2, 2)$.

tuple notation of sub-grids correspond to scalar node numbers $k \in \mathbb{N}_0$ and vice versa according to the following mappings:

$$g_{D_h}(i_0, i_1, \ldots, i_{d-1}) := \sum_{p=0}^{d-1} (i_p - c_p) \cdot \prod_{q=1}^{p} n_q,$$

$$g_{D_h}^{-1}(k) = \left(c_p + \left\lfloor k / \prod_{p=0}^{p-1} n_q \right\rfloor \bmod n_p \right)_{p=0,\ldots,d-1}$$

with $\lfloor . \rfloor$ as lower Gaussian bracket and $c \in \mathbb{Z}^d$ as index of the first node of the base grid D_h. Fig. 3 shows the correspondents via the mappings g_{D_h} and $g_{D_h}^{-1}$ for a two-dimensional grid.

For traversing through grids and sub-grids, the class templates `Grid<DIM>` and `GridSub<DIM>` each have an internal class named `Iterator`. These internal classes are designed like the iterators of the STL. Thus, users do not have to learn new patterns but can apply the iterators in the same way. Furthermore, the employment of these iterators has the advantage that the numbering of the nodes is hidden from the user and therefore, can be easily changed if necessary (see Listing 3).

```
ScaFES::Grid<DIM> gd;
for (ScaFES::Grid<DIM>::iterator it = gd.begin(),
     et = gd.end(); it < et; ++it) { // [...]
}
```

Listing 3: Application of an iterator of the class template `ScaFES::Grid<DIM>`.

The index $(i_0, i_1, \ldots, i_{d-1})$ of the current grid node can be accessed via `it.idxNode()`, the corresponding scalar $g(i_0, i_1, \ldots, i_{d-1})$ of the current tuple can be accessed via `it.idxScalarNode()`.

B. Representation of the (Discretized) Computational Domain

In order to solve initial boundary value problems using grid-based methods, the computational domain Ω has to be discretized on a given set of grid nodes, first. Usually, the number of grid nodes is very huge ($> 10^7$). This would result in a system of equations which would be too large to be solved in serial. Thus, the discretized computational domain $\Omega_h \subset \mathbb{R}^d$ (called "global grid") has to be decomposed into a given number $q \in \mathbb{N}$ of sub-grids S_k with

$$\bigcup_{k=0}^{q-1} S_k = \Omega_h$$

and the corresponding grid partitions S_k have to be distributed to the appropriate cores of the parallel hardware such that each core will work on an appropriate subset of all grid nodes, only. This so-called "domain decomposition approach" [8] fits to our needs as we have restricted the framework to structured grids. The communication in terms of messages between the cores is enabled by the Message Passing Interface (MPI), Currently, the global grid is partitioned based on the well-known RCB ("Recursive Coordinates Bisection") algorithm [9]. The global grid Ω_h is described by the grid partitions S_k, the number of partitions n_p, and the relations between the grid partitions. Due to the regular structure of the underlying grids, these relations are completely described by the identifiers and the directions of the direct neighboring grid partitions.

The type of all nodes $j \in \mathcal{G}(\Omega_h)$ of the global grid will be stored in a vector $T \in \mathbb{N}^N$ in order to distinguish if a node is a global interior one or a global boundary one:

$$T_j := 1 \quad \text{for all } j \in \mathcal{G}_I(\Omega_h),$$
$$T_j := 2 \quad \text{for all } j \in \mathcal{G}_B(\Omega_h).$$

Additionally, the type of all nodes related to a grid partition S_k will be stored in a vector $R^{(k)} \in \mathbb{N}^{M_k}$ with $M_k \in \mathbb{N}$ as the number of nodes of the grid partition S_k:

$$R_j^{(k)} := 4 \quad \text{for all } j \in \mathcal{G}_I(S_k),$$
$$R_j^{(k)} := 8 \quad \text{for all } j \in \mathcal{G}_B(S_k).$$

The values of T and $R^{(k)}$ are chosen as elements of the dual basis such that the sum can be created and values can be easily extracted via bitwise operators. Fig. 4 illustrates the different grid node types on a two-dimensional computational domain which is discretized and decomposed into four grid partitions.

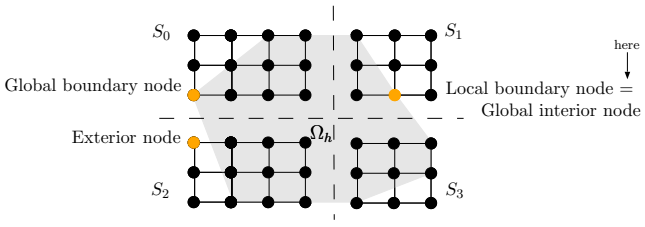


Fig. 4: Decomposition of a two-dimensional global grid Ω_h with 7×6 grid nodes into four grid partitions S_k. Local and global interior resp. boundary nodes are identified.

In ScaFES, global grids are represented by the sub class template `GridGlobal<DIM>` derived from the class template `Grid<DIM>`. The grid partitions S_k, the identifiers of the neighboring partitions and its directions are stored as member variables (see Listing 4). The direction of the

```
// All grid partitions: S_k for k=0,1,...,d-1
std::vector< GridSub<DIM> > mPartition;
// Identifiers of all neighbours of all partitions.
std::vector< std::vector<int> > mvNeighbourId;
// Directions of all neighbours of all partitions.
std::vector< std::vector<int> > mvNeighbourDir;
```

Listing 4: Member variables of the class template `ScaFES::GridGlobal<DIM>`.

neighbors of a given grid partition S_k is described for all $p \in \{0, 1, 2, \ldots, d-1\}$ by the following variable:

$$n(k, 2 \cdot p) := \text{left neighbor of } k \text{ in direction } p,$$
$$n(k, 2 \cdot p + 1) := \text{right neighbor of } k \text{ in direction } p.$$

C. Representation of Physical Fields

Let $v : \Omega \to \mathbb{R}^m$ a given vector-valued physical field and $v_h : \Omega \to \mathbb{R}^m$ the corresponding approximation of v. The approximation v_h to v on the domain Ω can be alternatively described by the matrix $V \in \mathbb{R}^{N,m}$ which contains the function values of the discrete function v_h at all grid nodes $x_j \in \Omega_h$:

$$V_{j,q} := \left[v_h(x_j)\right]_q \qquad \forall j \in \{0, 1, 2, \ldots, N-1\}, \quad \forall q \in \{0, 1, 2, \ldots, m-1\}.$$

The global matrix V is partitioned into sub-matrices $V^{(k)}$ accordingly to the domain decomposition approach, i.e. given the grid partition S_k, the matrix $V^{(k)}$ works on the nodes of this grid partition and is mapped to the corresponding MPI process. An index of the global matrix V is mapped onto the grid partition S_k via

$$h_k := g_{\Omega_h} \circ g_{S_k}^{-1}.$$

Thus, the following equalities hold for all elements $j \in \{0, 1, 2, \ldots, N_k\}$, for all components $l \in \{0, 1, 2, \ldots, m-1\}$, and for all grid partitions $k \in \{0, 1, 2, \ldots, q-1\}$ (see also Fig. 5):

$$V_{j,l}^{(k)} = V_{h_k(j),l} \ , \qquad V_{j,l} = V_{h_k^{1-}(j),l}^{(k)} \ .$$

The sub-matrix $V^{(k)}$ is represented by the class template

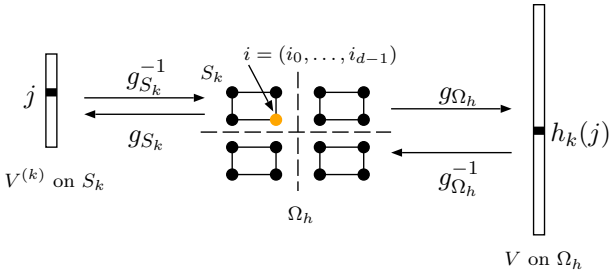


Fig. 5: Mappings of element j of the (local) vector $V^{(k)}$ related to grid partition S_k to element $h_k(j)$ of the (global) vector V related to grid global Ω_h and vice versa.

`DataField<CT,DIM>`. The template parameter `CT` can be replaced by the ScaFES type `ScaFES::Ntuple<CT,MM>` in order to handle the above vector-valued physical fields or by basic data types like `double` in order to handle

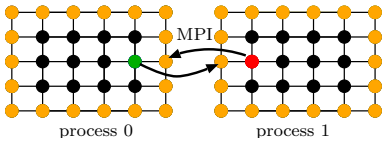


Fig. 6: Synchronization of a function value at the boundary of both grid partitions The corresponding grid nodes at the boundary of each partition are colored red resp. green, ghost grid nodes are colored orange.

real-valued physical fields, too. The function values at all grid nodes are stored continuously in memory. These values can be easily set and accessed just by passing the positions of the grid nodes (see Listing 5). There are two variants for each access method: According to the class template `std::vector<CT>`, the method `operator()` returns the addressed component, directly, whereas the method `at()` performs an additional range check. For enabling access to

```
ScaFES::DataField<double,3> V;
ScaFES::Ntuple<int,3> idxNode(0); V(idxNode) = 0;
int i(0), j(0), k(0);              V(i,j,k) = 0;
```

Listing 5: Possibilities to set the elements of a three-dimensional vector to zero at the first node of a grid.

function values on a different grid partition, the function values at additional grid nodes ("halos") are stored (see Fig. 6). This speeds up the computations as one does not have to fetch the necessary values again and again. Send and receive buffers are provided for the halos in order to exchange the function values between the grid partitions. The communication is implemented asynchronously. As the structures of the send and receive buffers do not change over time, the Boost.MPI skeleton concept [10] is used. Within this concept, the contents are separated from the structures and thus, the contents have to be exchanged only once, resulting in an improved MPI communication [11]. The class template consists of several grids and sub-grids like the grid partition S_k represented by `mIdxSetNormal` or the sub-grids of all ghost grid nodes represented by `mIdxSetGhost` (see Fig.**??**). This has the advantage that the function values at these grid nodes can be easily accessed using the corresponding grid iterators. In particular, the elements of the sub-matrix $V^{(k)}$ can be traversed accordingly to the demands of the communication using the iterators of the member variables `mIdxSetComm` and `mIdxSetGhost`. The data exchange is done if and only if there are at least two grid partitions and the stencil width of the data field is not equal to zero.

D. Representation of Initial Boundary Value Problems

As we consider time-dependent problems, we have to discretize the time interval $[t_S, t_E]$. Let $n_\tau \in \mathbb{N}$ the number of time steps. Then, for a given time step $l \in \{0, 1, 2, \ldots, n_\tau - 1\}$, we get the time $t_l = t_S + l \cdot \tau$ for a time step size $\tau := (t_E - t_S)/(n_\tau - 1)$. We denote the set of all times t_l by τ_h. Let $u_h : \Omega \times [t_S; t_E] \to \mathbb{R}$ be an approximation of u.

```
// Program parameters.
ScaFES::Parameters mParams;
// Global grid.
ScaFES::GridGlobal<DIM> mGG;
// Grid of all nodes.
ScaFES::Grid<DIM> mIdxSetAll;
// sub grid of all normal nodes.
ScaFES::GridSub<DIM> mIdxSetNormal;
// Sub grid of all boundary nodes.
std::vector< ScaFES::GridSub<DIM> > mIdxSetBorder;
// Grid of all ghost nodes.
std::vector< ScaFES::GridSub<DIM> > mIdxSetGhost;
// Grid of all nodes to be communicated.
std::vector< ScaFES::GridSub<DIM> > mIdxSetComm;
// Number of ghost layers.
int mNghostLayers;
// Pointer to the memory of the vector.
CT* mElemData;
// Buffers for sending and receiving.
std::vector<ScaFES::Buffer<CT>> mValuesToExchange;
// Output file for writing vectors.
ScaFES::DataFile<CT, DIM> mOutput;
```

Listing 6: Member variables of the class template `ScaFES::DataField<CT,DIM>`.

Then, defining the matrices

$$U_{j,q}^{(l)} := [u_h(x_j, t_l)]_q, \qquad F_{j,q}^{(l)} := [f(x_j, t_l)]_q,$$
$$G_{j,q}^{(l)} := [g(x_j, t_l)]_q, \qquad \widetilde{U}_{j,q} := [\widetilde{u}(x_j)]_q$$

for all $x_j \in \Omega_h$ and for all $t_l \in \tau_h$, and using a numerical method like finite differences in space leads to the following system of equations for all time steps l:

$$\begin{aligned} U_j^{(l+1)} &= \left(A(U^{(l)})\right)_j - F_j^{(l)} && \forall j \in \mathcal{G}_I(\Omega_h), \\ U_j^{(l+1)} &= G_j^{(l)} && \forall j \in \mathcal{G}_B(\Omega_h), \\ U_j^{(0)} &= \widetilde{U}_j && \forall j \in \mathcal{G}(\Omega_h) \end{aligned}$$

The matrix $A \in \mathbb{R}^{N,N}$ results from the discretization in space. A depends on the current iterate $U^{(l)}$. If the differential operator F in the initial boundary value problem is a linear one, then A depends linear on $U^{(l)}$, too. The discretization of the underlying problems using a numerical method like the finite difference method very often results in a sparse system matrix A, i.e. only the values at direct neighboring nodes are needed for the computation of the new iterate at an interior grid node. In fact, the matrix A will never be set up, but the matrix vector product $AU^{(l)}$ will be computed in each time step. All equations are independent from each other. Thus, the computation of the new iterate $U^{(l+1)}$ can be done completely in parallel. In particular, one can reorder the system of equations such that the communication and the computations can be done concurrently on each grid partition S_k:

- Compute $U_j^{(k;0)} = \widetilde{U}_j^{(k)}$ at all grid nodes $j \in \mathcal{G}$.
- Perform for all time steps $l \in \{0, 1, 2, \ldots, n_\tau - 1\}$:
  - Compute iterate $U_j^{(k;l+1)}$ at all (partitions related) boundary nodes $j \in \mathcal{G}_B(S_k)$,
  - Copy values $U_j^{(k;l+1)}$ at all boundary nodes $\mathcal{G}_B(S_k)$ to the send buffers.
  - Exchange values of the send buffers with all directly neighboring grid partitions using non-blocking sends and receives
  - Compute iterate $U_j^{(k;l+1)}$ at all (partitions related) interior nodes $j \in \mathcal{G}_I(S_k)$,
  - Wait until all communication calls have been finished.
  - Copy values from the receive buffers to halos of current iterate $U^{(k;l+1)}$.
  - Swap old iterate $U^{(k;l)}$ and new iterate $U^{(k+1;l)}$.

Initial boundary value problems are represented by the class template `Problem<PRBLM,CT,DIM>`. The class template makes use of the so called "curiously recurring template pattern" [12]. As a consequence, the user has to implement an own class inherited by this class template. This derived class has to contain the methods given in Listing 8. The template parameter `CT` represents the data type of all involved data fields and `DIM` represents the space dimension d. The old and new iterate at each time step are stored in the two member variables `mVectOld` and `mVectNew`. The type of all nodes (interior, boundary) of the grid partition is stored in a member variable named `mNodeType`. (see Listing 7). Matrices like $G^{(l)}$ can be added to the problem using the method `addDataField()`. This method requires the name of the physical field, its stencil width, and a flag if the field is an unknown one or not. The stencil width directly corresponds to the number of ghost layers at the boundary of a grid partition (see Fig. 6). Amongst other,

```
// Program parameters.
ScaFES::Parameters mParams;
// Global grid.
ScaFES::GridGlobal<DIM> mGG;
// Type of grid nodes.
ScaFES::DataField<short int, DIM> mNodeType;
// Old iterate U^{(k,l)} at grid partition S_k.
std::vector< ScaFES::DataField<CT, DIM> > mVectOld;
// New iterate U^{(k+1,l)} at grid partition S_k.
std::vector< ScaFES::DataField<CT, DIM> > mVectNew;
```

Listing 7: Member variables of the class template `ScaFES:Problem<PRBLM,CT,DIM>`.

there are access methods named `gridsize()` and `tau()` for the grid sizes h_p and the time step size τ. Known data fields can be accessed using the method `knownDf()`. The above algorithm over all time steps is implemented in the method `iterate()`, and the mapping c is implemented in the method `connect()`. In order to control program runs, the most important parameters can be read in from the command line. This has the advantage that one does not have to compile a program again if the program should be executed with a different parameter set. Furthermore, shell scripts can be easily created for parametrized test runs (like weak or strong scalability tests). The class `Parameter` represents a set of command line parameters.

E. Summarizing Used Parallelization Techniques

In the end of this section, we summarize the three used parallelization techniques.
On top, we adopt a domain decomposition approach by dividing the global grid into several grid partitions which are

```
template<typename T>
void updateInner(
   std::vector<ScaFES::DataField<T,DIM>>& v1,
   std::vector<ScaFES::DataField<T,DIM>> const& v0,
   ScaFES::Ntuple<int,DIM> const& idxNode,
   int const& timestep
);
template<typename T>
void updateBorder(
   std::vector<ScaFES::DataField<T,DIM>>& v1,
   std::vector<ScaFES::DataField<T,DIM>> const& v0,
   ScaFES::Ntuple<int,DIM> const& idxNode,
   int const& timestep
);
```

Listing 8: Methods which must be implemented by user in the derived problem class.

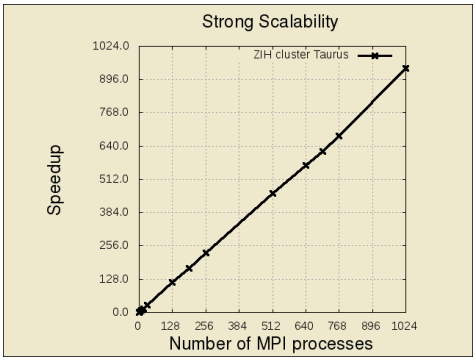


Fig. 7: Strong scaling w.r.t. MPI of the considered three-dimensional heat equation problem on ZIH cluster Taurus.

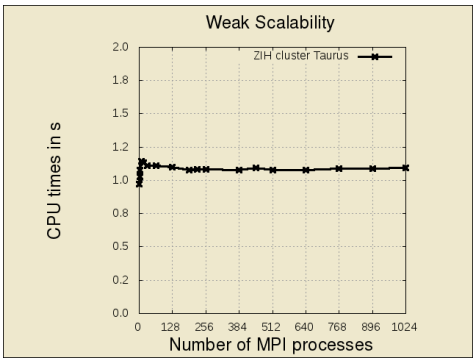


Fig. 8: Weak scaling w.r.t. MPI of the considered three-dimensional heat equation problem on ZIH cluster Taurus.

mapped via a one-to-one relation onto a given number of MPI processes such that each MPI process computes a portion of the global problem (cp. subsection II-B). This technique addresses distributed as well as shared memory systems. On the node level, we use OpenMP work sharing constructs in order to parallelize the traversing and computation of values of physical fields on one grid partition. On the core level, we vectorize small loops using the compiler SIMD vectorization. Nowadays, modern compilers like the GCC or ICC can SIMD-vectorize many loops automatically if these loops are written in an appropriate way. Thus, we decided to support this automatical compiler vectorization and prepared the loops in the framework. i.e. we used for- instead of do-while-loops, removed dependencies between involved elements within the loops e.g.
The user can choose between a pure MPI, a pure OpenMP and a mixed mode parallelization to adopt and benefit from the underlying hardware architecture like a cluster system with shared memory nodes and distributed memory across nodes or a full shared memory system.

III. Solving A d-dimensional Heat Equation Problem

In order to show how an initial boundary value problem can be solved using ScaFES, we consider the d-dimensional heat equation on the d-dimensional unit hypercube for arbitrary $d \in \mathbb{N}$. Given the time interval $[0;1]$, the domain $\Omega := (0,1)^d$, the source $f : \bar{\Omega} \times (0;1] \to \mathbb{R}$, $f(x,t) := 0$, the boundary condition $g : \partial\Omega \times (0;1] \to \mathbb{R}$, $g(x,t) := 0$, and the initial condition $\widetilde{u} : \Omega \to \mathbb{R}$, $\widetilde{u}(x) := \prod_{i=0}^{d-1} x_i \cdot (x_i - 0.5)^2 \cdot (1 - x_i)$, then the task is to find $u : \bar{\Omega} \times [0;1] \to \mathbb{R}$ such that the following system of equations is fulfilled:

$$\begin{aligned} \partial_t u - \Delta u &= f && \text{in } \Omega \times (0;1], \\ u &= g && \text{on } \partial\Omega \times (0;1], \\ u(\cdot, t_S) &= \widetilde{u} && \text{in } \Omega. \end{aligned}$$

We discretized this system of equations in space using the finite difference method with the standard centered stencil (7-point stencil in 3D, e.g.) and in time using the explicit Euler scheme. Therefore, we defined

$$\begin{aligned} U_j^{(l)} &:= u(x_j, t_l), & F_j^{(l)} &:= f(x_j, t_l), \\ G_j^{(l)} &:= g(x_j, t_l), & \widetilde{U}_j &:= \widetilde{u}(x_j). \end{aligned}$$

for all $x_j \in \Omega_h$, and for all $t_l \in \tau_h$. The resulting system of equations reads for all time steps $l \in \{0, 1, \ldots, n_\tau\}$:

$$\begin{aligned} U_j^{(l+1)} &= \tau \cdot \sum_{p=0}^{d-1} \big(-2 \cdot U_j^{(l)} + U_{c(j;2\cdot p))}^{(l)} + U_{c(j;2\cdot p+1))}^{(l)} \big) / h_p^2 \\ &\quad - \tau \cdot F_j^{(l)} + U_j^{(l)} \quad \forall j \in \mathcal{G}_I(\Omega_h), \\ U_j^{(l+1)} &= G_j^{(l)} \quad \forall j \in \mathcal{G}_B(\Omega_h), \\ U_j^{(0)} &= \widetilde{U}_j \quad \forall j \in \mathcal{G}(\Omega_h). \end{aligned}$$

The implementation of this problem is given in Listing 9. We emphasize that this implementation is not bound to a certain storage data type and a certain space dimension due to the employment of the class template parameters `CT` and `DIM`.

IV. Performance Results

In the last section, we demonstrated that ScaFES is very user-friendly. Users do not need any knowledge about parallelization techniques but can concentrate on the implementation of the numerical algorithms. But what is the price to pay for this rapid and simple prototyping? Does a ScaFES application really scale and can it compete against applications which are using one of the more general and heavyweight software packages mentioned in section I? In order to figure it out, we performed different scaling tests w.r.t. MPI and OpenMP and compared ScaFES to PETSc, representing a state-of-the-art software package.
To show the scalability of ScaFES, we used the implementation

```
#include "ScaFES.hpp"
% template<typename CT, std::size_t DIM> // Source f.
inline void funcF(CT& fx, ScaFES::Ntuple<CT,DIM> const& x, CT const& t) {
  fx = 0.0;
}
template<typename CT, std::size_t DIM> // Boundary condition g.
inline void funcG(CT& fx, ScaFES::Ntuple<CT,DIM> const& x, CT const& t) {
  fx = 0.0;
}
template<typename CT, std::size_t DIM> // Initial condition \tilde{u}.
inline void funcUt(CT& fx, ScaFES::Ntuple<CT,DIM> const& x, CT const& t) {
  fx = 1.0;
  for (std::size_t pp = 0; pp < DIM; ++pp) {
    fx *= (x[pp] * (x[pp] - 0.5) * (x[pp] - 0.5) * (1.0 - x[pp]));
}
template<typename CT, std::size_t DIM> // Own problem class.
class HeatEqnFDM : public ScaFES::Problem<HeatEqnFDM<CT,DIM>, CT, DIM> {
  public:
    HeatEqnFDM(ScaFES::Parameters const& cl,
               ScaFES::GridGlobal<DIM> const& gg)
      : ScaFES::Problem<HeatEqnFDM<CT,DIM>, CT, DIM>(cl, gg) {
        this->addDataField("F", 0, funcF<CT,DIM>, true);
        this->addDataField("G", 0, funcG<CT,DIM>, true);
        this->addDataField("U", 1, funcUt<CT,DIM>, false);
      }
    template<typename TT> // Method must be implemented!
    void updateInner(std::vector<ScaFES::DataField<TT,DIM>>& vNew,
                     std::vector<ScaFES::DataField<TT,DIM>> const& vOld,
                     ScaFES::Ntuple<int,DIM> const& idxNode,
                     int const& timestep) {
      vNew[0](idxNode) = vOld[0](idxNode)
                         - this->tau() * this->knownDf(0, idxNode);
      for (std::size_t pp = 0; pp < DIM; ++pp) {
        vNew[0](idxNode) += this->tau() * (
                 -2.0 * vOld[0](idxNode)
                     + vOld[0](this->connect(idxNode, 2*pp))
                     + vOld[0](this->connect(idxNode, 2*pp+1)) )
                     / (this->gridsize(pp) * this->gridsize(pp));
      }
    }
    template<typename TT> // Method must be implemented!
    void updateBorder(std::vector<ScaFES::DataField<TT,DIM>>& vNew,
                      std::vector<ScaFES::DataField<TT,DIM>>const& vOld,
                      ScaFES::Ntuple<int,DIM> const& idxNode,
                      int const& timestep) {
      vNew[0](idxNode) = this->knownDf(1, idxNode);
    }
};
int main(int argc, char *argv[]) {     // Main program.
  ScaFES::Parameters pp(argc, argv);   // Read in command line options.
  ScaFES::GridGlobal<3> gg(pp);        // Create grid partitions.
  HeatEqnFDM<double,3> prblm(pp, gg);  // Create 3D heat eqn. problem.
  prblm.iterate();                     // Iterate over all time steps.
  return 0;
}
```

Listing 9: Source code in ScaFES for solving the three-dimensional heat equation on the three-dimensional unitcube.

of the three-dimensional heat equation problem as shown in section III as test case with purely MPI parallelization. The strong and weak scaling tests w.r.t. MPI were performed on island 1 of the HPC system Taurus at ZIH which is based on Intel® Sandy Bridge multi-core chips with 16 cores per shared-memory node and a total of 4320 cores [13]. The application was compiled with GCC 4.8.0 and highest optimization level. All measurements refer to computing 20 time steps, the computational domains were partitioned in the third dimension, the times for initialization and output of simulation results are not considered. We discretized the computational domain using $128 \times 128 \times 8192$ nodes as fixed workload for the strong scaling test and used a fixed grid of $128 \times 128 \times 8$ nodes per process for the weak scaling test. The results in Fig. 7 and in Fig. 8 show that the application indeed scales weakly and strongly.

The strong scaling tests w.r.t. OpenMP were performed on the HPC cluster Atlas at ZIH as this cluster possesses 64 cores per shared-memory node [14]. We used the implementation of the corresponding two-dimensional heat equation problem as test case. The application was compiled with GCC 4.7.1 and highest optimization level. Again, all measurements refer

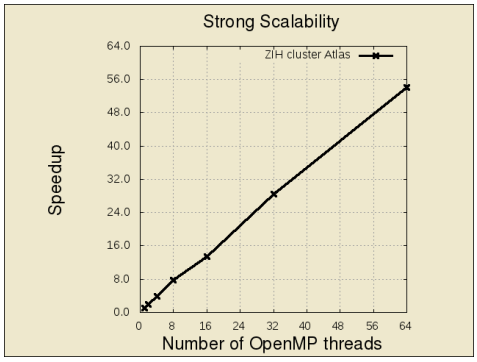


Fig. 9: Strong scaling w.r.t. OpenMP of a two-dimensional heat equation problem on ZIH cluster Atlas.

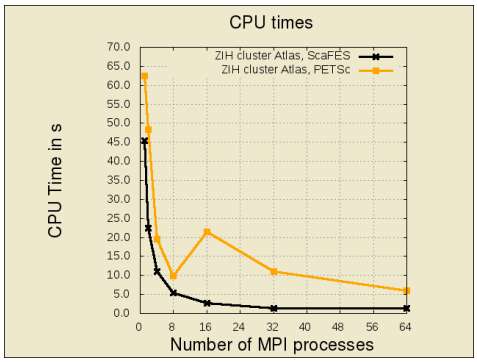


Fig. 10: Comparison of ScaFES with PETSc for a two-dimensional heat equation problem on ZIH cluster Atlas.

to computing 20 time steps, the computational domains were partitioned in the second dimension. The times for initialization and output of simulation results are not considered. We discretized the computational domain using 4096×4096 nodes as fixed workload. The results in Fig. 9 show that the application scales strongly w.r.t. OpenMP, too.

We have seen that ScaFES indeed is a high-scaling framework. But will it be as fast as one of the other existing frameworks? Therefore, we implemented the above heat equation problem in two dimensions in PETSc 3.4.3, too. The performance test was again run on the ZIH cluster Atlas. Fig. 10 shows that implementation in ScaFES is not only comparable to PETSc, but outperfoms it for the considered example, for 64 processes by a factor of approximately 7.

All source codes, scripts and results of the performance tests are provided at `tu-dresden.de/zih/scafes` such that the presented results can be reproduced by interested people.

V. Conclusions and Outlook

We described the principal design aspects of the HPC framework for explicit solvers on structured grids named ScaFES and showed its good scalability. By presenting an implementation example, we illustrated the user-friendly interfaces. The development resp. design of ScaFES was driven by the ambitions to create a high quality and scalable software tool, which is easy to use and is portable to various platforms and architectures. The underlying parallelization is encapsulated and hidden from the user. The user has to implement serial code, only. The parallelization and communication is managed by ScaFES. Because of its user-friendliness, ScaFES can be used as a rapid prototyping tool to evaluate and compare numerical methods as well as to write high quality production code without loosing scalability and efficiency.

Load balancing is normally a key aspect of adaptive mesh methods. If it should be necessary, one could achieve load balancing by introducing a cost function to the domain decomposition algorithm.

Acknowledgments

ScaFES is funded by the Federal Ministry of Education and Research (BMBF) within the project HPC-FliS under the support code 01 IH 11 009.

References

[1] S. Vey and A. Voigt, "AMDiS: adaptive multidimensional simulations," *Computing and Visualization in Science*, vol. 10, no. 1, pp. 57–67, 2007.

[2] S. Balay, W. D. Gropp, L. C. McInnes, and B. F. Smith, "Efficient Management of Parallelism in Object Oriented Numerical Software Libraries," in *Modern Software Tools in Scientific Computing*, E. Arge, A. M. Bruaset, and H. P. Langtangen, Eds. Birkhäuser Press, 1997, pp. 163–202.

[3] P. Bastian, M. Blatt, A. Dedner, C. Engwer, R. Klöfkorn, M. Ohlberger, and O. Sander, "A generic grid interface for parallel and adaptive scientific computing. Part I: abstract framework." *Computing*, vol. 82, no. 2-3, pp. 103–119, 2008.

[4] K. Yee, "Numerical solution of inital boundary value problems involving maxwell's equations in isotropic media," *IEEE Transactions on Antennas and Propagation*, vol. 14, pp. 302–307, May 1966.

[5] ISO, *ISO/IEC 14882:2011 Information technology – Programming languages – C++*. Geneva, Switzerland: International Organization for Standardization, Feb. 2012, last checked on 2013-11-14 (07:30 CET). [Online]. Available: http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=50372

[6] B. Dawes, D. Abrahams, and R. Rivera, "Boost C++ Libraries Homepage," last checked on 2013-11-02 (07:12 CET). [Online]. Available: http://www.boost.org

[7] G. V. Vaughan, B. Elliston, T. Tromey, and I. L. Taylor, "The Goat Book," 2000, last checked on 2013-10-18 (16:12 CET). [Online]. Available: http://sources.redhat.com/autobook/

[8] G. Hager and G. Wellein, *Introduction to High Performance Computing for Scientists and Engineers*, 1st ed. Boca Raton, FL, USA: CRC Press, Inc., 2010.

[9] M. J. Berger and S. H. Bokhari, "A Partitioning Strategy for Nonuniform Problems on Multiprocessors." *IEEE Trans. Computers*, vol. 36, no. 5, pp. 570–580, 1987, last checked on 2013-11-08 (12:30 CET). [Online]. Available: http://dblp.uni-trier.de/db/journals/tc/tc36.html#BergerB87

[10] M. Gauckler and D. Egloff, "The Meat and Bones of Message Passing," Sep. 2006, last checked 2013-10-28 (15:44 CET). [Online]. Available: http://daveabrahams.com/files/2010/09/meat_and_bones_of_mpi.pdf

[11] M. Flehmig, "Framework zur effizienten parallelen Berechnung expliziter orts- und zeitdiskreter Verfahren," Diploma Thesis, Center For Information Services And High Performance Computing At TU Dresden, 12 2011.

[12] J. O. Coplien, "Curiously Recurring Template Patterns," *C++ Report*, 1995.

[13] Center For Information Services And High Performance Computing At TU Dresden, "HPC Web-Compendium: Cluster Taurus," last checked 2013-11-14 (13:22 CET). [Online]. Available: https://doc.zih.tu-dresden.de/hpc-wiki/bin/view/Compendium/HardwareTaurus

[14] ——, "HPC Web-Compendium: Cluster Atlas," last checked 2013-10-28 (15:30 CET). [Online]. Available: https://doc.zih.tu-dresden.de/hpc-wiki/bin/view/Compendium/HardwareAtlas

[15] R. Byrd, P. Lu, J. Nocedal, and C. Zhu, "A Limited Memory Algorithm for Bound Constrained Optimization," *SIAM Journal on Scientific Computing*, vol. 16, no. 5, pp. 1190–1208, 1995.

A Perfomance Study of Parallel Cauchy Reed/Solomon Coding

Peter Sobe and Peter Schumann
Faculty of Informatics/Mathematics
University of Applied Sciences Dresden, Germany
Contact: sobe@htw-dresden.de

***Abstract*—Cauchy-Reed/Solomon coding is applied to tolerate failures of memories and data storage devices in computer systems. In order to obtain a high data access bandwidth, the calculations for coding must be fast and it is required to utilize parallelism. For a software-based system, the most promising approach is data parallelism which can be easily implemented with OpenMP on a multicore or multiprocessor computer. A beneficial aspect is the clear mathematical nature of coding operations that supports functional parallelism as well. We report on a storage system application that generates the encoder and decoder as C-code automatically from a parametric description of the system and inserts OpenMP directives in the code automatically.**
We compare the performance in terms of achieved data throughput for data parallelism and for functional parallelism that is generated using OpenMP.

I. Introduction

Nowadays, a computer typically owns a number of processor cores that can be utilizes to accelerate computation-intense applications by exploiting parallelism. Such an application is data en- and decoding which is a necessary operation for fault-tolerant memory and storage systems. Every block of data that is stored is involved in calculations that produce a high computational load. This computations normally slow down data access, but this can be mitigated by processing the data for coding in parallel. Such coding algorithms allow data-parallel calculations on independent blocks of data. A closer look on the en- and decoding algorithms reveals that calculations can be separated from another and functional parallelism can be applied as well. The question is whether data parallelism, function parallelism or a combination of both is the best choice.

The contribution of the paper is a performance evaluation of a failure-tolerant coding application that exploits the potentials of data parallelism and of functional parallelism. These different approaches to parallelism are implemented using OpenMP [1], a multiprocessing library with compiler support. Due to the clear mathematical concept of coding, the algorithms can be generated automatically with specification of code parameters. In addition, the OpenMP parallelism extensions are included automatically.

The rest of the paper is organized as follows. The technical background and related work are described in Section II, particularly the Cauchy-Reed/Solomon code that is selected for the coding algorithm. The method of automatic code generation including to organize the workload in different functions and to insert OpenMP directives is described in Section III. In Section IV we report on the achieved acceleration by parallelism and draw conclusions for further optimizations.

II. Background and Related Work

In the first part of this section the coding algorithm is described, together with the structure of data that is distributed accross the system. A second part is dedicated to the OpenMP approach to parallel programming and runtime support. In a third part, a selection of related work is addressed.

A. The application: Cauchy Reed/Solomon coding

This work is based on the Reed/Solomon code [2], particularly on the XOR-based variant introduced in [3] that is denoted by Cauchy-Reed/Solomon code (CRS). CRS is a so called erasure-tolerating code that allows to recalculate parts of the original data that got lost (or got erased) as result of hardware failures or unreachability of data in networks. In practice, CRS can be applied as well for correcting failures. For this, it is combined with error detecting codes that validate blocks as correct or corrupted, and CRS replaces the corrupted blocks by recalculated content.

The CRS coding works as follows: For encoding, data must be split in k parts that have to be assigned independent storage devices (or memory modules). A number of m additional blocks are calculated from the k original blocks. The m blocks are redundancy and used solely for decoding to recalculate lost blocks. The redundant m blocks are placed on additional storage devices.

The CRS code shows several beneficial properties.

- It is a so called regular code that separates original data and redundant data. This property allows to read the data without execution of decoding calculations in failure-free situations. When data is written (or changed) the encoding calculation have to be executed always.
- The code is optimal w.r.t. the amount of redundant data and the number of failures that can be tolerated. With m redundant blocks, every situation with up to m lost blocks among the original and redundant ones can be tolerated.
- The code uses XOR operations that are available as machine instructions. In addition, XOR is included in the MMX and SSE instruction set extensions for parallel computations on wide registers (128 Byte) of x86 processors.

- A specific coding and decoding algorithm can be generated for every combination of k and m ($k > 0$,$m > 0$). CRS can be applied for every distribution factor (k) and every number of additional blocks (m), where the latter parameter scales number of failures that can be tolerated.

The coding algorithm handles every block as split in ω fragments. From $k \times \omega$ fragments taken from the original data, $m \times \omega$ redundant fragments are calculated. CRS constrains the value of ω by the relation $2^\omega > k + m$. The original and redundant data fragments are distributed block-wise across the devices (memory modules, storage devices, computers). The data layout is depicted in Fig. 1 for the example of a $k = 3, m = 2, \omega = 3$ code. In this case, every block is split in three fragments that are differentiated for the encoding and decoding calculations.

The calculation can be seen as a linear equation system $A \cdot o = r$, with o as a vector of original data fragments and r as the redundant fragments. The coding matrix A consists of elements $\in \{0, 1\}$ and is constructed under mathematical constraints in order to allow a decoding using the inverse matrix. The elements of A are factors and normally would require a multiplication with the data fragments. Due to the factors 0 and 1, solely a selection takes place whether a fragment is included in the equation or not. Every equation is reduced to a sum of selected fragments that is expressed by bitwise XOR-ing the fragments. The XOR operation is expressed by the symbol $\oplus$. The equations for encoding are shown in (1) with fragments that are numbered by $u0, u1, \ldots, u8$. The two redundant blocks contain the fragments $u9, u10 \ldots, u14$.

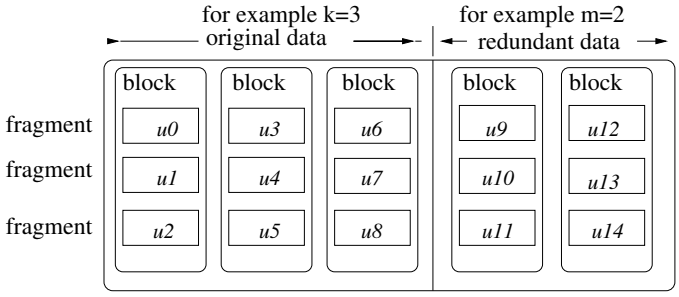


Fig. 1. Data layout for a system that distributes data across 3 devices and adds 2 redundant devices to store redundant data.

$$
\begin{aligned}
u9 &= u1 \oplus u2 \oplus u3 \oplus u4 \oplus u6 \ , \\
u10 &= u0 \oplus u1 \oplus u5 \oplus u7 \ , \\
u11 &= u0 \oplus u1 \oplus u2 \oplus u3 \oplus u8, \\
u12 &= u0 \oplus u1 \oplus u2 \oplus u5 \oplus u6 \oplus u8 \ , \\
u13 &= u0 \oplus u3 \oplus u5 \oplus u6 \oplus u7 \oplus u8 \ , \\
u14 &= u0 \oplus u1 \oplus u4 \oplus u7 \oplus u8
\end{aligned}
\tag{1}
$$

The equations can be optimized by elimination of common subexpressions. As a result, encoding can be expressed by equations that do not contain redundant calculations, but show data dependencies among them. The t-fragments are temporary and have to be calculated before being used in other equations. We denote this coding as an iterative style (see (2)), and the non-optimized variant as the direct style (see (1)) .

$$
\begin{aligned}
t15 &= u1 \oplus u2 \ , \quad t16 = u3 \oplus u6 \ , \\
t17 &= u0 \oplus u1 \ , \quad t18 = u5 \oplus u7 \ , \\
t19 &= u0 \oplus u8 \ , \quad t20 = t15 \oplus t19 \ , \\
& \\
u9 &= u4 \oplus t15 \oplus t16 \ , \quad u10 = t17 \oplus t18 \ , \\
u11 &= u3 \oplus t20 \ , \quad u12 = u5 \oplus u6 \oplus t20 \ , \\
u13 &= t16 \oplus t18 \oplus t19 \ , \quad u14 = u4 \oplus u7 \oplus u8 \oplus t17
\end{aligned}
\tag{2}
$$

The principle of generating XOR-equations for en- and decoding is described in detail in [4]. A tool, called *cauchyrs* [5] is applied to prepare XOR equations from the code parameters. It creates XOR-based equations according to a given number of data storage resources (k) and redundancy storage resources (m). Additional parameters for the equations are the block length, as well as, whether the tool should generate equations in a direct style or in an iterative style. The equations are generated independently from the actual data storage operation (Write, read, update) in the storage or memory system.

As a preliminary analysis, coding equations allow to assess the cost of coding by the number of XOR operations in relation to the number of original blocks that have to be written. The number of XOR operations directly follows from the equations that were generated by the cauchyrs tool. The more XOR operations, the more compute-intense the coding is. This normally would cause reduced data access rates with a negative influence on the write rate and with an influence on the read rate, in case of failures. The result show that the variation of the distribution factor k has only a little impact. However, increasing the failure tolerance by the number of redundant blocks m influences the costs noticeably as can be seen in Fig. 2. Seen roughly, encoding requires $m+1$ XOR operations per block for every single block that is written. For instance, when 1kByte is written and m is 2, 384 XOR operations have to be executed that combine two 8 Byte operands (1024 Byte/ 8 Byte $\times$ 3). In addition to the 384 instructions, 3072 Bytes must be moved through the processor for calculation. As a result of this high effort, only a few hundred MByte/s remain as data rate for sequential encoding, even when the storage system offers higher data access rates.

B. OpenMP for Multicore

OpenMP [1] is a multi processing library with compiler support that was first published in 1997 and is now available for all major compilers (C,C++, Fortran). OpenMP is supported for instance by the Gnu gcc compiler and MS Visual Studio. OpenMP leads to a multithreaded program execution and is suitable for shared memory multiprocessing platforms. The typical way to use OpenMP is to add a few compiler directives and library calls to the original sequential code. These so called pragma directives give hints to create thread pools in the way of fork-join-parallelism with a master thread and a number of worker threads. It allows to gradually add parallelism to a sequential program without changing the program marginally.

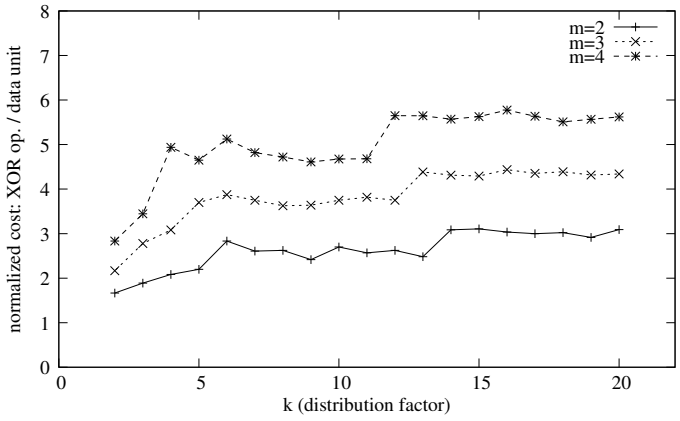


Fig. 2. Analysis of computational cost: number of XOR operations per data unit that is stored.

A common directive is *#pragma omp parallel* that marks a block to be executed by several threads. It creates a thread pool where every thread executes the program. The programmer distributes the workload among threads, particularly the work that is done within loops. A typical form is the combination of a parallel block and a for-loop (*#pragma omp parallel for*) that is illustrated in the C-program in listing 3. In a sequential program, the for-loops are the origin of data parallelism.

Another way to express parallelism are sections of different code blocks as illustrated in listing 4. These sections are used to express function parallelism, where threads execute different parts of the program code (i.e. functions).

```
#include <omp.h>
#define N_VALUES 10000

main()
{
  double a[N_VALUES], b[N_VALUES], d[N_VALUES];
  double c=3.14;
  int i, nt=8;

  omp_set_num_threads(nt); // set number of threads

#pragma omp parallel for
  for (i=0;i<N_VALUES;i++)
    d[i] = a[i]+c*b[i];
}
```

Fig. 3. *OpenMP* example of data parallelism

```
void calc (in , out ) {
# pragma omp parallel sections {
# pragma omp section {
    f1 ( in , out );
  }
# pragma omp section {
    f2 ( in , out );
  }
.....
}}
}
```

Fig. 4. *OpenMP* example of parallel sections for function parallelism

Besides sections, explicit tasks (*#pragma omp task*) are another way to assign code blocks and functions to threads. This can be used to implement function parallelism as well.

OpenMP allows to add clauses in order to control, whether variables are shared among threads or have to be thread-private ones. Serialization and synchronization of threads is supported by additional directives, such as *#pragma omp critical* or *#pragma omp single*.

C. Related Work

For long time, research is directed to fast data en- and decoding for failure-tolerating systems, especially for compute-intense codes that flexible tolerate multiple failures. Around ten years ago, there was the need for specific hardware accelerated solutions, mostly implemented by FPGAs, e.g. [6]. Besides RAID controllers with a function-specific hardware, it is common to implement coding by software and using the GPU or multicore capabilities. The concept of multicore Reed/Solomon coding is described in [7]. An introduction of the translation concept to OpenMP and OpenCL is published in [8] that applies the concept of equations to multicore and to GPU-architectures. A purely GPU-directed work can be found in [9], [10].

Another way to implement software-based erasure-tolerant codes is to use flexible libraries, such as the jerasure library [11], a C/C++ library for matrix-based erasure-tolerant coding. At the time of publication (2007) this library contained sequential code. Meanwhile, several sources report on usage of this library in the context of GPU acceleration and multithreading.

A way between implementing the coding algorithms by oneself and the application of ready-to-use libraries is to develop the own functionality on base of code skeletons. These skeletons are pre-defined, reusable code components that can be applied to several algorithms and encapsulate parallelism. The most common example is Map-Reduce, but also other patterns of parallelism can be included in skeletons, such as a farm (master worker) or pipelined execution. An example for a skeleton programming library is [12] that supports parallel programming models like OpenMP, OpenCL and CUDA.

Even though it would be possible, in this work we do neither utilize a library nor build on skeletons. The parallel coding is implemented by pure C code that is automatically generated.

III. Equations, Program Code and OpenMP

In this section, we explain how the coding algorithm is transformed from an symbolic description (using equations) to C program functions, including the control statements for OpenMP parallelism.

Initially, data encoding and decoding is expressed by equations that have to be applied to data. For encoding, the equations do not change over time. Thus, encoding equations can be translated to C functions and included in the system before operations start.

Decoding equations change with the specific failure situation. Thus, these equations are generated on on demand and JIT compilation techniques have to be applied to produce a specifically optimized decoder for a failure situation. For small

system configurations, it is possible to calculate all sets of decoding equations in advance and to provide C functions for all failure cases that can be tolerated.

We follow the approach of decoupling the equation generation and the coding operations that are related to data. Fig. 5 shows the system components that are involved into en- and decoding. The cauchyrs tool generates equations, as well as C source code. The C functions are later used in the storage system to provide write and read operations with application data.

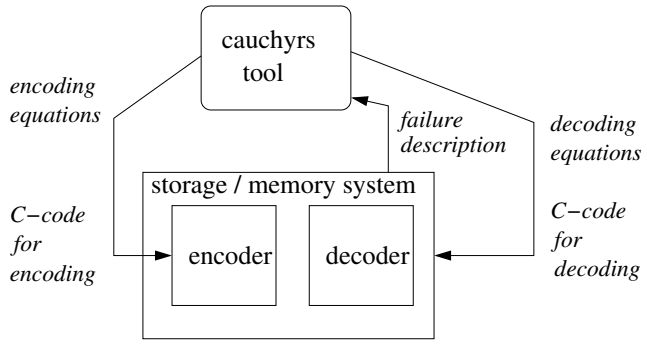


Fig. 5. Automatic C-code generation for the system that executes the coding algorithm.

From the internal representation of the coding equations, C functions can be generated that take an array of bytes as the first input parameter and an array of calculated redundancy bytes as the second parameter. Fig. 6 shows an encoding function, according to the previously given example with $k = 3$, $m = 2$ and $\omega = 3$ as parameters. The unit numbers can be recognized as the macro parameters of Of() and Rf(). The redundant units $u9, \ldots, u14$ are expressed by the array r, with macro parameters 0 to 5.

Typically, data that is written is longer than a single code word. Thus, fragments contain a number of Bytes (for instance 1024, or 2048), denoted by $blocklen$. A complete input data set covers $k \times \omega \times blocklen$ Bytes and is taken to produce $m \times k \times blocklen$ Bytes redundancy. The coding is repeated $blocklen$ times, which is expressed by a for-loop. This for-loop is the block that is distributed across several threads in the way of data parallelism. The inner block of the for-loop does not contain data dependencies to other iterations and therefore it can be easily handled by OpenMP. The directive *omp parallel for* precedes the for-statement. This instructs the compiler to introduce functionality that distributes the work across the threads in the thread pool. Special care must be taken for the local variables that store the values of the common subexpressions. These variables must be declared as thread-local and have to be assigned to every thread as private variables (see the *private* clause in Fig. 6)

The generated code for functional parallelism can be seen in Fig. 7 for a $k = 3$, $m = 2$, $\omega = 3$ system. The result of the computation is the same as of the data-parallel variant (see Fig. 6).

Depending on the number of cores (which is a parameter of the automatic code generation), a number of functions are created to cover different equations. The calculation of

```
#define UNIT_LEN 1024
#define Rf(a) a*UNIT_LEN+i
#define Of(a) a*UNIT_LEN+i

inline void encode(const char*, char*);

void encode(const char *n, char *r)
{
  int i;
  char t15,t16,t17,t18,t19,t20;

#pragma omp parallel for \
private (t15,t16,t17,t18,t19,t20)
  for (i = 0; i < UNIT_LEN; i++)
  {
    t15 = n[Of(1)]^ n[Of(2)];
    t16 = n[Of(3)]^ n[Of(6)];
    t17 = n[Of(0)]^ n[Of(1)];
    t18 = n[Of(5)]^ n[Of(7)];
    t19 = n[Of(0)]^ n[Of(8)];
    t20 = t15^ t19;

    r[Rf(0)] = n[Of(4)]^ t15^ t16;
    r[Rf(1)] = t17^ t18;
    r[Rf(2)] = n[Of(3)]^ t20;
    r[Rf(3)] = n[Of(5)]^ n[Of(6)]^ t20;
    r[Rf(4)] = t16^ t18^ t19;
    r[Rf(5)] = n[Of(4)]^ n[Of(7)]^ n[Of(8)]^ t17;
  }
}
```

Fig. 6. Automatically generated C-code that supports data parallelism

the equations within a function follows a direct coding style without data parallelism.

Equations are assigned to the individual coding functions (calc1, calc2, ...) as follows: First, equations are sorted according to the number of XOR operations. A number of calculation functions is created that are initially empty. Starting with the equation with the most XOR operations, the equations are assigned to the calculation functions, translated to C statements and placed in the function bodies. As long as there are empty coding functions, this is a round-robin assignment. When all calc-functions contain at least one equation, the next equation is assigned to the coding function with the least number of XOR operations.

In the example given in Fig. 7 it is not possible to generate an optimal balanced distribution. The 6 equations are distributed across 4 functions in the best way as possible, but with a slight imbalance. This effect disappears with larger system configurations that require more equations (due to higher values of m and/or ω).

IV. Performance Evaluation

The performance of the encoding operation was studied on two different systems, a 4-core AMD-Phenom II-X4, 3.2 GHz system and a 2× Opteron 6128, 2GHz (2×4 cores) system. The second system is slightly slower clocked, but offers the double number of processor cores.
We measured the data throughput in terms of the amount of original data (MByte) that was encoded during a fixed quantity of time (seconds). Fig. 8 shows data throughput under a varied number of threads on a 4-core system. All measurements show a clear advantage of the iterative encoder over the direct

```
void calc0 (const char *n, char *r)
{
  for (int i = 0; i < UNIT_LEN; i++)
  {
     r[Rf(3)] = n[Of(0)]^ n[Of(1)]^ n[Of(2)]^
                n[Of(5)]^ n[Of(6)]^ n[Of(8)];
  }
}
void calc1 (const char *n, char *r)
{
  for (int i = 0; i < UNIT_LEN; i++)
  {
     r[Rf(4)] = n[Of(0)]^ n[Of(3)]^ n[Of(5)]^
                n[Of(6)]^ n[Of(7)]^ n[Of(8)];
  }
}
void calc2 (const char *n, char *r)
{
  for (int i = 0; i < UNIT_LEN; i++)
  {
     r[Rf(0)] = n[Of(1)]^ n[Of(2)]^ n[Of(3)]^
                n[Of(4)]^ n[Of(6)];
     r[Rf(5)] = n[Of(0)]^ n[Of(1)]^ n[Of(4)]^
                n[Of(7)]^ n[Of(8)];
  }
}
void calc3 (const char *n, char *r)
{
  for (int i = 0; i < UNIT_LEN; i++)
  {
     r[Rf(2)] = n[Of(0)]^ n[Of(1)]^ n[Of(2)]^
                n[Of(3)]^ n[Of(8)];
     r[Rf(1)] = n[Of(0)]^ n[Of(1)]^ n[Of(5)]^
                n[Of(7)];
  }
}

void calc(const char *n, char *r)
{
#pragma omp parallel sections
  {
     #pragma omp section
     {
       calc0(n,r);
     }
     #pragma omp section
     {
       calc1(n,r);
     }
     #pragma omp section
     {
       calc2(n,r);
     }
     #pragma omp section
     {
       calc3(n,r);
     }
   }
}
```

Fig. 7. Automatically generated C-code with function parallelism

encoder. The variants of data parallelism scale with the number of threads used for encoding until the number of processor cores is reached. Unfortunately, the function-parallel encoder did not produce a higher throughput and is ranked on the level of direct data-parallel encoding. This can be explained by the same number of XOR operations as the direct encoder.

Surprisingly, measurements on smaller system configurations showed an up to three times better throughput of the function-parallel encoder in comparison to the best data-parallel variant. This effect motivated a deeper analysis of the generated assembler code through an inspection using objdump. We found that the gcc compiler generated SSE instructions for XOR operations on consecutive Bytes for the function-parallel encoder version, but only for small configurations (e.g. $k = 5$, $m = 2$) The compiler creates single Byte accesses for longer blocks and for higher values of k and m in the function-parallel version. The reason for this disadvantageous choice are runtime checks that have to be added for SSE acceleration that check that data in the arrays n and r do not overlap. We assume a compiler strategy that stops SSE vectorization when these runtime checks become too costly. A small hint to the compiler by declaring the pointers n and r as *__restrict__* allowed to use SSE vectorization for longer blocks and bigger system configurations.

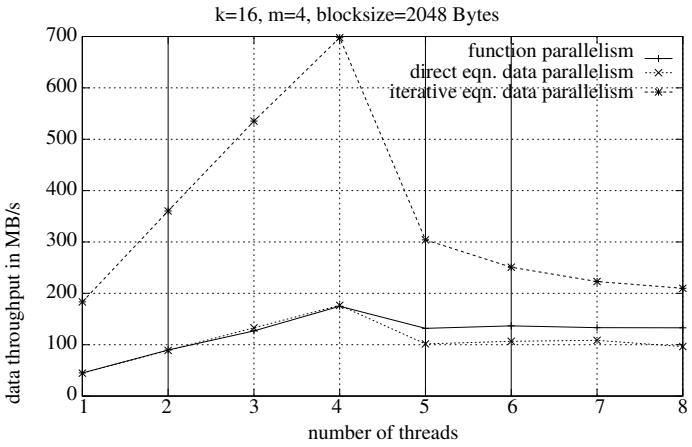


Fig. 8. Data throughput on a 4-core system

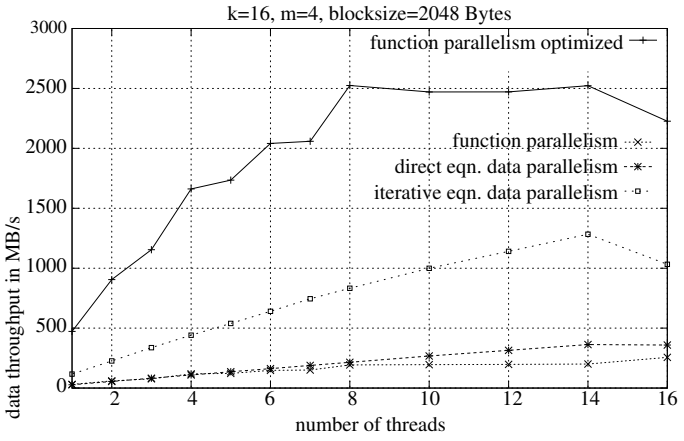


Fig. 9. Data throughput on a 8-core system, including the optimized version for SSE vectorization

After optimizing the function-parallel encoding function by restricted pointer parameters, the measurements were repeated on a 8-core system (see Fig. 9). Besides the two data-parallel coding functions that scale well up to 8 cores, the function-parallel coding variant showed a beneficial performance over the data-parallel versions.

At the current status, the performance analysis revealed that without optimizations the function-parallel does not perform better than the data parallel variant. In the best case, one can achieve a near optimal distribution of the computational load that is comparable to the best possible load distribution of data-parallel coding. Regardless, it is beneficial to provide variants for function-parallel coding when there is a chance for compiler-based optimizations. We could show that gcc generates faster code from the function-parallel variant, compared to the data-parallel variant.

Further investigations are directed to use iterative equations for the functional parallelism as well and to distribute the equations in a way that common subexpressions within the functions are taken into account for the distribution of equations. When there are more cores than equations, a combination of function parallelism and data parallelism should offer additional room for performance improvements.

V. Summary

Coding for failure-tolerant storage can be significantly accelerated by multithreading on multicore computer systems. We demonstrated the automated program code generation that includes the placement of OpenMP directives for data-parallel processing. Function-parallel processing is possible as well, but requires a slightly more code-modifying technique to assign operations to threads. With the current optimizations, the function-parallel variant reaches the best performance of 2.5 GByte/s on 8 cores.
Data en- and decoding is an example of applications that base on relatively complex mathematical principles but show a simple and regular code structure. In such a case, automatic generation of program code including the control statements for parallel execution is a feasible technique, compared to flexible and high-optimized code libraries.

References

[1] "OpenMP Application Program Interface, Version 3.1," 2011, The OpenMP Architecture Review Board. [Online]. Available: http://www.openmp.org/mp-documents/OpenMP3.1.pdf

[2] I. S. Reed and G. Solomon, "Polynomial Codes Over Certain Finite Fields," *Journal of the Society for Industrial and Applied Mathematics*, vol. 8, no. 2, p. 300, 1960.

[3] J. Bloemer, M. Kalfane, R. Karp, M. Karpinski, M. Luby, and D. Zuckerman, "An XOR-Based Erasure-Resilient Coding Scheme," International Computer Science Institute, Technical Report TR-95-048, Aug. 1995.

[4] P. Sobe and K. Peter, "Flexible Parameterization of XOR based Codes for Distributed Storage," in *2008 Seventh IEEE International Symposium on Network Computing and Applications*, Cambridge, MA, USA, Jul. 2008, pp. 101–110. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4579645

[5] P. Sobe, "cauchyrs Documentation," University of Luebeck, Institute of Computer Engineering, Tech. Rep., Sep. 2009.

[6] A. Wiebalck, P. Breuer, V. Lindenstruth, and T. Steinbeck, "Fault-Tolerant Distributed Mass Storage for LHC Computing," in *CCGrid 2003*, 2003.

[7] P. Sobe, "Parallel Reed/Solomon Coding on Multicore Processors," in *International Workshop on Storage Network Architecture and Parallel I/Os*. IEEE Computer Society, 2010, pp. 71–80.

[8] P. Sobe, "Parallel Coding for Storage Systems - An OpenMP and OpenCL-capable Framework," in *PASA- Workshop, GI Proceedings on ARCS (Workshops)*, 2012.

[9] M. L. Curry, A. Skejellum, H. L. Ward, and R. Brightwell, "Acellerating Reed-Solomon Coding in RAID Systems with GPUs," in *Proceedings of the 22nd IEEE Int. Parallel and Distributed Processing Symposium*. IEEE Computer Society, 2008.

[10] M. L. Curry, H. L. Ward, A. Skjellum, and R. Brightwell, "A lightweight, gpu-based software raid system," *2012 41st International Conference on Parallel Processing*, vol. 0, pp. 565–572, 2010.

[11] J. S. Plank, "Jerasure: A Library in C/C++ Fasciliating Erasure Coding to Storage Applications," University of Tennessee, Tech. Rep. CS-07-603, September 2007.

[12] U. Dastgeer, J. Enmyren, and C. Kessler, "Auto-tuning SkePU: A Multi-Backend Skeleton Programming Framework for Multi GPU Systems," in *Proceedings of the 4th International Workshop on Multicore Software Engineering*. ACM, 2011.

A comparison of CUDA and OpenACC: Accelerating the Tsunami Simulation EasyWave

Steffen Christgau, Johannes Spazier, Bettina Schnor
Institute of Computer Science
University of Potsdam
August-Bebel-Straße 89
14482 Potsdam
{*christgau, spazier, schnor*}*@cs.uni-potsdam.de*

Martin Hammitzsch, Andrey Babeyko, Joachim Wächter
GFZ German Research Centre for Geosciences
Telegrafenberg
14473 Potsdam
{*martin.hammitzsch, babeyko, wae*}*@gfz-potsdam.de*

***Abstract*—This paper presents an GPU accelerated version of the tsunami simulation EasyWave. Using two different GPU generations (Nvidia Tesla and Fermi) different optimization techniques were applied to the application following the principle of locality. Their performance impact was analyzed for both hardware generations. The Fermi GPU not only has more cores, but also possesses a L2 cache shared by all streaming multiprocessors. It is revealed that even the most tuned code on the Tesla does not reach the performance of the unoptimized code on the Fermi GPU. Further, a comparison between CUDA and OpenACC shows that the platform independent approach does not reach the speed of the native CUDA code. A deeper analysis shows that memory access patterns have a critical impact on the compute kernels' performance, although this seems to be caused by the compiler in use.**

I. Introduction and Motivation

Within the EU-co-funded project *Collaborative, Complex and Critical Decision-Support in Evolving Crises Project (TRIDEC)* an early warning system for tsunamis is developed [1]. In case of a seismic event, the warning center has to evaluate the probability, the dimension and the locality of a potential tsunami based on the gathered sensor data. One of the core components of TRIDEC is the simulation *EasyWave*. It uses the sensor data as well as topology and bathymetric information and computes characteristics of the tsunami, e.g. the wave heights and coastal impact times in the affected regions.[2] A visualization of EasyWave's output is shown in Figure 1.

As computational time is a critical aspect in its use-case, EasyWave should execute as fast as possible. Modern GPUs offer capabilities to solve massively parallel problems in short times. Further, they are achievable and can be easily integrated within the compute server of the Early Warning System. Therefore, GPUs have been chosen as target platform for the parallel version.[1]

For the parallelization of EasyWave, there were different programming models available. Since our test systems were equipped with NVIDIA cards, a native CUDA [4] implementation was one option. Another choice was the use of OpenCL [5] or OpenACC [6]. While CUDA is tied to NVIDIA's GPUs, the OpenCL standard addresses different vendors and hardware platforms (GPUs, CPUs, Clusters etc.). Further the OpenACC standard resembles the OpenMP approach for shared memory programming: With few to no

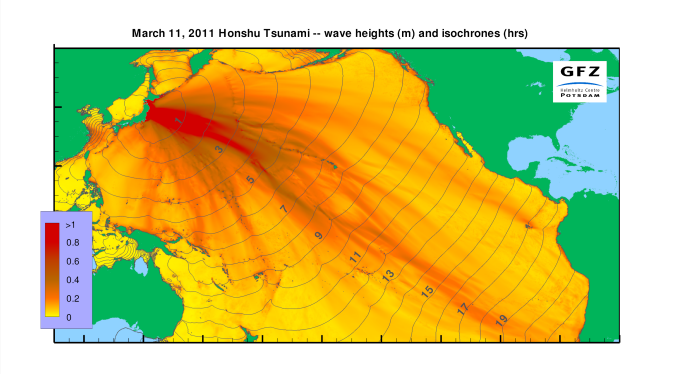


Figure 1. Visualization of the Honshu Tsunami in March 2011 showing the heights and propagation of the tsunami wave. [3]

changes in the original source, compiler directives gives hints on the automatic translation of the sequential source code in a parallel version for the according hardware architecture. In case of OpenACC, the target platform are hardware accelerators like GPUs.

A comparison of the effort-performance ratio of OpenCL and OpenACC for two real-world applications was given in [7]. The authors conclude from their experiments that OpenACC offers a promising ratio of development effort to performance. The experiments were done on a NVIDIA Tesla C2015 with an early OpenACC implementation by Cray. The OpenACC implementation shows 80 % resp. 40 % of the OpenCL performance, depending on the application.

Since a native CUDA implementation should result in the best performance, we decided to use CUDA in the first step. In the following, the CUDA version of EasyWave was tuned using different techniques to exploit the capabilities of the GPU hardware. Since those optimizations tend to be hardware specific, the expected improvements on the performance was verified on different GPU hardware available in the two participating research institutes.

On one hand, these optimizations have improved the speedup of the application, but on the other hand they were time-consuming and needed knowledge about the underlying hardware. Additionally, they are likely to be specific to a certain generation of the compute hardware. This requirement

makes it difficult for a non-expert programmer to easily write efficient code. This gave us motivation to implement also a parallel GPU version based on OpenACC which allows to mark code to be offloaded to accelerator hardware, e.g. GPUs.

The remainder of the paper is organized as follows: Section II and III give an overview over the sequential version of EasyWave and the experimental environment. In the following section, the design of the CUDA implementation is presented and different performance optimizations are discussed. Section V presents the OpenACC variant of the accelerated Tsunami simulation, followed by the conclusion.

II. Tsunami Simulation EasyWave

A. Sequential Version

The sequential version EasyWave [8] is written in C++ and uses a grid that represents the geographical area potentially affected by a tsunami. In most cases, the grid uses two dimensions with a granularity of two arc minutes. Usual scenarios of real-world incidents simulated with EasyWave have dimensions of about 2800×1800 grid points. For each grid point, the current and maximum wave height as well as physical water fluxes are stored in separated single precision floating point arrays. The pointers to these arrays are organized in another array leading to an array of pointers.

Due to the time-critical aspects of the simulation, simplified computational means like linear approximations are used [9]. Further, the physical aspects of wave propagation allow the usage of a window that restricts the computation: Only grid points within the window have to be computed. At the end of a single time step, i.e. after computing wave heights and fluxes for all points within the window, the borders of the window are analyzed to determine the need for an expansion of the window, which would be extended dynamically in that case. It has to be noted that data dependencies exist between the computation of fluxes and wave heights.

Within the window, the update of the wave height is done with a three-point stencil computation. It uses the upper and the left neighbor of the current cell as well as the current cell itself (TLC-stencil). Similar, the update of the fluxes in each grid point is done with a three-point stencil as well, but uses the right and the lower neighbor (BRC-stencil). In any case the update is done by iterating over the horizontal lines of the grid. Thus, the accesses to the array elements are ideal for the CPU's caches.

B. Simulation Scenarios

The experimental data used in this paper is from the earth quake in Begkulu, Sumatra, on September 12, 2007. The dataset has a grid size of 2851×1801, leading to a memory usage of 233 MB. The simulated time equals 10 hours, requiring 7200 time steps, i.e. 5 seconds per time step. The experiments described in the paper were also conducted with three other datasets having different sizes and geographical data, but led to the same performance results.

III. Hardware Environment

The hardware used for tuning and analyzing the program is presented in Table I. The GPU cards have different CUDA compute capabilities and a different total number of cores whereas System A with a Tesla C1060 possesses only about the half of compute cores as System B having a Tesla C2075. The architectural difference between the GPUs is illustrated in Figure 2: The Tesla C2075 includes caches which were introduced with compute capability 2.0 respectively the Fermi hardware architecture. In detail, the C2075 possesses a 768 KB unified L2 cache that is shared by all streaming multiprocessors (SM) and individual L1 caches for the SMs of configurable size. In the presented experiments a cache size of 48 KB was used.

Property	**System A**	**System B**
GPU product name	Tesla C1060	Tesla C2075
HW Architecture	Tesla	Fermi
Compute Capability	1.3	2.0
Multi Processors (SM)	30	14
Cores per SM	8	32
Cores (total)	240	448
Global Memory	4096 MB	5375 MB (ECC)
Caches	none	L1 per SM, L2 for all SM
Driver version	304.88	310.32
CPU	Intel Xeon E5520	Intel Xeon E5-1603
Cores	4 Cores, 1 Socket	4 Cores, 1 Socket
Frequency	2.27 GHz	2.8 GHz
Main Memory	24 GB	8 GB

Table I. Specification of the experimental hardware environment.

On the software side, we used the CUDA 5.0 toolkit to compile the CUDA based version. The C++ compiler for the CPU code was `g++` from the GNU Compiler Collection version 4.6. For OpenACC, the PGI Compiler version 13.6 has been used. Although alternatives from the academic [10] as well as the industry supported open source community [11] are available, the commercial product was chosen as we expected increased quality of the compiled code from a vendor collaborating with the hardware manufacture.[1] Moreover, the OpenACC branch of the GCC seems still to be experimental and did not work in our setup.

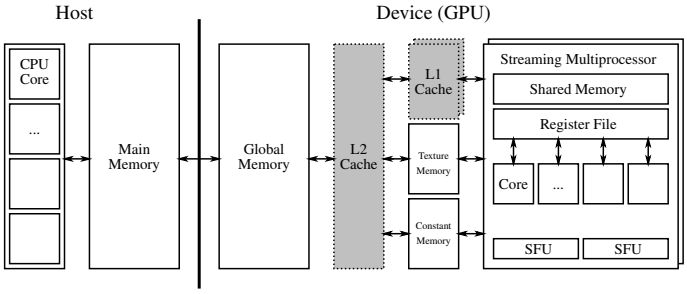


Figure 2. Overview of the memory hierarchy and GPUs used in this paper. Gray/dotted components are only available on the Fermi based Tesla C2075 device.

IV. CUDA Implementation

The sequential version of EasyWave was ported to the GPU using CUDA first. The implementation was incrementally optimized. After each optimization step, the improvement on the performance of the application was analyzed. In the following sections, the optimizations are discussed in detail.

[1] PGI was acquired by NVIDIA during the progress of the work presented in this paper.

A. Algorithmic Changes

1) Grid Point Update on GPU: According to the programming model of the GPU hardware, the sequential algorithm using nested loops (see Section II) had to be rewritten following the SIMD resp. SIMT paradigm. Thus, for the EasyWave port each GPU thread computes the physical properties of a single grid point. To ensure completion of the individual computational steps, computing the wave heights, fluxes, and the decision to expand the window is done in separated kernels.

This straightforward parallelization already delivered a substantial speedup of the application: The sequential version requires 348 seconds on the CPU of System A, respectively 305 seconds on System B's CPU. On the according GPUs the runtime is reduced to 162 seconds (Tesla C1060) and 28,4 seconds (Tesla C2075). This equals a speedup of 2.15 for the Tesla-based card, whereas the Fermi achieves a speedup of 10.7.

2) Parallel Window Extension: In the parallel version described above, the extension of the computational window is carried out by a single GPU thread. This can be parallelized as follows: The threads test in parallel if the threshold of the boundary cells residing in their computational window has been reached. For synchronization atomic instructions are used to store a boolean (non-zero) value that signals the extension of a window boundary. If an extension is necessary, this step is performed on the GPU by a single thread.

Compared to the GPU port of the main loop, the parallelized window extension reduces the runtime on the C1060 to 142 seconds, thus improving the performance by 13 %. The improvement is even more significant on the C2075 where the speed is enhanced by 46 % leading to a runtime of 15,3 seconds.

B. Hardware-specific Optimizations

Further performance improvements can be achieved by adapting the code to the architectural demands of the GPUs.

1) Memory Alignment: One of the most common tuning steps is to ensure the alignment of the memory used for the computation. Therefore, the memory used for the computation was allocated using `cudaMallocPitch`/`cudaMemcpy2D` to enable the hardware to optimize the memory accesses.

This modification resulted in a 4 % lower runtime for the C1060 card, and in negligible improvement for the C2075. The reason for this behaviour may be the additional computation for adjusting the boundaries of the computed window to the aligned memory addresses. Again, the use of CPU caches makes software improvements unnecessary.

2) Call by Value: A further enhancement was achieved by changing the way the arguments are passed to the kernel functions. Originally, the main array containing the pointers to the data arrays (see Section II) was passed directly to the GPU. When referencing an element in an data array, two accesses to the GPU memory where required: first, picking the element from the pointer array that contained the pointer to the data array. Then the required data element was accessed. This access pattern did not involve performance issues on the CPU as the cache would hold the values of the pointer array after an access. As the Tesla generation of Nvidia GPUs do not provide caches, the double memory access slows down the computation. Moreover, the parallel read on the pointer array is serialized by the GPU hardware and results in additional performance loss. This applies as well to variables that are constant during the execution of a kernel, like the current boundaries of the compute window.

To avoid these issues, all data arrays were passed directly to the kernel as pointer (in contrast to a pointer to a pointer array). Compared to the aligned memory optimization, the runtime of computation could be reduced by further 41% on the Tesla C1060. On the other hand, for the Fermi-based C2075 a relatively small improvement of about 6 % could be observed. As this card provides caches, the optimization does not come as much into effect as on the cache-less C1060.

3) Shared Memory: As the global memory containing the important computational data is the slowest memory type available on the GPU, avoiding accesses is likely to increase the speed of the computation. This is especially true for the stencil computations that are used by EasyWave and are memory-bound. Since the Tesla-generation cards do not have caches, using the Shared Memory of the Multiprocessors is often suggested as software-managed substitute. This common optimization technique was applied for both cards: Before running the computation, the area computed by a multiprocessor is loaded into its shared memory. When computation is finished on this scratch pad memory the computed values are stored back.

Running the kernel using shared memory significantly improves the performance on the Tesla-architecture card (C1060). Compared to the call by value version, the runtime is reduced to 55 %. In total, this leads to a runtime that is only about a fifth of the naïve port that transferred the core computation to the GPU (see Section IV-A). In contrast, when using shared memory on the Fermi-based C2075, the runtime increases compared to call by value version of the application. This can be accounted to the additional computational effort to copy data in and out of the shared memory emulating the cache's functionality, which is already present in the hardware.

C. Comparison

When comparing the performance of all optimization versions on both GPUs (see Figure 3), it is obvious that the optimizations done on the older Tesla C1060 card improve the performance significantly. Although, much effort and knowledge of the underlying hardware is required by the application programmer to achieve this speedup. Moreover, some well-known and often suggested tuning approaches, such as ensuring memory alignment, did not have the large impact as expected. Compared with changing the parameter passing method, the effort-benefit ratio of the latter seems to be better, nevertheless the gain in performance is surprising.

Comparing the performance of the algorithm between both cards, the most optimized version on the old-generation card gets very close to the performance of the Fermi-Card running the application with only the algorithm adjusted to the GPU's architecture (see Section IV-A). Concerning productivity, it is therefore reasonable to acquire hardware based on recent hardware architecture. Moreover, certain optimization techniques

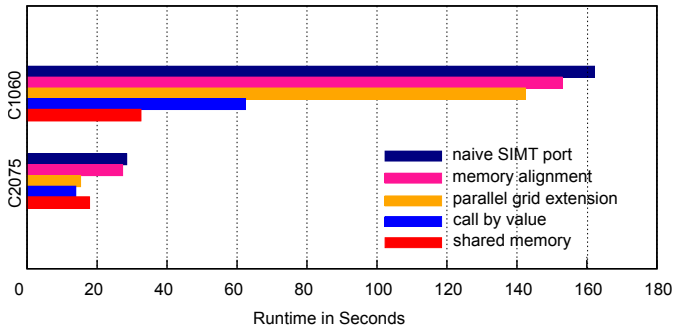


Figure 3. Runtime comparison of the discussed optimizations on both GPUs

leading to large performance gains on old hardware seem to provide low benefits on modern GPUs.

V. OpenACC Implementation

As intended by OpenACC, no changes in EasyWave's original sequential code, i.e. the loops performing the computation, were committed. OpenACC directives were added to ensure that data arrays reside in the GPU's memory and transfers between GPU and CPU are reduced to a minimum. To parallelize the code, the sequential `for`-loops were decorated with directives to convert the loops into kernels running on the GPU, i.e. `kernels` and `loop`. In total, this led to 21 additional lines of compiler directives compared to the sequential program having 462 lines of code. In comparison, 248 additional lines were required for the most tuned CUDA version of the application.

The OpenACC code was compiled for the different compute capabilities of the two GPUs using the PGI compiler version 13.6. On the Tesla C1060 a disappointing speedup of 1.15 compared to the sequential CPU version was achieved. Similar, on the system containing the Tesla C2075 the speedup was at 2.67 which is also much less than the native CUDA version discussed in Section IV-A1 achieved (speedup of 2.15 resp. 10.7).

A deeper analysis revealed that the parallelized loops exhibit different performance compared to the native and optimized CUDA version. The update of the wave height, which uses a TLC-stencil (see Section II) performs worse compared to both CUDA versions on both GPUs. In contrast, the OpenACC version of the flux update, that uses a BRC-stencil, outperforms the optimized CUDA code on the Tesla C2075, whereas it reaches similar performance on the C1060 as shown in Figure 4. We assume this as a compiler issue as the structure of of the functions is very similar and only differs in the memory access pattern. This problem has already been discussed with the vendor's support, but is still in discussion at the time of writing.

VI. Conclusion

This paper presents performance results of different parallel versions of the Tsunami simulation EasyWave. While the native CUDA version achieves a speedup of 10.7 on a Tesla C2075, the speedup of the OpenACC version was only 2.67 compared the sequential CPU version.

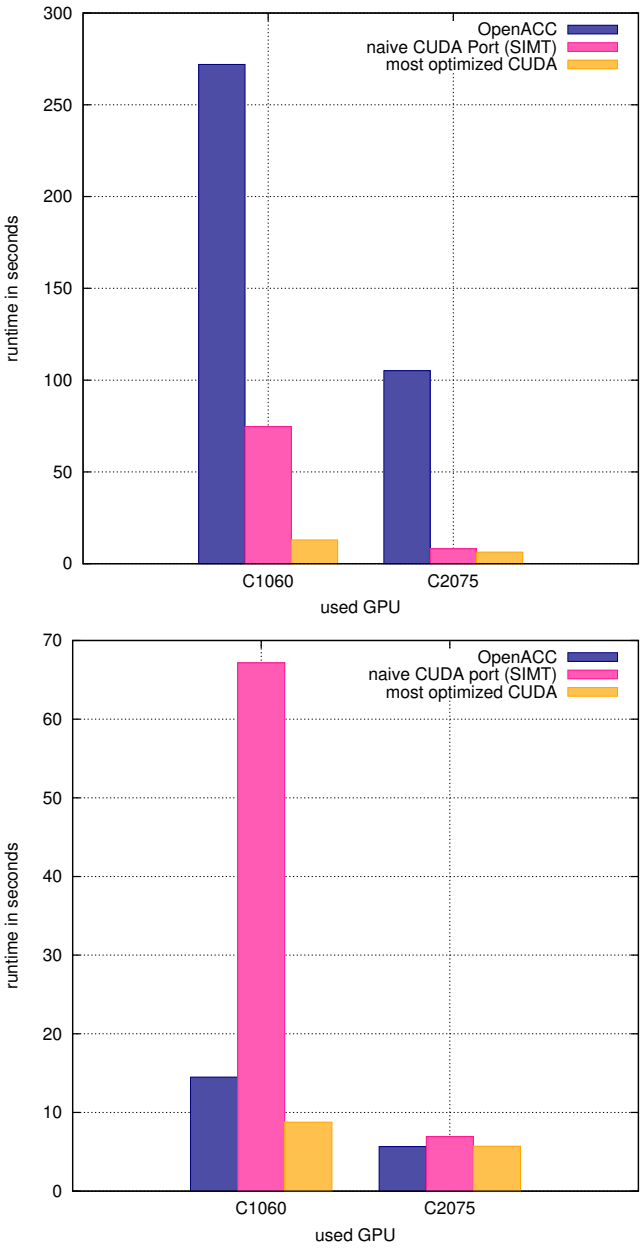


Figure 4. Performance of OpenACC compiled code for the loops performing the wave height update using TLC-stencil (top) and the flux update using BRC-stencil (bottom).

Further, it was shown that recent hardware has a positive effect on the computational speed when a program is directly programmed for CUDA. This is especially true if the source code is not tuned manually to exploit the features of the hardware but is only programmed following the SIMD paradigm of GPUs. In contrast, it is necessary for a programmer to be much more aware of hardware details to gain a significant performance improvement on old hardware. Thus, the usage of recent hardware can unburden a non-expert programmer from the task of tuning their specific application to the hardware.

Even more, from a (scientific) application programmer's perspective, the usage of Open-ACC seems to be promising as very few code changes are required to enable support

for accelerators. It was further shown that compute kernels differing only in the memory access pattern can result in very different performance when using OpenACC. This emphasizes the crucial role of compiler support for the OpenACC standard.

REFERENCES

[1] J. Wächter, A. Babeyko, J. Fleischer, R. Häner, M. Hammitzsch, A. Kloth, and M. Lendholt, "Development of tsunami early warning systems and future challenges," *Natural Hazards and Earth System Science*, vol. 12, no. 6, pp. 1923–1935, 2012. [Online]. Available: http://www.nat-hazards-earth-syst-sci.net/12/1923/2012/

[2] M. Hammitzsch, F. J. Carrilho, O. Necmioglu, M. Lendholt, S. Reißland, J. Schulz, R. Omira, M. Comoglu, N. M. Ozel, and J. Wächter, "Meeting unesco-ioc icg/neamtws requirements and beyond with tridec's crisis management demonstrator for tsunamis," in *23rd International Ocean and Polar Engineering Conference - ISOPE*, Anchorage, USA, 2013.

[3] A. Babeyko, online, https://media.gfz-potsdam.de/gfz/wv/05_Medien_Kommunikation/Bildarchiv/Erdbeben_Japan/temp_xs/33408642-110313_Tsunami_Japan_2_DRUCK.png.

[4] E. Lindholm, J. Nickolls, S. Oberman, and J. Montrym, "Nvidia tesla: A unified graphics and computing architecture," *Micro, IEEE*, vol. 28, no. 2, pp. 39–55, 2008.

[5] J. E. Stone, D. Gohara, and G. Shi, "OpenCL: A Parallel Programming Standard for Heterogeneous Computing Systems," *Computing in Science and Engineering*, vol. 12, no. 3, pp. 66–73, 2010.

[6] openacc.org, *The OpenACC Application Programming Interface*, 2011, version 1.0, http://www.openacc.org/sites/default/files/OpenACC.1.0_0.pdf. [Online]. Available: http://www.openacc.org/sites/default/files/OpenACC.1.0_0.pdf

[7] S. Wienke, P. Springer, C. Terboven, and D. Mey, "OpenACC — First Experiences with Real-World Applications," in *Euro-Par 2012 Parallel Processing*, ser. Lecture Notes in Computer Science, C. Kaklamanis, T. Papatheodorou, and P. G. Spirakis, Eds. Springer Berlin Heidelberg, 2012, vol. 7484, pp. 859–870. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-32820-6_85

[8] D. J. Greenslade, A. Annunziato, A. Y. Babeyko, D. R. Burbidge, E. Ellguth, N. Horspool, T. S. Kumar, C. P. Kumar, C. W. Moore, N. Rakowsky, T. Riedlinger, A. Ruangrassamee, P. Srivihok, and V. V. Titov, "An assessment of the diversity in scenario-based tsunami forecasts for the indian ocean," *Continental Shelf Research*, 2013. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0278434313002021

[9] A. Babeyko, "EasyWave: fast tsunami simulation tool for early warning," February 2012, ftp://ftp.gfz-potsdam.de/pub/home/mod/babeyko/easyWave/easyWave_About.pdf.

[10] R. Reyes, I. López-Rodríguez, J. J. Fumero, and F. Sande, "accULL: An OpenACC Implementation with CUDA and OpenCL Support," in *Euro-Par 2012 Parallel Processing*, ser. Lecture Notes in Computer Science, C. Kaklamanis, T. Papatheodorou, and P. G. Spirakis, Eds. Springer Berlin Heidelberg, 2012, vol. 7484, pp. 871–882. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-32820-6_86

[11] E. Gavrin, "Openacc branch [openacc-1_0-branch]," Gnu GCC Mailing list, online http://gcc.gnu.org/ml/gcc/2013-09/msg00235.html, Sep 2013. [Online]. Available: http://gcc.gnu.org/ml/gcc/2013-09/msg00235.html

An Architecture Framework for Porting Applications to FPGAs

Fabian Nowak, Michael Bromberger and Wolfgang Karl
Karlsruhe Institute of Technology
Chair for Computer Architecture and Parallel Processing
76128 Karlsruhe, Germany
Email: <lastname>@kit.edu

***Abstract*—High-level language converters help creating FPGA-based accelerators and allow to rapidly come up with a working prototype. But the generated state machines do often not perform as optimal as hand-designed control units, and they require much area. Also, the created deep pipelines are not very efficient for small amounts of data. Our approach is an architecture framework of hand-coded building blocks (BBs). A microprogrammable control unit allows programming the BBs to perform computations in a data-flow style. We accelerate applications further by executing independent tasks in parallel on different BBs. Our microprogram implementation for the Conjugate-Gradient method on our data-driven, microprogrammable, task-parallel architecture framework on the Convey HC-1 is competitive with a 24-thread Intel Westmere system. It is $1.2\times$ faster using only one out of four available FPGAs, thereby proving its potential for accelerating numerical applications. Moreover, we show that hardware developers can change the BBs and thereby reduce iteration count of a numerical algorithm like the Conjugate-Gradient method to less than $0.5\times$ due to more precise operations inside the BBs, speeding up execution time $2.47\times$.**

I. Introduction

Once being too small in terms of area and too slow in terms of clock rate, FPGAs are now sufficiently large and fast to accelerate complex algorithms. They are employed in several different target domains, ranging from digital signal processing to scientific computations. In the latter domain, processing floating-point data is of paramount importance.

Porting algorithms to FPGAs and implementing them requires great efforts. The developer has to cope with implicit parallelism, synchronization issues, clock rates, and data paths. As a result, tools for converting sequential high-level language code to hardware descriptions came up [6], trying to ease hardware development and to make FPGA hardware programmable for high-level application programmers. Each of these converters makes assumptions about data exchange, synchronization, hardware capabilities and interfaces. Much work is required to successfully interface with the surrounding hardware. Taking this into account, the generated designs are not portable from one FPGA environment to the other. To exchange data between interfaces and the generated or instantiated arithmetic-logical circuits, state machines are generated by the converters, which consume much space due to many required states when controlling pipelining and targeting data-driven execution. Although development is facilitated, the time-consuming hardware synthesis is still required, and the development cycle still consists of describing the algorithm, simulating, emulating, synthesizing and finally testing the resulting hardware design. Overall, the existing methodologies for creating hardware designs are quite cumbersome.

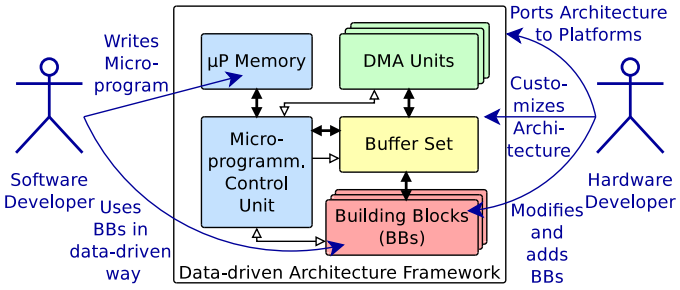


Figure 1. Data-driven architecture framework for FPGAs and responsibilities of hardware and software developers.

Our approach to this problem is microprogramming the control of medium-grained, preconfigured, existing building blocks (BBs), and calling the microprogram from software side on the host. We propose an architecture framework as depicted in Fig. 1 that consists of a microprogram memory, control unit, functional and data-transferring blocks, and of a decoupling buffer set to enable both data-driven and task-parallel execution. Once an FPGA is configured with our framework, an application developer needs to come up with microprogram implementations of the to-be-ported algorithm, but can ignore hardware constraints such as streaming, caching, data-parallel and task-parallel execution, resource usage, and clock rates. As changes to the microprogram can be tested within seconds, the developer is able to debug at software level and is freed from waiting several hours for the synthesis.

When using reconfigurable accelerators, data transfer poses a limiting factor. However, with our data-driven architecture framework, this is no longer the case as data are stored internally and reused for further computations. Targeting numerical programs, the framework allows accelerating the Conjugate-Gradient method [10], for example, because it is no longer memory-bound. To port further algorithms to the architecture framework, application developers only need to provide a different microprogram for the same building blocks. As a hardware developer, it is possible to change implementations of the BBs, e.g. for increased accuracy to reduce iteration count without changing the microprogram, though synthesis is required then.

Approaches for helping in FPGA designs are discussed in Section II. Next, Section III explains our concept. To

prove portability, we implemented the architecture on two different FPGA systems (Section IV). We present previous evaluation results [10] of the framework in Section V. To take previous work one step further, we evaluate ease of high-level programming and optimization, and possibility for low-level hardware adaptations. Through adaptations such as more precise internal operations inside the BBs, the convergence criterion of numerical algorithm is reached faster. This reduces iteration count to less than $0.5\times$. Section VI closes the article and gives a short outlook.

II. Related Work

For scientific computing, efficient floating-point operation is of great importance. Over the last ten years, FPGAs have become capable of performing floating-point calculations, which has been researched extensively with regard to area, clock rate, precision, pipelining [9]. Very helpful is the automated creation of floating-point cores [2]. With such hardware capabilities and software tools, FPGAs can provide benefit for scientific computing [3].

One major issue with FPGAs is to develop the design, then simulate it, synthesize and test it. Therefore, High-Level Languages (HLLs) evolved, and with fixed-point and floating-point cores for all kinds of operations, it became possible to translate arithmetic codes to hardware descriptions that instantiate available cores for arithmetic operations. Among such translators, ROCC Compiler [6] is probably known best. Besides the C-based translators, there are efforts to accelerate poorly performing Matlab code, coming up with a complete toolbox and design flow for FPGA acceleration [8].

Basis for many translators is the SUIF compiler [17] that creates an abstract syntax tree upon which the translators further produce state machines and wiring of the instantiated cores or generated arithmetic-logical circuitries. In tackling the large area consumption of state machines in FPGA designs with more than 30 states, microprogramming proved helpful and has found its way into FPGA research [1].

In the MORA architecture [16], no explicit control is necessary because the arithmetic instructions in the program code are translated to parameterized processing elements that are connected with one another according to the data graph of the code, and the elements execute in a data-driven fashion. But MORA lacks from being freely (re-)configurable to another application, again requiring high synthesis time for new application accelerators. Same applies to Altera's OpenCL path where the to-be-accelerated kernel function is translated to a data-parallel, deeply pipelined hardware description in Verilog [14].

Although the use of high-level languages and converters such as MORA, OpenCL and ROCCC facilitates the use of FPGAs by not requiring error-prone developing of hardware descriptions and no time-consuming Modelsim-based simulations, the general methodology remains the same. At first, an algorithm must be ported to the language itself. Secondly, especially with OpenCL, a suitable mapping must be found, i.e., optimal dimension and grid size. This task requires more than only basic understanding of the hardware. Third, the algorithm description must be simulated in its high-level-language version and then in its translated version, i.e. emulating the hardware. Fourth, hardware synthesis and place&route will take several minutes or even hours. Only then can the ported algorithm and its mapping be tested in real hardware, potentially detecting performance bottlenecks or even implementation errors. Unless data is written back to external memory after each single operation, tracking an error is rather difficult and requires the use of expert tools such as Chipscope. Every change to the high-level code requires another time-consuming hardware synthesis run instead of reusing previously translated and synthesized subcomponents such as a vector adder. This process must be repeated again and again until the performance goal is met and execution delivers correct results. Normally, this takes a couple of days. Programmers of scientific applications however need an efficient and easy-to-program means for exploiting FPGA-based accelerators in diverse floating-point-intense applications. SHARC [7] tries to close this gap by parameterizing the instantiated cores for further iterations, e.g. for processing different color values. The connection of the cores can be changed at runtime by instantiating and parameterizing additional switch modules. These parameters are extracted from Matlab code and cannot be programmed by the user in a convenient fashion.

As a further problem, data exchange with FPGA-external memories and internal data storage frequently pose a huge problem when employing accelerators. The high-level tools can only help to a limited amount in optimizing data transfer. Thus, the programmer has to care for efficient communication and data reuse, which is a non-trivial task that requires profound understanding of the underlying system. The average domain specialist application programmer should be freed from such optimization tasks. Aside, the resulting optimized hardware implementations are no longer portable from a performance point of view.

Much benefit is gained from FPGAs when implementing special implementations that operate at bit level or cannot be executed efficiently in general-purpose processors, such as a highly accurate dot product implementation [12].

The Xilinx Vivado suite [4], [18] allows to easily connect different Intellectual Properties (IP) cores to an embedded processor like ARM using the AMBA bus interface. It is possible to integrate IPs from Xilinx, third parties or custom IPs. An IP can be defined at different abstraction levels like C source code, RTL description in Verilog or VHDL or as netlist. The focus of Vivado is to realize Systems on Chips (SoCs) using different IP cores. Their approach suffers from the interconnection bottleneck when several data-hungry, memory-intense IP cores have to communicate over the shared bus with memory. Instead, our focus is to allow easy and efficient programming of accelerators that are included in high performance computing (HPC) systems.

Using a rather small control unit compared to a soft core inside an FPGA, we can integrate more building blocks (BBs) on an FPGA. These BBs communicate efficiently and directly by using a central buffer set so that no bus-sharing or complex bus protocol is required. Therefore, we propose to combine micro-programming, data-driven programming and execution, and preexisting cores, i.e, BBs. Thereby, programmers can exploit potential data-level parallelism inside the BBs, task-level parallelism by executing several BBs concurrently, and foremost pipeline parallelism where available, which helps

minimize data exchange with memory. In case special instructions shall be integrated, we favor the aproach suggested by Strozdka [15] of tightly collaborating mathematicians, i.e, domain specialists, and computer scientists, i.e, hardware experts, as is also depicted in Fig. 1.

III. Microprogrammable, Data-driven Architecture

We design the architecture as illustrated in Fig. 1 with portability, easy programmability and adaptivity in mind. Data are exchanged with external memory via DMA units and internally via a central buffer set to leverage data-driven execution. Execution is controlled via microprograms instead of fixed state machines that trigger the BBs, such as DMA or vector adder.

The microprograms are provided by users, e.g., domain specialists. In contrast, the BBs are implemented by hardware specialist. We achieve concurrency by data-driven execution of the BBs. Independent blocks can proceed completely concurrently, and dependent blocks can proceed in a pipelined fashion if output and input rates allow. The building blocks, in return, can exploit data-level parallelism. Internal operations in the BBs are pipelined, if possible. Different BBs can form a pipeline using the central buffer set. So, one BB (source) generates data that will be processed by another BB (drain) in later steps. Synchronization between BBs is achieved by the buffer set and therefore no extra mechanism is needed inside the control unit. The pipeline including different BBs is formed by the microprogramm. Hence, application domain specialists can employ the architecture according to their needs by only writing the microprogram without caring for the details of data transfer, data reuse or hardware synthesis and bitstream generation. This is also key to achieving portability of the architecture and of the developed microprograms.

In the microprogram assembly language, BBs are denoted by representative mnemonics, if possible, for example `vadd` to add a pair of vectors streamed via two input buffers. All instructions take registers or immediate values as arguments, and the 32 registers can be written to explicitly via `regw`. The central micro-programmable control unit depicted in Fig. 2 passes instructions to BBs or to the sequencer modifying the instruction pointer, or to the ALU for operations on the register set. No instruction pipelining is used inside the control unit because most microinstructions are asynchronous BB instructions that will occupy the BB for a long time. So, only litte performance gain can be achieved using instruction pipelining in our case. This aspect also saves resources on the FPGAs and therefore more BBs and buffers are instantiated.

As indicated in Fig. 3, BBs have two interfaces, one for instructions from the control unit, and the other for interaction with the buffer set. Assume the BB is a vector adder that consumes two input vectors from the buffer set and writes one result vector. Then the instruction is to add or subtract the vectors, and the parameter is the length of the vectors. Figure 4 gives the corresponding example microprogram for adding two vectors.

Figure 5 shows in detail the data flow between source and drain blocks via the central buffer set. Any unit is normally both source and drain, e.g, the vector adder consumes as a

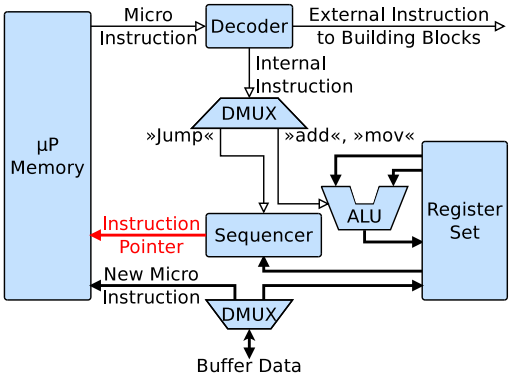


Figure 2. Micro-programmable control unit.

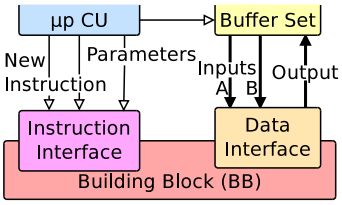


Figure 3. Instruction and data interface for BBs.

drain unit from the two source DMA units, and it is the source unit for the third DMA drain unit. The MUXes are assigned their configurations by `buf_assign`. A source block can write to any destination buffers due to the full crossbar interconnect. These are bound to exactly one drain block that consumes its data. The buffers are then filled by reading via DMA (`dmar`) or by a data-producing BB, and the outputs are consumed by writing to external memory via DMA (`dmaw`) or by another BB. In the end, a pipeline reading data from external memory, concurrently processing it within several BBs and already writing back some results can be established. Computation and communication in a data flow style support this achievement so that no more data dependencies need to be resolved manually. As shown in the program code in Figure 4, the domain specialist, non-hardware expert programmer only has to interconnect the blocks in the most natural data flow way. This can easily be accomplished via graphical user interfaces [13].

To generate machine-readable instructions from the assembly instructions, the toolchain sketched in Fig. 6 parses them and outputs binary code. It can also produce a coefficient file for static initialization of the microprogram memory at synthe-

```
regw R10, 20
buf_assign dma1, buf1
buf_assign dma2, buf2
dma1r 0xadd1, R10
dma2r 0xadd2, R10
buf_assign vadd1, buf3
vadd1 R10
dma3w 0xadd3, R10
j 0
```

Figure 4. Microprogram for adding two vectors.

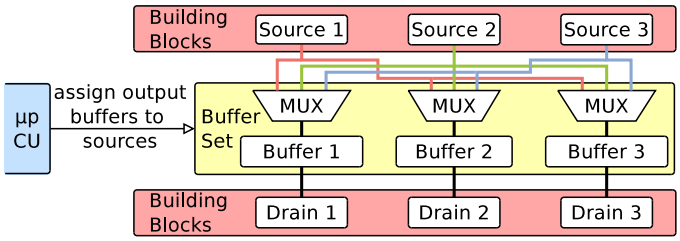


Figure 5. Buffer set and interface between building blocks.

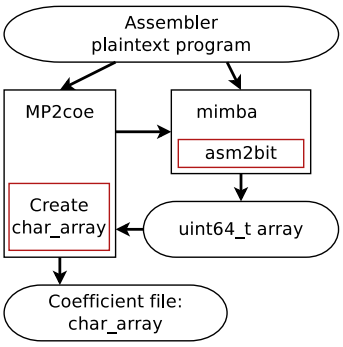


Figure 6. Toolchain parsing the assembly instructions, converting to binary code and generating memory configurations.

sis time. Otherwise, a driver mechanism needs to transfer the compiled microprogram. Depending on the system environment, this driver can even be embedded into the application.

IV. Framework Implementation and System Integration

To demonstrate portability, we implemented the architecture on different heterogeneous systems. The first implementation served as a prototype study and targeted the UoH HTX-Board that connects an FPGA via the HyperTransport Expansion (HTX) Socket to an AMD Opteron-based host system. Due to bad routing results on the chosen system with a Xilinx Virtex-4 FX100 FPGA, we do neither present detailed place&route results nor detailed evaluation results. Using two BBs, one for reading data and one for writing data, we can only process 500 MB data per second. The theoretically available bandwidth is 1600 MB/s per direction. On the one hand, there is overhead due to the protocol, and on the other hand, the host system does only allow 4MB contiguous DMA memory regions. Therefore, we can not fully exploit the bandwidth of the system. Hence, (pipeline-)parallel processing on the FPGA must compensate the low bandwidth usage by raising the computation-to-communication ratio. But integrating more BBs is out of scope for this system setup due to the tight timing results, unless running at a much lower frequency not optimally fitting the HyperTransport system.

To obtain performance-oriented results, we implemented the concept on the Convey HC-1 depicted in Fig. 7. We target supporting developers of numerical programs. Hence, we need several vector blocks that should execute in an overlapped fashion. Foremost, vectors need to be added and scaled (cmp. axpy operation). Secondly, dot products are required that can also be used for implementing matrix multiplications. Many solvers rely on stencils. Thus, we implemented a data-driven

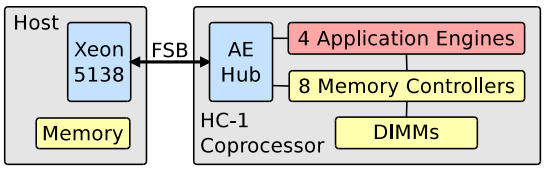


Figure 7. Convey HC-1 comprising 4 user-programmable FPGAs on a coprocessor board attached to CPU socket.

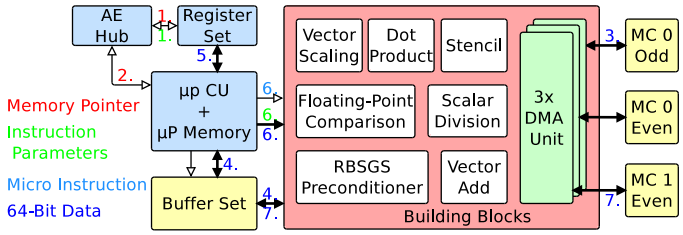


Figure 8. Implementation of the architecture framework on the Convey HC-1.

stencil unit. Upon this basis, a pipelined solver for Red-Black schemes was elaborated [10]. Moreover, a scalar division unit and comparison unit are required to calculate reciprocal values or divisors and to check when the algorithm has converged, respectively. Apart from these arithmetical blocks, data must be read from and written to external memory. Hence, a number of DMA units support several concurrent memory accesses in both read and write direction. As data is reused frequently due to the central buffer set, external memory access does no longer pose the main problem with reconfigurable computing.

The implemented framework, the arithmetical blocks and DMA units were implemented by hardware specialists. It is depicted in Fig. 8 together with the Convey AE interface, memory control logic, sequencer and buffer set. To use it, domain specialists only need to provide microprograms for controlling data flow, which might also be eased by providing a graphical user interface to connect the building blocks in data flow style.

Upon start of usage, the host application passes the memory references for the new, user-supplied microprogram, input data and output locations via the general register set (1.). Then (2.), the coprocessor is invoked, triggering the default microprogram to fetch the new microprogram from memory (3.). Having obtained the microprogram via a DMA unit and the buffer set (4.), the instructions of the new microprogram are decoded, the required registers are read (5.) and passed together with the opcode to the corresponding BB (6.). The blocks execute (7.) in a concurrent and data-driven fashion depending on the availability of data in the buffer sets.

V. Evaluation

We developed the data-driven, microprogrammable architecture for reconfigurable computing with special focus on developers of numerical applications. As case study for our evaluations, we employ the Conjugate-Gradient (CG) method, which is an iterative solver for linear equation systems. We solve the Poisson problem, so that we can employ a 2D stencil operation with problem-related coefficients instead of a matrix A and also a stencil-based preconditioner.

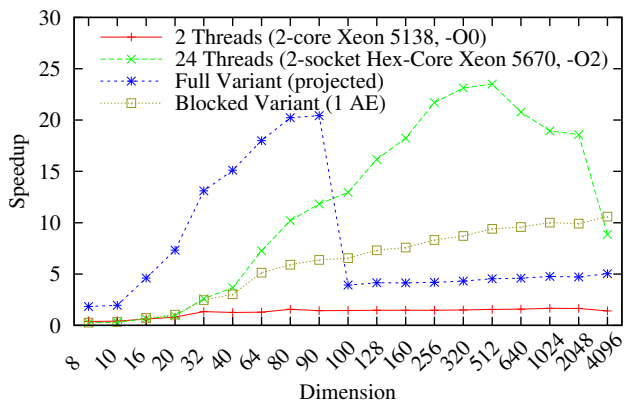


Figure 9. Speedup of CG method on architecture framework on Xilinx Virtex-5 LX330 and on Intel Xeon X5670 against Convey HC-1's Intel Xeon 5138.

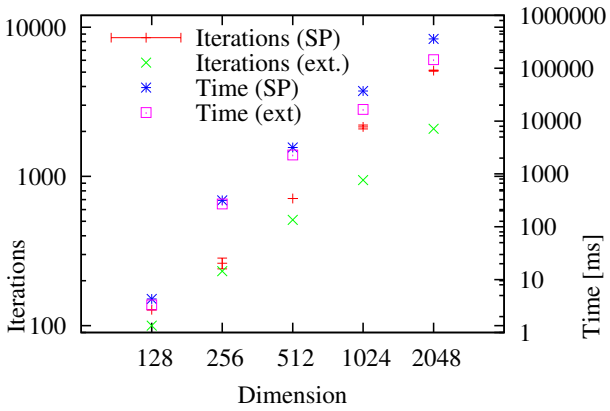


Figure 10. Reducing number of iterations from single-precision (SP) to higher precision (ext) for the dot product.

A. Accelerating the Conjugate-Gradient Method

To fully port the CG method, we would need 13 vector blocks and 9 DMA units. Among the 13 blocks are 3 vector adders, 3 vector-scaling blocks, 2 dividers, 2 dot products, 1 floating-point comparison, 1 stencil unit, and 1 red-black symmetric Gauss-Seidel preconditioner consisting of two differently preconfigured stencil units. Instantiating all these blocks requires a vast amount of BRAMs and DSP units and produces a congested design with densely packed control logic so that 150 MHz timing is not feasible. Hence, we split the main loop of the CG method into 3 distinct parts ("Blocked Variant") so that only one instance of each unit is required, together with only 3 DMA units. The implementation results of the fully functional design are given in Table I. We obtained speedup of $10\times$ over sequential execution time on host processor Intel Xeon 5138 of the the Convey HC-1 as illustrated in Fig. 9 [10]. The untimable design is also displayed, labelled "Full Variant", based on early measurements and a simple timing model. It can provide its massive speedup when its small 90×90 buffers suffice for handling the input data. Our blocked design is even $1.2\times$ as fast as a 24-thread version for large dimensions with optimization turned on (-O2).

B. High-Level Optimization

It might be advantageous to change the order in which the micro instructions are called so that more data would be transferred earlier, thereby potentially enhancing memory bandwidth usage. Knowing the best order is also important for automatically synthesizing the microprograms. Hence, we evaluated three versions of the microprograms for the CG blocks. In the first, original version, DMA operations are slightly optimized toward fetching data while also executing independent operations. We aggressively optimize the second version toward starting all DMA operations as early as possible. And finally, the third version calls DMA operations as late as possible. From the results given in Table II, we can conclude that the order plays only a minor role because a new instruction is fetched, decoded and executed every few cycles and the micro-programmable control unit (μpCU) is not stalled, but only waits for availability of the BB before passing the opcode and parameters. For larger data, it seems slightly advantageous to start DMA operations rather later than sooner, thereby overlapping computation with data transfers as early and much as possible. However, the results do not provide statistical relevance. With this experiment, we have shown that no deep understanding of FPGA development and underlying principles is required from application programmers when using this architecture. Further, optimally exploiting the architecture reduces to changing the microprograms only, neither requiring any synthesis nor reconfiguration. Similarly, software developers only need provide other microprograms for executing other algorithms on the BBs.

C. Low-level Hardware Adaptation

When executing until the convergence criterion is met, our FPGA-supported CG method requires many more iterations than its dimension, and moreover, the amount of iterations varies, similarly to calculating dot products with OpenMP and dynamic scheduling. To find the root cause of this behaviour, we wrote the intermediate results back to coprocessor memory and checked the values. We could track the cause down to the dot product implementation, which in order to yield pipelining stores partial products until they can be fully accumulated, thus not being deterministic. Ideally, one would integrate an exact dot product implementation [12], [11] because with the data-transfer problem being solved now, the required tight integration of the special unit is achieved. Though, to save area on the FPGA, we only extended the instantiated floating-point cores by 23 and 32 additional mantissa bits, respectively. Both the multiplication and the internal accumulator do now suffer less from rounding and the typical windowing problem now [5]. As Fig. 10 indicates, we can thereby already halve the number of iterations and reduce application time up to 2.47 times.

VI. Conclusions and Outlook

Today, there exist several approaches to port scientific, floating-point-based applications onto reconfigurable logic. In this article, we propose a microprogrammable architecture together with data-driven execution of the building blocks (BBs) to conveniently and efficiently employ FPGA-based accelerators in scientific computing. The BBs on the FPGA work internally data-parallel and thereby optimally exploit the available bandwidth, especially with regard to the memory controllers on the Convey HC-1 coprocessor. Task-level

Table I. RESOURCE USAGE OF THE SINGLE-INSTANCE CONVEY HC-1 DESIGN ON A XILINX VIRTEX-5 LX330 ("BLOCKED VARIANT").

Resource	Slices	LUTs	Registers	DSPs	BRAMS	Max. Freq.
Number	34520	89447	86379	110	166	6.643 ns
Usage	66 %	43 %	41 %	57 %	57 %	150.5 MHz

Table II. AVERAGE EXECUTION TIMES OVER AT LEAST 5 RUNS AND SPEEDUP FOR DIFFERENT SCHEDULING OF THE DMA OPERATIONS WITH VARIYNG PROBLEM SIZES.

	320	512	640	1024	2048	4096
Time $(m.s)$						
Original	517.7	2269.8	4425.9	16558.8	145459.4	1171274.3
DMA early	518.7	2269.3	4425.6	16558.9	145478.9	1171063.9
DMA late	517.9	2270.9	4424.8	16557.3	145457.7	1171091.2
Speedup						
DMA early	0.99805	1.00021	1.00007	0.99999	0.99987	1.00018
DMA late	0.99967	0.99953	1.00025	1.00009	1.00001	1.00016

parallelism is achieved by executing several units concurrently, called subsequently from a control unit. It enables fully leveraging the massively parallel FPGA hardware. A central buffer set allows streaming data from DMA units through potentially several BBs, termed pipeline parallelism, and finally writing back to coprocessor memory so that data reuse is maximized and transfer from/to memory minimized. For a given application domain such as solving equations, a set of BBs needs to be developed once. Domain specialists are then freed from creating reconfigurable designs because they only need to develop the microprograms for other algorithms to be executed on the framework.

The architecture was implemented and tested on two different systems. Evaluation of a microprogram for a preconditioned Conjugate-Gradient method on only a single FPGA against an OpenMP implementation running with 24 hardware threads already yielded slight speedup of $1.2\times$ [10]. We showed its ability to quickly try optimizing execution time by changing the microprogram only. No additional time-consuming synthesis runs were required. Further, we extended the dot product unit by additional bits for more accurate results to model the case that special-purpose units are integrated into the domain-specific BB set. Thereby, we could decrease the number of required iterations of our test case. The benefit of embedding special arithmetic units, e.g. for very exact arithmetics, should be evaluated in future.

We have thereby also developed a novel methodology. From an application programmer's perspective, development starts with connecting function blocks, e.g, in a regular C program, from a software library that is formulated in a data-driven way and whose functions can execute concurrently, i.e, in a task-parallel fashion. This program can already serve as a highly-performant program version because first investigations showed that data-driven, task-parallel execution can provide competitive performance to data-parallel execution due to enhanced data locality. Having mapped the algorithm onto function blocks for the software version, the microprogram must be formulated to connect the building blocks via the central buffer set for execution in reconfigurable accelerator hardware. The microprogram only needs to be compiled, then the algorithm implementation can already be tested in our architecture framework. With both a software and a hardware implementation, the programmer can easily and stepwise run distinct parts of the application on the accelerator hardware. The data transfer problem when using acclerators is solved by exploiting data flow principles, and therefore the programmer can efficiently and conveniently benefit from accelerator hardware. We envision that automatic design tools will create such microprograms instead of generating huge state machines in low-level hardware languages.

REFERENCES

[1] B. Bomar, "Implementation of microprogrammed control in FPGAs," *IEEE Trans. on Industrial Electronics*, vol. 49, no. 2, pp. 415–422, April 2002.

[2] F. de Dinechin, C. Klein, and B. Pasca, "Generating high-performance custom floating-point pipelines," in *19th International Conference on Field Programmable Logic and Applications*. IEEE, August 2009.

[3] D. DuBois, A. DuBois, T. Boorman, C. Connor, and S. Poole, "An Implementation of the Conjugate Gradient Algorithm on FPGAs," in *16th Int. Symp. on Field-Programmable Custom Computing Machines*. IEEE, April 2008, pp. 296–297.

[4] T. Feist, "Vivado Design Suite," Xilinx, White Paper Version 1.1, June 2012. [Online]. Available: http://www.xilinx.com/support/documentation/white_papers/wp416-Vivado-Design-Suite.pdf

[5] D. Goldberg, "What Every Computer Scientist Should Know About Floating-Point Arithmetic," *ACM Computing Surveys*, vol. 23, no. 1, pp. 5–48, March 1991.

[6] Z. Guo, W. Najjar, and B. Buyukkurt, "Efficient hardware code generation for FPGAs," *ACM Trans. on Architecture and Code Optimization*, vol. 5, no. 1, pp. 1–26, 2008.

[7] S. Kestur, D. Dantara, and V. Narayanan, "SHARC: A streaming model for FPGA accelerators and its application to Saliency," in *Design, Automation Test in Europe*, March 2011, pp. 1–6.

[8] J. S. Kim, L. Deng, P. Mangalagiri, K. Irick, K. Sobti, M. Kandemir, V. Narayanan, C. Chakrabarti, N. Pitsianis, and X. Sun, "An automated framework for accelerating numerical algorithms on reconfigurable platforms using algorithmic/architectural optimization," *IEEE Trans. on Computers*, vol. 58, no. 12, pp. 1654–1667, 2009.

[9] G. Morris, "Floating-Point Computations on Reconfigurable Computers," in *DoD High Performance Computing Modernization Program Users Group Conference*. IEEE, 2007, pp. 339–344.

[10] F. Nowak, I. Besenfelder, W. Karl, M. Schmidtobreick, and V. Heuveline, "A data-driven approach for executing the cg method on reconfigurable high-performance systems," in *Architecture of Computing Systems – ARCS 2013*, ser. Lecture Notes in Computer Science, vol. 7767. Springer Berlin Heidelberg, 2013, pp. 171–182.

[11] F. Nowak and R. Buchty, "A tightly coupled accelerator infrastructure for exact arithmetics," in *Architecture of Computing Systems – ARCS '10*, ser. LNCS, vol. 5974. Springer, February 2010, pp. 222–233.

[12] F. Nowak, R. Buchty, D. Kramer, and W. Karl, "Exploiting the htx-board as a coprocessor for exact arithmetics," in *Proceedings of the First International Workshop on HyperTransport Research and Applications (WHTRA2009)*, February 2009, pp. 20–29.

[13] F. Philipp and M. Glesner, "(GECO)^2: A graphical tool for the generation of configuration bitstreams for a smart sensor interface based on a Coarse-Grained Dynamically Reconfigurable Architecture," in *22nd International Conference on Field Programmable Logic and Applications*, ser. FPL '12, 2012, pp. 679–682.

[14] D. Singh, "ImplementingFPGA Design with the OpenCL Standard," Altera Corporation, White Paper, November 2013. [Online]. Available: http://www.altera.com/literature/wp/wp-01173-opencl.pdf

[15] R. Strzodka, "Hardware Efficient PDE Solvers in Quantized Image Processing," Dissertation, University Duisburg, 2004.

[16] W. Vanderbauwhede, S. Chalamalasetti, S. Purohit, and M. Margala, "A Few Lines of Code, Thousands of Cores: High-level FPGA Programming using Vector Processor Networks," in *Int. Conf. on High Performance Computing and Simulation*. IEEE, July 2011, pp. 561–567.

[17] R. P. Wilson, R. S. French, C. S. Wilson, S. P. Amarasinghe, J. M. Anderson, S. W. K. Tjiang, S.-W. Liao, C.-W. Tseng, M. W. Hall, M. S. Lam, and J. L. Hennessy, "Suif: an infrastructure for research on parallelizing and optimizing compilers," *ACM SIGPLAN Notices*, vol. 29, no. 12, pp. 31–37, 1994.

[18] Xilinx, "9 Reasons Why The Vivado Design Suite Accelerates Design Productivity." [Online]. Available: www.xilinx.com/publications/prod_mktg/vivado/Vivado_9_Reasons_Backgrounder.pdf

Experimental Generation of Configurable Circuits for Rotationally Symmetric Functions

Andreas C. Doering
IBM Research – Zürich
Säumerstrasse 4
8803 Rüschlikon/Switzerland
ado@zurich.ibm.com

***Abstract*—With increasing one-time costs for the production of integrated circuits, the drive to integration of configurable circuits together with standard processor cores and interface will increase. So far, either established FPGA fabrics (e.g. Xilinx ZYNQ family) have been used or the configurable units were custom designed for a very specific function (e.g. PowerEN EFSM – TBD). It is therefore of interest to investigate the structures and algorithms for configurable circuits for a well-defined set of functions. As a first step, this paper investigates the class of functions which are invariant under cyclic shifts of their input vectors.**

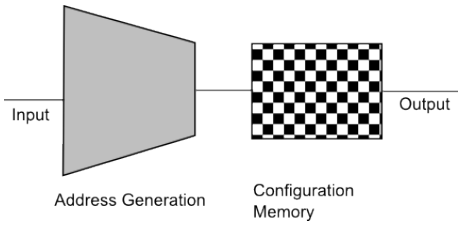


Fig. 1. Configurable circuit for target function set defined by an input property

I. Introduction

Configurable circuits, such as Field Programmable Gate Arrays (FPGAs) or Programmable Logic Devices (PLDs), have been developed with a pragmatic approach: benchmarks are used to evaluate whether a given architecture can implement the functions of the desired domain with acceptable efficiency. Some work [1] has been done to adapt circuits to a given set of target functions, but by far not systematically. With the current high integration of several processor cores and specialized processing cores, such as Digital Signal Processing, or Graphics Processing Units, configurable logic blocks on a System-On-Chip device will have very specific application field[2]. In the same way as application-specific processors have become established building blocks, application-specific configurable logic is interesting from a scientific and a commercial point of view. For non-configurable circuits, trade-off strategies between speed and cost are well understood, but the field of configurable circuits has additional parameters such as size of configuration data, speed of reconfiguration, or coverage of the target function space.

One approach to gaining a deeper understanding is the design of configurable circuits for a given application domain, e.g., cryptographic functions, even if that field can only be vaguely defined. This paper follows the opposite approach: a mathematically well understood set of functions is chosen and their configurable logic is investigated. This has the advantage that the exact size of the set of target functions is known. One of the simplest cases for a special class of functions is obtained by restricting the input: A single input set of binary vectors of length n, $\{(0,0,\ldots,0),\ldots,(1,1,\ldots,1)\}$ is partitioned into sets. Only those functions are considered that have the same input value on the partition sets. Formally, $\{f : a \sim b \Rightarrow f(a) = f(b)\}$. To obtain interesting results the equivalence relation $\sim$ is chosen such that the set of equivalence classes is considerably smaller than the original input set, but not too small. This definition of a function class could be thought of as an input property. Accordingly, the resulting circuit consists of a first stage that compresses the input space into a continuous address range and a look-up table (Figure 1). Therefore, and not surprisingly, the properties of the input set and the set of function values are independent: the compression circuit that assigns an address to each equivalence class of input values could be the same for a small or a large output set. For a small output set (e.g. one bit), there might however be a tradeoff by using a less complex compression circuit that maps some equivalent input values to different addresses. The configuration memory would be larger than the minimum. The functions could be considered filters of the input.

To investigate circuits, a cost measure is needed. To handle the complex functions used in this paper, a synthesis approach is used. The sub-circuits used to build the address generation circuit, such as pairwise-AND followed by a population count, were generated by a C program in verilog source and then synthesized. The un-routed netlist after driver strength tuning is used because placement and wiring are better done for the combined circuit. The open-source tool chain Qflow (www.opencircuitdesign.com) combines various academic tools and an open gate library (for 0.35μm). Cost is measured relative to the size of the smallest inverter. A perl script was developed that sums the sizes of all cells. In this way the methods and results of this paper can easily be reproduced in contrast to using a potentially more optimized synthesis tool and cell library.

The algorithm presented has a high degree of parallelism on several levels, including batch parallelism to run the verilog circuit generator, synthesis, and cost analysis script for a considerable number of basic rotationally symmetric functions.

The paper is organized as follows:

- Section II contains the terminology, and an introduc-

tion of rotationally symmetric functions.

- Section III describes the circuit generation algorithm.
- Section IV lists the cost of basic rotationally symmetric functions for various input sizes, one detailed study for one input size, run-time
- Section V discusses alternative approaches.
- The paper concludes with an outlook on further work.

II. Rotationally Symmetric Functions

In the following, the Boolean values true and false are interpreted as natural numbers 1 and 0. Hence, the population count function of a vector of Booleans, which gives the number of elements with value true, can be written as

$$\mathrm{pc}(x_{n-1},\ldots,x_0)=\sum_{i=0}^{n-1}x_i.$$

The length n of the input or intermediate vectors is typically omitted. Of course, cost, speed, algorithm complexity, etc. are considered in relation to n.

A fully symmetric function f is invariant under any permutation of its inputs. The population count (pc) function covers all fully symmetric functions, i.e., any given symmetric function can be represented as a concatenation of pc and a third function: $f(x)=g(\mathrm{pc}(x))$, as each input could be sorted without changing the function f.

A rotationally symmetric function is invariant only under rotations (cyclic shifts) of its input vector:

$$\forall(x_0,\ldots,x_{n-1}):f(x_0,\ldots,x_{n-1})=f(x_{n-1},x_0,\ldots,x_{n-2})$$

It is sufficient to require only shift by one position because the equivalence under wider shifts follows by several applications of the rule. There is an entire mathematical discipline that covers functions under invariants, called invariance theory. As this field is quite mature, I refer the interested reader to a text book [3].

In Table I several rotationally symmetric functions are listed as examples, but also because of their relevance later in this paper. For clarity, the following function names are used: $r_i(x_0,\ldots,x_{n-1})=(x_i,\ldots,x_{n-1},x_0,\ldots x_{i-1})$ denotes the rotation by i positions, $m(x_0,\ldots,x_{n-1})=(x_{n-1},\ldots x_0)$ is the reflection ("mirror image") of the given vector, and $z(\mathbf{x})=x_0\&x_1\&\ldots\&x_{n-1}$ is the AND-combination of the vector elements.

As can be seen all functions listed have polynomial cost. By the rotational symmetry, the equivalence classes contain at most n elements. Therefore, approximately m^{2^n}/n different functions exist (where m is the size of domain). Hence, almost all rotationally symmetric functions have exponential cost. This applies in particular for the configuration memory in Figure 1.

The equivalence problem for vectors under rotational symmetry is known in combinatorics as *0-1-colored necklace*.

The mirrored vector $m(x)$ is in some cases equivalent under rotational symmetry with x, for instance if $\mathrm{pc}(x)\in\{0,1,2,n-2,n-1,n\}$. Whether this is the case is is indicated by "Invariance under Reflection". It is interesting because it complements the counted product term functions. For those input vectors which are not invariant under reflection it is desirable to distinguish between the two classes. The function Orientation in the table does this, so at a high cost. For odd length I have found a better function.

The counted product term functions represent a large set of functions, which contains both population count and the canonical functions. For example, if the product term is $x_0\&\bar{x_1}$, the corresponding counted product terms function determines the number of sequences of ones in the input vector. Therefore, the counted product term for $\bar{x_0}\&x_1$ is the same function. The parameter $\mathbf{c}$ gives the variables in the product term, $\mathbf{b}$ marks those variables which are inverted. If all input bits are included in the product term, the canonical functions result. Again, there are redundancies, if rotating both $\mathbf{a}$ and $\mathbf{c}$ yields the same function. For each input equivalence class exists a canonical function that is one (or n) only for input values from this class. Consequently, the set of canonical functions alone is sufficient to identify any input class, and thus the set of counted product term functions even more so.

Symmetric functions can be constructed from a given arbitrary function f by creating all rotated copies $f\circ r_i$ and combining them, for instance, by using parity, AND, OR, or any other (fully) symmetric function. The counted product terms function in Table I belong to this class. This method can be used to create a large number of rotationally symmetric functions with low (linear or polynomial) cost.

A function partitions its input set, where each partition belongs to those input values that yield the same function result. The same applies to a set of functions: $P(\{f_i\},(X))=\{\{x:\forall i f_i(x)=y_i\},\forall i:\exists x\in\mathbf{X},y_i=f_i(x)\}$. On each partition, all functions are constant. An address generation function results in a partition that corresponds to the given equivalence relation $\sim$; in the case considered here equivalence is rotational symmetry.

III. Circuit-Generation Algorithm

The algorithm for generating the addressing circuit for the configurable rotationally symmetric circuit is an A*-search on a directed, levelled (and hence acyclic) graph. Each level corresponds to a basic function (for instance the product-term-based functions of Table I). At each node it is decided whether the function of this level will be included into the resulting address generation. Hence, each node corresponds to a set of functions and thus a partitioning of the input set. In the general case, in which all rotationally symmetric functions should be implemented by the circuit constructed, the root corresponds to a partition with a single set, the input set. However, the algorithm can also be applied to special cases in which not all functions are used by selecting a different partition (e.g. a partition which consists of a true subset of all equivalence classes) at the root. A leaf is reached if the partition corresponds to the rotational symmetry. In my implementation, the cost of a node is simply the sum of the costs of the functions selected. This is an overestimation because when combining several functions, part of their additional information can be redundant.

TABLE I. TABLE OF TYPICAL ROTATIONALLY FUNCTIONS

Function	Definition	Cost	Logic Depth
Population Count	$\mathrm{pc}(\mathbf{x}) = \sum x_i$	$O(n \log n)$	$\log^2 n$
Counted Product Terms	$\mathrm{pc}(s_i), s_i = z((r_i(\mathbf{x})\mathrm{XOR}\mathbf{a})\&\mathbf{c})$	$O(n \log n + n(\mathrm{pc}(\mathbf{a})))$	$(\log^2 n) + \log \mathrm{pc}(\mathbf{a})$
Normalization	$k(\mathbf{x}) = \min(\{r_i(\mathbf{x})), i \in \{0, \ldots, n-1\}\})$	$O(n^2)$	$\log^2 n$
Invariance under reflection	$k(\mathbf{x}) = k(m(\mathbf{x}))$	$O(n^2)$	$\log^2 n$
Orientation I	$k(\mathbf{x}) < k(m(\mathbf{x}))$	$O(n^2)$	$\log^2 n$
Canonical	$k(\mathbf{x}) = \mathbf{c}$	$O(n^2)$	$\log n$

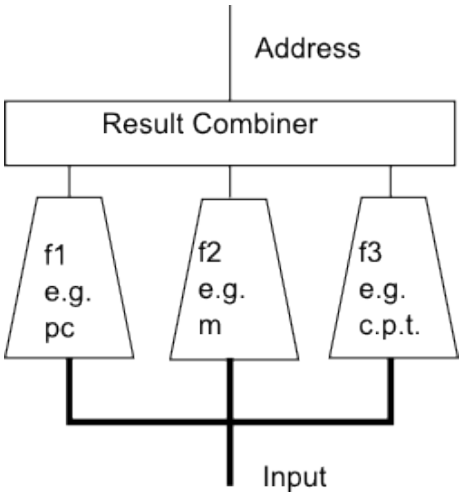


Fig. 2. Concept of address-generation circuit by combining several functions of the target set

The results of selected functions are combined in a final step, see Figure 2. This combination circuit is not discussed in this paper.

The A*-search also needs an estimation of the remaining cost. Of course, the sizes of the partition sets are one indicator how many more functions will be needed. In the ideal case, the functions at the lower levels will divide each partition set into nearly equally large sub-sets, so that the size of the largest set of the partition of the current node could be used. Its logarithm based on the result set size of the next functions is the lower bound of the number of remaining levels.

As an example, consider $n = 7$. Table II shows an example path through the search graph, adding functions with each row. Only the normalized result of each class is listed. Since 7 is prime, there are $(2^7-2)/7+2$ equivalence classes. Two classes with one element only (1111111 and 0000000), and 18 classes of 7 elements each.

Note that for larger n there are many more classes that are not invariant under reflection. These larger cases would have been impractical for a table, though. The first two functions are much stronger then.

The processing at each node requires the application of the new function to all non-singular partitions. This step has a high level of parallelism. At the high levels of the search graph, 2^n independent function applications have to be computed and the results collected in sets, e.g. using a hash table. The partitions can be stored in a distributed way as only their size is needed. The communication effort between the processors is low, as it scales with the number of function results of the new function.

TABLE III. SIZES AND RUN-TIME FOR SELECTED VALUES OF n

n	Cost	Functions used	runtime(s)	Nodes visited
6	198	4	5	5205
7	305	4	0.1	99
8	596	6	1.1	559
9	845	7	48	20000
10	1153	9	65	16645
11	2960	6	98	20000
12	3982	11	111	20000

A prerequisite for the search to converge is that the cost of filters has to grow eventually with increasing levels. Otherwise the search could follow the path that does not add new functions infinitely, in particular if the functions are worse than the heuristic estimates. Functions that are considered particularly useful can be put at high levels even if they are costly.

IV. RESULTS

The algorithm has been implemented in C++ using the Xerces-C and the Boost graph libraries. The algorithm is configured using a file in XML format. In addition to options, such as the length of the input vector and required output files, also a number of steps starting at the root can be controlled to be pre-computed before the search is started. Table III lists the runtime and the resulting size of functions for a range of input length values.

The search algorithm terminates when either no nodes for investigation are available anymore, three solutions were found or 20000 nodes visited. Of course, these quantitative limits are configurable. Once the first solution is found the algorithm discards all open nodes that have a higher cost than the previously found nodes. A further condition would be desirable, yet is not yet implemented: the condition that the search should also stop at levels below which only functions appear that are more expensive than the already found solution. As mentioned the cost has to increase eventually to enforce convergence. This is difficult to implement when the assignment of functions to tree levels is configurable. This sequence was for the results in Table III: Reflect, Orientation, Population Count, AND-2 (counted product terms with two positive literals), all other product terms. None of the solutions used the expensive Orientation function, most used population count.

The exceptional low runtime for the case $n = 7$ is due to the fact that the rotational symmetry has a simpler structure for prime number vector length cases. Why $n = 11$ was not simpler, I can't say. Maybe a different selection or sequence

TABLE II. REFINED PARTITIONING FOR $n = 7$, OR. STANDS FOR ORIENTATION I, IUR IS INVARIANCE UNDER REFLECTION, AND C.P.T. IS COUNTED PRODUCT TERM

Function	Partition
Start	$\{0, 1, 3, 5, 7, 9, 11, 13, 15, 19, 21, 23, 27, 29, 31, 43, 47, 55, 63, 127\}$
IuR	$\{0, 1, 3, 5, 7, 9, 15, 19, 21, 27, 31, 43, 47, 55, 63, 127\}, \{11, 13, 23, 29\}$
Or.	$\{0, 1, 3, 5, 7, 9, 15, 19, 21, 27, 31, 43, 47, 55, 63, 127\}, \{11, 23\}\{13, 29\}$
pc	$\{\{1\}, \{3, 5, 9\}, \{7, 19, 21\}, \{15, 27, 43\}, \{31, 47, 55\}, \{63\}, \{127\}, \{11\}, \{23\}, \{13\}, \{29\}\}$
C.P.T., $\mathbf{a} = 0000011$	$\{\{1\}, \{3\}, \{5, 9\}, \{7\}, \{19\}, \{21\}, \{15\}, \{27\}\{43\}, \{31\}, \{47, 55\}, \{63\}\{127\}\{11\}\{23\}\{13\}\{29\}\}$
C.P.T., $\mathbf{a} = 0000101$	$\{\{1\}, \{3\}, \{5\}, \{9\}, \{7\}, \{19\}, \{21\}, \{15\}, \{27\}, \{43\}, \{31\}, \{47\}, \{55\}, \{63\}, \{127\}, \{11\}, \{23\}, \{13\}, \{29\}\}$

of the functions would have revealed a better solution. The reason for the role of prime numbers is as follows: if the vector length n has a divider $1 < d < n$ then there are input pattern that already repeat after a rotation of d digits. Hence, the equivalence class for this pattern has only $n/$ elements, and therefore in total there are more equivalence classes than for comparable prime number length cases.

Figure 3 shows the search tree for $n = 6$. The best solution (with cost 357) is the one at the bottom, at level 13, which is found last.

For reference, Figure 4 shows the cost after synthesis of various functions.

V. ALTERNATIVE APPROACHES

It is not evident that the proposed method will find solutions that are at least nearly optimal. Therefore, it is desirable to investigate alternative approaches. Search methods such as genetic algorithms could be used to search directly for an address-generation function, or the address generation function is constructed bit-wise.

Another method is to apply functional decomposition methods to the problem. To do this the entire configurable circuit is given as one complex function; the configuration pins are also treated as inputs. This function is then analyzed by decomposition methods such as the one described in [4]. This has the advantage that memory addressing and access multiplexing are not enforced upfront, as in the approach presented here. The address to the configuration memory in Figure 1 is assumed as a binary address in the algorithm of Section III.

Finally, one could first establish an upper bound for the circuit size and then represent all possible circuits a closed form, so that either Binary Decision Diagrams or a SAT solver can be used to find optimal solutions.

VI. OUTLOOK

Although cost of the address generation function for the implementation of configurable rotationally symmetric functions can be determined with the algorithm presented, a comparison with existing configurable architectures is not yet possible, because this requires the determination of the minimum number of resources (look-up tables for FPGAs or product terms for PLDs) for the set of target functions. A first indication could be obtained by applying the method in [5] to randomly selected functions. As many functions will have a high complexity, a randomly selected sample should come close to the most

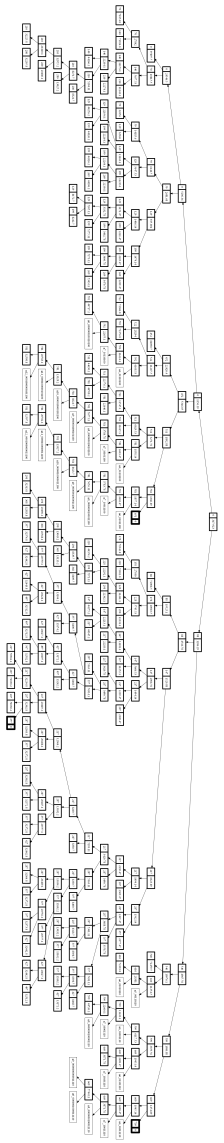


Fig. 3. Search tree for $n = 6$

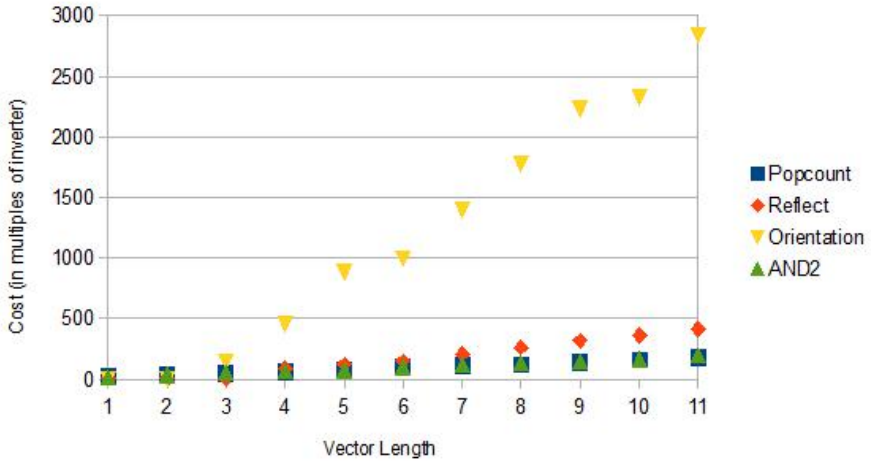


Fig. 4. Cost of various rotationally symmetric functions after synthesis as used as input to the search algorithm

expensive case. This approach could entail of course a high computational effort.

In addition to the creation of a configurable circuit, another important aspect is a mapping algorithm that translates a given function into a configuration of the circuit. Given the output-table-based architecture, this task is trivial for the architecture presented if the symmetry cycle is known. In the definition of rotational symmetry, the cycle was pre-defined as $(0, 1, \ldots, n-1)$. If the circuit presented is integrated in a system that allows arbitrary routing, such as an the routing structure in an FPGA, also functions that have a different symmetry cycle could be implemented in the circuit presented. I have developed an algorithm that can determine whether an arbitrary function has a symmetry, including symmetries consisting of several cycles or cycles covering only a subset of the inputs. This algorithm has been published in [6], to prevent a patent on the algorithm by others, but without any details or explanations. Another algorithm for the same problem is presented in [7], but it entails however a very high computation effort.

When a configurable circuit for a set of functions with a common input property is to be created, the primary need is a method to generate a large number of functions from the target set with low cost.

REFERENCES

[1] M. Holland and S. Hauck, "Automatic creation of domain-specific reconfigurable cplds for soc," in *Proc. FPL*, 2005, pp. 95–100.

[2] M. B. Taylor, "A landscape of the new dark silicon design regime," *IEEE Micro*, vol. 33, no. 5, pp. 8–19, 2013.

[3] B. Sturmfels, *Algorithms in invariant theory.*, ser. Texts and monographs in symbolic computation. Springer, 1993.

[4] C. Scholl, *Functional Decomposition with Application to FPGA Synthesis*. Kluwer, 2001.

[5] K. Atasu, T. Todman, O. Mencer, and W. Luk, "Optimal Implementation of Combinational Logic on Lookup Tables," in *The Fourth Conference on Ph.D. Research in Microelectronics and Electronics (PRIME'08), Istanbul, Turkey*, May 2008.

[6] A. Doering, "Determination of a generating cycle for a rotationally symmetric function," Ip.com, April 2009, iP.com Disclosure Number: IPCOM000182091D.

[7] P. Jasionowski, "Matrix characterization of symmetry groups of boolean functions," *Journal Algebra and Discrete Mathematics*, vol. 14, no. 1, pp. 71–83, 2012. [Online]. Available: http://adm.lnpu.edu.ua/downloads/issues/2012/N3/adm-n3(2012)-6.pdf

Evaluating the Energy Efficiency of Reconfigurable Computing Toward Heterogeneous Multi-Core Computing

Fabian Nowak
Karlsruhe Institute of Technology
Chair for Computer Architecture and Parallel Processing
76128 Karlsruhe, Germany
Email: nowak@kit.edu

***Abstract*—Future exascale systems need to have a much better performance-to-power ratio than today's systems. Accelerators are a promising approach to pave this path by more energy-efficient computing. We show some early results of our investigations toward energy efficiency of reconfigurable and heterogeneous computing against multi-core processors for special applications. The results are supported by a general framework and toolchain for early evaluation of potential benefits of reconfigurable hardware. As a result, heterogeneous systems based on reconfigurable hardware, efficient data exchange mechanisms, data-driven and component-based programming, and task-parallel execution can help achieve power-efficient exascale systems in future.**

I. Introduction

Multi-core microprocessors nowadays feature a thermal design power (TDP) of more than 100 Watt. Not only is energy consumption high, but it is also hard to cool down the components. New solutions are required to lower average and maximum energy consumption and to also distribute the heat spots along the processor chip. Hardware accelerators can fulfill a lot of tasks not only faster, but even much more energy-efficiently, i.e., consuming less energy in the same time or consuming more energy while also being unproportionally faster, or more formally:

$$energy\ efficiency := \frac{ratio_{speedup}}{ratio_{energy}}$$

. It is therefore of paramount importance to evaluate other resources as alternatives to microprocessor cores for energy-efficient execution in order to someday yield exascale computing systems [6]. In addition, other means than data-parallel programming and execution must be found, evaluated and propagated to make more use of the number of cores already present in nowadays processors. For example, OpenMP 3.0 already supports task-parallel execution. Ideally, such other means will allow seamless migration of program parts from microprocessor cores to accelerators and vice versa.

Apart from energy consumption, another issue with today's microprocessors is that many parts such as floating point units, multimedia or SIMD streaming extensions are unused [13]. Reconfigurable hardware instead of such hard-wired special units allows to dynamically "load" required or useful hardware so that chip area will always be used. Field programmable gate arrays (FPGAs) contain lots of reconfigurable hardware and can therefore serve as evaluation candidates. t

In this paper, we employ the Convey HC-1, a heterogeneous system equipped with four user-programmable FPGAs, for our investigations toward energy-efficient computing. The FPGAs are connected to the host microprocessor via the Front-Side Bus (FSB), thereby posing a tightly coupled heterogeneous system that can serve as a basis and evaluation platform for future architectures of multi- or many-core processors. We also employ a 2-socket Intel Westmere X5670 system with six cores each and HyperThreading as homogeneous multi-core system, providing up to 24 hardware threads in total, to compare coprocessor-extended heterogeneous computing against.

After giving an overview of related work in Section II, we show in Section III three implementations: a kernel for the bioinformatics application HHblits in reconfigurable hardware based on previous work [2], [8] and its extension toward heterogeneous computing; a micro-programmed, data-driven, task-parallel, component-based and domain-adaptive framework in software; and the implementation of said framework for leveraging FPGAs based on previous work [7]. Our evaluation in Section IV shows that reconfigurable computing can be up to 24× more energy-efficient than legacy computing on microprocessors. We draw the conclusions in Section V, giving some final remarks on possible heterogeneous microprocessors to come in the future.

II. Related Work

A solution to low usage of many parts of the microprocessor [13] is to distribute work onto all available cores to execute in a data-parallel fashion. OpenMP and Intel Array Building Blocks yield high processor utilization and high speedups this way, with the latter also exploiting SIMD streaming extensions (SSE) and thereby even more of a chip's resources. When memory bandwidth cannot provide enough data to the processing cores for the application to scale only by means of exploiting data-level parallelism, an obvious solution is to bring in task parallelism, i.e., executing different functions of an application concurrently, hopefully reusing already loaded or cached data. Also, the nodes of an application's directed acyclic graph representation can execute independently. Intel's Cilk Plus [12], OpenMP 3.0 and StarSS [10] are promising

programming models to exploit task parallelism in multi- and manycore systems.

Another solution to tackle dark silicon issues [13] is to provide heterogeneous cores. Basically, with MultiAmdahl the allocation problem of which and how many resources should be integrated on a chip, given a specific target application, has been solved [14]. Now what remains to be investigated and qualified are programmability and energy efficiency of reconfigurable hardware-based accelerators that represent a potential "specialized horseman".

Comparing the FPGA-based heterogeneous computing system Convey HC-1 against a GPU-based system, Bakos finds the HC-1 $5\times$ less energy-efficient [1]. However, we expect more benefit from heterogeneous computing with reconfigurable hardware. Therefore, we evaluate our bioinformatics FPGA-based accelerator for ungapped score calculations as used by HHblits [11]. We also shortly present our task-based programming model for both homogeneous and heterogeneous resources and upon this basis, we evaluate energy efficiency of task-based programming and computing in heterogeneous environments.

III. Implementations

We present three different implementations to evaluate the fitness of existing homogeneous and heterogeneous computing systems in regard to required energy efficiency for exascale computing.

A. Accelerating Ungapped Score Calculations for Bioinformatics Application

HHblits [11] is a bioinformatics application to find homologous sequences among a query protein string and a database of reference protein strings. HHblits employs prefiltering of the millions of database entries by means of a striped Smith-Waterman implementation [3], which has been implemented on FPGA recently [2], [8].

HHblits works as depicted in Figure 1: For the query protein sequence, at first a Hidden Markov Model (HMM) is created. Then, a profile is created upon the query HMM. The profile is used to prefilter suitable entries from the large protein database. This is done by first calculating a so-called *ungapped score* without considering any gaps between the query profile and the database reference sequence, i.e, the query and the reference are quickly evaluated with regard to contiguously matching parts. Those sequences passing this ungapped-score prefilter are roughly checked whether they would also deliver a suitable score when accepting gaps between subsequences. The large number of millions of database entries is thereby decimated to only some ten thousands. These "few" remaining protein HMMs are regularly compared against the query HMM. To obtain more suitable candidates, additional runs can be used to find more results by "widening" the query so that more reference sequences can match the query. If another run is intended or required, then an update of the initial HMM by the information from the accepted HMMs is needed and accordingly performed by HHblits. The most time-consuming part is the prefiltering step, with the first step (ungapped score calculation) accounting for 1601.9 seconds of the total application time of 1626.2 seconds when not

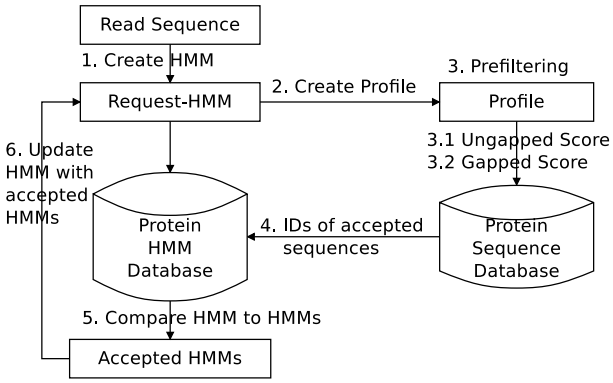


Fig. 1. Comparing a query sequence against a protein sequence database within HHblits.

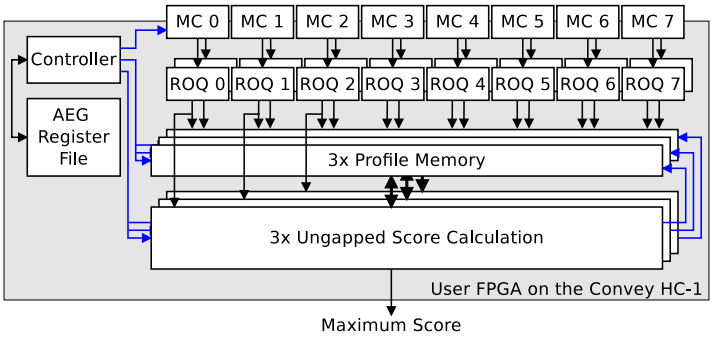


Fig. 2. Implementation of the first prefiltering step of HHblits on the four FPGAs on the Convey HC-1. (MC – Memory Controller, ROQ – Read-Order Queue, AEG registers – Application Engine General registers).

employing Farrar's efficient striped SSE implementation for the prefiltering step when performing only a single run. When prefiltering is accelerated via Farrar's SSE implementation, then prefiltering takes roughly 67 % of the total application time of 74.1 seconds for one run. If more runs are performed, the proportion of the ungapped score calculations to overall application execution time decreases.

Key to successful acceleration is enhancing data reuse. So in contrast to the SSE execution, our implementation [2], [8], shown in Figure 2, of the first prefiltering step for calculating a locally ungapped score first loads the entire profile in parallel over all 16 memoy controllers (MCs) and then streams the query sequence over one MC per ungapped score calculation unit (UCU) that can then heavily reuse data from different parts of the profile. The profile is shared by all instantiated UCUs. Apart from data reuse, exploiting massive parallelism is required to successfully speedup an application despite the low processing frequency of FPGAs. We do so by employing the four FPGAs of the Convey HC-1, with each FPGA instantiating three UCUs for calculating three different ungapped scores in parallel. These calculations are made up of 16 parallel, 8-Byte-processing units; and the maximum is calculated in a reduction tree over the 128 calculated individual scores and is implemented as a pipeline. In total, we exploit $4{\cdot}3{\cdot}16{\cdot}8 = 1536$ fold parallelism and can output a new maximum score per FPGA every cycle, while the SSE version suffers from the non-parallel maximum calculations that introduce undesired additional delay.

The two cores on the host processor of the HC-1 can also perform useful work during coprocessor execution: the second prefiltering step depends only on the individual results of previous single ungapped score calculations and can therefore be carried out in an overlapped, task-parallel fashion.

B. Task-Parallel, Data-Driven and Component-Based Programming and Execution (TDDcomp)

As second application domain to be analyzed toward performance and efficiency of homogeneous and heterogeneous computing systems we investigate numerical applications. As can be seen from Figure 3 of the conjugate gradient method (CG method), not only are some functions completely independent from others, but many functions can also be executed in an overlapped fashion, using already calculated vector entries to compute the next depending functions. This is interesting and helpful for two reasons: first, the available bandwidth to/from external memories is too low to provide enough data to the processing cores as soon as data become too large to fit into the caches so that some cores are only waiting for data. Second, newly calculated data that would normally be replaced in the caches is reused now.

We implemented a set of linear algebra routines in a data-driven fashion. The functions are executed as separate POSIX threads and consume data from matrices or vectors and exchange progress information in order to not use yet invalid data. From a performance point of view, it proved to be best to split matrices into ten blocks, so that communication would not occur on a per-datum or per-line basis. Implemented and evaluated mechanisms ensure that the function threads do not consume valuable processing time as long as the required input data are not available. When processing data blocks, OpenMP is employed to exploit data parallelism in addition. From an application programmer's perspective, programming resembles that of sequential programming. The programmer has to care that as much task parallelism can be exploited as possible. During development, our investigations revealed that matrix data and structures should be allocated rather sooner than later in order to leverage more task parallelism after the allocation phase; and the structures should be freed as late as possible in order to not pollute the caches during ongoing more useful calculations. As it does not make sense to poll continuously for new data, we evaluated sleeping against waiting on semaphores. Upon the results gained, we decided to employ semaphores. The final implementation and configuration of the routines allow processing of large vectors with competitive speed to data-parallel execution for the CG method.

C. Implementation of a Micro-Programmable TDDcomp Framework on the Convey HC-1

When employing FPGAs to accelerate applications partially or even entirely, one of the most crucial problems is data transfer due to limited off-chip bandwidth and lack of hardware caches. Therefore, we implemented a streaming-oriented, component-based architecture framework similar to the one proposed above in Subsection III-B as depicted in Fig. 4. To exchange data between components and external memories in an efficient manner, a central FIFO buffer set connects the components much like a crossbar. Data flow is

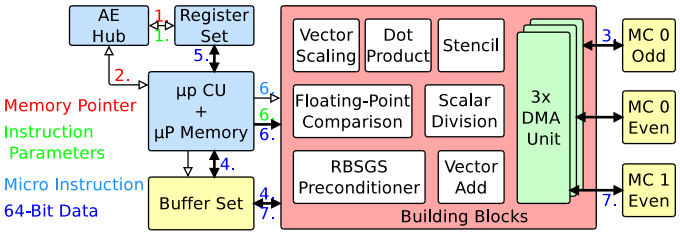


Fig. 4. Implementation of the architecture framework on the Convey HC-1.

controlled via a user-supplied micro program. This framework allows application programmers to provide and exploit custom, application-targeted accelerators in reconfigurable hardware by developing micro programs only instead of describing hardware at low level and instead of employing high-level language converters. For example, the CG method can be accelerated up to ten times in comparison to the Convey HC-1's host processor and the framework even achieves comparable performance to 24 hardware threads on a two-socket Intel Westmere X5670 system [7], although employing only one out of the four available FPGAs.

With special units such as an enhanced-precision dot product implementation, even more speedup can be gained because the algorithm can converge in only half the number of iterations. This architecture framework is hence also a possible approach of how to interconnect multiple cores on a manycore chip and how to have them communicate efficiently via buffer sets in a data flow style. Moreover, the special units represent the case of exploiting reconfigurable logic for custom accelerators, while the numeric functions might also be executed on standard microprocessor cores.

IV. Analysis and Evaluation

We aim at analyzing the fitness of heterogeneous, FPGA-based computing systems as a possible approach for future exascale computing where much more energy-efficient execution than on today's regular, multi-core processor-based computing systems is required. In future, manycore chips might want to integrate reconfigurable logic for faster and less energy-consuming, i.e, more energy-efficient, computing. An early example is the Intel Atom E6x5C [4] with an Altera FPGA connected to the low power core over PCI Express. This way, the dark logic issues [13] might be solvable when integrating special instructions, operations or functions. Also will heat dissipation be less problematic then. We do hence study execution time and energy consumption of the implementations presented in Sect. III on the Convey HC-1 and a two-socket Intel Westmere system. Table I summarizes the maximum power consumption of these systems. Although many processors can be dimmed down to a fraction of the maximum frequency, this is not much different from using FPGAs that are clocked with only 100 MHz to 300 MHz, except that FPGAs can implement special operations at bit-level or with smart data structures so that they also reduce execution time in addition to having a lower power consumption footprint. Some processors can also be boosted to a higher frequency, but this is possible only for a very short time due to heating problems and going rather into the opposite direction than the approach favored in this paper.

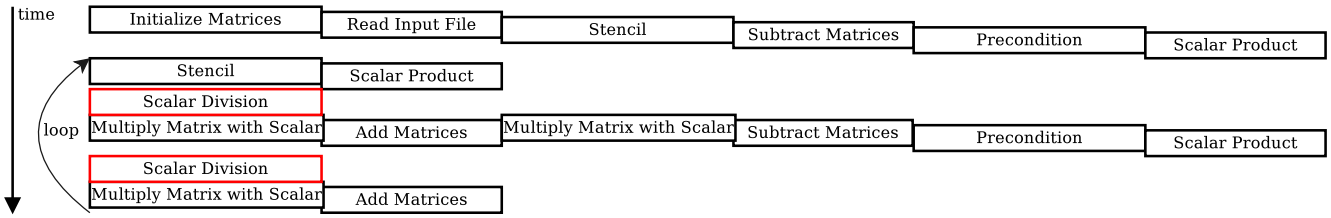


Fig. 3. Graph for the Conjugate Gradient Method indicating exploitable task parallelism. Data flows from left to right, while time advances vertically. Many tasks can execute in an overlapped fashion as they only depend on few previously calculated data.

TABLE I. THERMAL DESIGN POWER (TDP) AND MAXIMUM POWER CONSUMPTION OF THE USED SYSTEMS.

System setup	Intel Xeon 5138 @Convey HC-1	Xilinx Virtex-5 LX330 @Convey HC-1	Convey HC-1	Intel Xeon X5670	2-socket X5670
TDP / maximum	35 W	35.8 W	178.2 W	95 W	190 W

A. Energy Efficiency of HHblits Implementations

The performance of HHblits largely depends on the performance of the ungapped score calculation kernel (60 %-70 % of total execution time [2]). We study the individual aspects of coprocessor-supported execution and task-parallel execution and finally merge these.

1) Ungapped Score Calculation on the FPGA-based Coprocessor: As can be seen from Table II, employing the FPGAs dramatically increases power consumption because as of now, a hardware thread is required to control execution on the FPGAs (fifth row). FPGA power consumption is determined via Xilinx Power Analyzer (XPA), while CPU power consumption is taken from the specified thermal design power (TDP). Also, for the remaining parts of HHblits, power and energy consumption of the standard processor must be accounted for. However, due to shortened overall execution time and because the coprocessor runs only for the ungapped score calculations, but not for the remainder of the application, the total energy efficiency of the heterogeneous system shall be better, as will be seen later.

2) Overlapped Execution of HHblits: We already sketched the approach of overlapping different subtasks of HHblits. As multi-core processors may easily run into the memory wall, it must be investigated whether task-parallelism allows to better exploit the available processing resources so that memory accesses do not become the bottleneck. Figure 5 shows that although not necessarily providing real benefit over data-parallel execution, it is possible to entirely hide the computations of swStripedByte behind the ungapped-score calculation by executing in a task-parallel, component-based manner. The overhead imposed by synchronization must be regarded the culprit for the low advantages of the task-parallel approach for our coprocessor-accelerated version of HHblits. As consequence, more data must be transferred so that memory bandwidth is stressed more, or a different communication model must be used for synchronization.

3) HHblits on the Heterogeneous System: Applying the task-parallel model while also using the coprocessor allows to easily synchronize the ungapped score calculations more tightly with the subsequent swStripedByte because the coprocessor is invoked for a set of sequences only, and swStripedByte can process the previous sequence set then. Again, Table III shows that energy consumption is higher when also processing HHblits on the coprocessor, but due to achieved speedup, the heterogeneous systems performs more than 1.5× more energy-efficiently.

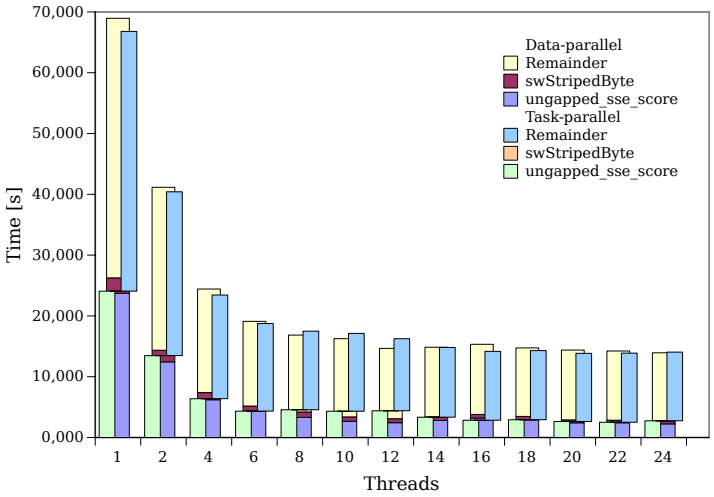


Fig. 5. Data-parallel execution of HHblits on Homogeneous System with two X5670 processors against task-parallel execution, which hides swStripedByte nearly entirely behind ungapped-score calculation and thereby is faster in most cases.

B. Energy Efficiency of Conjugate Gradient Method Implementations

To finally support our thesis that heterogeneous computing based on reconfigurable hardware can provide benefit over computing on homogeneous resources only, we regard energy consumption and efficiency of an FPGA for task-parallel, data-driven computing (TDDcomp) in reconfigurable hardware on the Convey HC-1's coprocessor against a two-socket Intel Xeon X5670 system. A previous implementation [7] of the CG method in the framework serves as basis for these investigations. As indicated by Xilinx Power Analyzer, the FPGA implementation of TDDcomp draws 17.439 W of power. Note that the FPGA implementation includes an extended dot product unit in order to draw more benefit from reconfigurable hardware by arbitrary bit-width implementations. This also reflects the case that special operations (→ "specialized horseman" [13]) are tightly integrated into execution of a program on various resources, such as an exact dot product implementation [9] which is approximated here by the extended-precision unit. Table IV lists the execution time, calculated

TABLE II. POWER CONSUMPTION AS USED IN OUR ANALYSIS MODEL, EXECUTION TIME AND ENERGY CONSUMPTION OF ALL KERNEL CALCULATIONS OF HHBLITS ON THE CONVEY HC-1.

System setup	Power consumption [W]	Kernel execution time [s]	Energy consumption [mWh]
Xeon 5138, 1 Thread (TDP)	17.5	1601.9	7787.0
Xeon 5138, 1 Thread, SSE (TDP)	17.5	50.0	243.1
Xeon 5138, 2 Threads, SSE (TDP)	35	25.1	244.0
Xilinx Virtex-5 LX330 (XPA)	18.561	28.6	147.5
Convey HC-1 with Coprocessor (1 Hardware Thread, 4 FPGAs)	91.744	13.1	333.8

TDP – Thermal Design Power, XPA – Xilinx Power Analyzer

TABLE III. EXECUTION TIME, ENERGY CONSUMPTION AND RATIOS OF CPU EXECUTION, COPROCESSOR-EXTENDED EXECUTION AND OVERLAPPED CPU/COPROCESSOR EXECUTION FOR THE ENTIRE APPLICATION HHBLITS.

	Xeon 5138 (1 Thread, SSE)	With Coprocessor (+ 4 FPGAs)	Overlapped
Execution time [s]	74.585	37.465	34.147
Energy consumption [mWh]	362.566	507.048	507.048
Speedup over CPU	(1.0)	1.991	2.184
Relative energy consumption against CPU	(1.0)	1.398	1.398
Relative energy efficiency against CPU	(1.0)	1.424	1.562

energy consumption and energy efficiency ratio of FPGA-based computing against a 24-hardware-thread system. The FPGA implementation of the conjugate gradient method within the micro-programmable TDDcomp is more than 17 times as energy-efficient as execution on a 24-hardware thread Intel Westmere system.

V. CONCLUSIONS AND FUTURE WORK

We showed that reconfigurable computing can be 17× more energy-efficient than computing on general-purpose multi-core processors. To fully leverage the benefits of heterogeneous, accelerator-based systems, both the kernels ported to accelerators in reconfigurable hardware and the application need to take much care of data locality and data reuse. The task-parallel execution of components in reconfigurable hardware and software allows to exploit such data locality and data reuse by means of FIFO buffers and by the concept of streaming. As a result, our investigations show that upcoming multi- and many-core microprocessors should be heterogeneous by nature, for example by instantiating a couple of FPGA-like resources to provide application-required accelerators on demand. Moreover, the task-parallel execution of components in a streaming fashion requires data buffers for linear data access. Such data buffers should make up another part of microprocessors in addition to the regular caches as the buffers will support both kernels in reconfigurable hardware and in software. With such a setup of multiple processing cores, reconfigurable hardware and custom memory structures on chip, we can expect to meet the required power and energy goals of exascale systems. Future work includes seamless provision of the required accelerators based on previous work toward self-management of heterogeneous systems [5].

ACKNOWLEDGMENTS

The author acknowledges the invaluable contributions of Ingo Besenfelder and Michael Bromberger to the implementations on the Convey HC-1 and also the helpful comments from the reviewers.

REFERENCES

[1] J. Bakos, "High-Performance Heterogeneous Computing with the Convey HC-1," *IEEE Computing in Science & Engineering*, vol. 12, no. 6, pp. 80–87, Nov.-Dec. 2010.

[2] M. Bromberger and F. Nowak, "Parallel Prefiltering for Accelerating HHblits on the Convey HC-1," in *Mitteilungen*, vol. 30. Gesellschaft fr Informatik e. V., Parallel-Algorithmen und Rechnerstrukturen, Sept. 2013, pp. 47–57.

[3] M. Farrar, "Striped Smith-Waterman speeds database searches six times over other SIMD implementations," *Journal of Bioinformatics (Oxford University Press)*, vol. 23, no. 2, pp. 156–161, Jan. 2007.

[4] "Intel® Atom™ Processor E6x5C Series," online, Intel, Dec. 2010, Product Preview Datasheet, Revision 001US. [Online]. Available: http://www.intel.com/content/dam/www/public/us/en/documents/datasheets/atom-e6x5c-datasheet.pdf

[5] M. Kicherer, F. Nowak, R. Buchty, and W. Karl, "Seamlessly Portable Applications: Managing the Diversity of Modern Heterogeneous Systems," *ACM Trans. Archit. Code Optim.*, vol. 8, no. 4, pp. 42:1–42:20, Jan. 2012.

[6] N. Leavitt, "Big Iron Moves Toward Exascale Computing," *IEEE Computer*, vol. 11, pp. 14–17, November 2012.

[7] F. Nowak, I. Besenfelder, W. Karl, M. Schmidtobreick, and V. Heuveline, "A Data-driven Approach for Executing the CG Method on Reconfigurable High-Performance Systems," in *Architecture of Computing Systems*, ser. LNCS, vol. 7767. Springer, Jan. 2013, pp. 171–182.

[8] F. Nowak, M. Bromberger, M. Schindewolf, and W. Karl, "Multi-Parallel Prefiltering on the Convey HC-1 for Supporting Homology Detection," in *Proceedings of the 20th European MPI Users' Group Meeting*, ser. EuroMPI '13. ACM, Sept. 2013, pp. 169–174, international Workshop on Parallelism in Bioinformatics (PBio 2013).

[9] F. Nowak, R. Buchty, D. Kramer, and W. Karl, "Exploiting the HTX-Board as a Coprocessor for Exact Arithmetics," in *Proceedings of the First International Workshop on HyperTransport Research and Applications (WHTRA2009)*. Computer Architecture Group, Institute for Computer Engineering (ZITI), University of Heidelberg, Feb. 2009, pp. 20–29. [Online]. Available: http://www.ub.uni-heidelberg.de/archiv/9114

[10] J. Planas, R. M. Badia, E. Ayguadé, and J. Labarta, "Hierarchical Task-Based Programming With StarSs," *International Journal of High Performance Computing Applications, Sage Publications*, vol. 23, pp. 284–299, August 2009.

[11] Remmert, M., Biegert, A., Hauser, A., and Sding, J., "HHblits: Lightning-fast iterative protein sequence searching by HMM-HMM alignment," *Journal of Nature methods (Nature Publishing Group)*, vol. 9, no. 2, pp. 173–175, Feb. 2012.

[12] A. D. Robison, "Composable Parallel Patterns with Intel Cilk Plus," *Computing in Science and Engineering*, vol. 15, no. 2, pp. 66–71, 2013.

TABLE IV. EXECUTION TIME OF CG METHOD ON COPROCESSOR FPGA (XILINX VIRTEX-5 LX330) AGAINST 2-SOCKET INTEL XEON WESTMERE X5670 SYSTEM, AND THE CORRESPONDING ENERGY CONSUMPTION AND RATIO AS DEFINED IN SECT. I.

	512×512	1024×1024	2048×2048	4096×4096
Execution time on 2 X5670	2.405 s	21.242 s	168.618 s	1896.006 s
Execution time 1 FPGA	2.286 s	16.607 s	145.628 s	1196.383 s
Maximum Energy Consumption, CPU	126.938 mWh	1121.100 mWh	8899.266 mWh	100066.993 mWh
Maximum Energy Consumption, 1 FPGA	22.729 mWh	165.144 mWh	1448.186 mWh	11897.358 mWh
Calculated Energy Consumption, 1 FPGA	11.072 mWh	80.446 mWh	705.444 mWh	5795.476 mWh
Ratio	11.5	13.9	12.6	17.3

[13] M. B. Taylor, "A Landscape of the New Dark Silicon Design Regime," *IEEE Micro*, vol. 33, no. 5, pp. 8–19, 2013.

[14] T. Zidenberg, I. Keslassy, and U. Weiser, "Optimal resource allocation with multiamdahl," *Computer*, vol. 46, no. 7, pp. 70–77, 2013.

Workshop Grid4Sys 2013, 27./28.11.2013, Dresden

»Grid-, Cloud- und Big-Data-Technologien für Systementwurf und -analyse«

Aufruf zum Einreichen von Beiträgen

Grid4Sys ist die fünfte Auflage einer Workshop-Reihe, die 2005 unter dem Namen „Grid-Technologie für den Entwurf technischer Systeme“ begonnen wurde. Die Neuauflage unter erweitertem Blickwinkel soll dem Erfahrungsaustausch von Ingenieuren, Naturwissenschaftlern und Informatikern dienen, die Grid-, Cloud- und BigData-Technologien erarbeiten und nutzen, um komplexe Systeme zu entwickeln und zu analysieren.

Den Entwicklern solcher technischen oder nicht-technischen Systeme ermöglichen Cloud- und Grid-basierte Infrastrukturen - jederzeit und von beliebigen Orten aus - einen flexiblen und skalierbaren Zugang zu benötigten Entwurfs- und Analysewerkzeugen, z.B. Simulatoren. Grundlage dafür sind Virtualisierungstechniken zur Ausführung von ins Netzwerk ausgelagerten Software-Werkzeugen und IT-Prozessen. Ein solcher Ansatz überwindet nicht nur Investitionsbarrieren beim Nutzer, sondern ermöglicht insbesondere bei aufwändigen und komplexen Entwurfs- und Analyseaufgaben, bei erheblichen Datenmengen sowie bei umfassenden Parametervariationen den bestehenden Bedarf an Prozessor- und Speicherressourcen praktisch unlimitiert zu befriedigen. Für die Entwicklung innovativer Anwendungen werden solche Infrastrukturen wegen des wachsenden Datenaufkommens, der heterogeneren Datenquellen und der intelligenteren Verarbeitungsstrukturen künftig unverzichtbar werden.

Neben anderen Beiträgen wird der Workshop auch Ergebnisse des vom BMWi in der Förderinitiative ‘Trusted Cloud‘ geförderten Projektes ‘Cloud4E‘ vorstellen, das Grid-, Cloud- und Big-Data-Technologien für Simulationaufgaben untersucht und entwickelt.

Beiträge zu folgenden und weiteren relevanten Themen werden erbeten:

Methoden / Technologien für Grid Computing und Verarbeitung großer Datenmengen in der Cloud

- Middleware für technische und nicht-technische Anwendungen
- Speicher-, Ressourcen-, Workflowmanagement und Virtualisierungstechniken
- Anwendungsspezifische Analyse umfangreicher, heterogener Datenbestände und skalierbare, nutzergerechte Visualisierungstechniken

Sicherheit, Zuverlässigkeit und Interoperabilität für Grid und Cloud Computing

- Vertraulichkeit und Privatsphäre, Angriffssicherheit und Risikomanagement
- Verlässliche Informationsverarbeitung und Verfügbarkeit von Services
- Interoperable und Standard-basierte Lösungen für das Big Data Management

Industrielle und wissenschaftliche Big Data Anwendungen in Grid und Cloud

- Optimierung technischer Systeme
- Fallstudien aus Computational Physics / Photonics / Chemistry / Biology
- Parameter- und Variantenuntersuchungen für komplexe Simulationen, Tests, Verifikationen und GPU-/FPGA-Accelerators
- Plattformen zur Unterstützung industrieller Produktionsprozesse und Demonstration von Produktions-Grids und Private Clouds

Akzeptierte Beiträge werden in einem Tagungsband mit ISBN-Nummer veröffentlicht. Die Teilnahmegebühr beträgt 150,00 EUR bei einer Anmeldung bis einschließlich 01.11.2013 (danach 200,00 EUR) und für Studenten 50,00 EUR bzw. 60,00 EUR (ohne Tagungsband und Abendveranstaltung).

Termine

20.09.2013	Einreichen der Erstfassung (4-8 Seiten) über Workshop-Webseite
11.10.2013	Benachrichtigung der Autoren
28.10.2013	Abgabe der Endfassung (Hinweise s. Webseite)

Programmkomitee

Christian Boehme
GWDG Göttingen
Dietmar Fey (PII)
Universität Erlangen-Nürnberg
Steffen Limmer
Universität Erlangen-Nürnberg
Bernd Klauer (PII)
Helmut-Schmidt-Universität Hamburg
Erik Maehle (PARS)
Universität Lübeck
Sandro Wartzack
Universität Erlangen-Nürnberg
André Schneider
Fraunhofer IIS/EAS
Marcel Kunze
Forschungsgruppe Cloud Computing, KIT
Thomas Harder
PT-Grid, Leibniz-Institut Greifswald
Jan Treibig
RRZE Erlangen
Jörg Keller (PARS)
FernUniversität Hagen

Organisation

Fraunhofer IIS/EAS Dresden:
Steffen Rülke
André Schneider
Marina Ewert

Veranstalter

- Fraunhofer-Institut für Integrierte Schaltungen IIS, Institutsteil EAS
- DPG/ITG/GI-Fachgruppe »Physik, Informatik, Informationstechnik (PII)«
- GI/ITG-Fachgruppe »Parallel-Algorithmen, -Rechnerstrukturen und -Systemsoftware (PARS)«
- GI/ITG-Fachausschuss «Arbeitsgemeinschaft Simulation (ASIM)«

www.eas.iis.fraunhofer.de/grid4sys

5. Grid4Sys-Workshop (Full Papers)

Basisdienste für die Cloud

Cloud-Anwendungen in der Medizin

Cloud-Anwendungen im Systementwurf

i3sched – Ein OpenNebula Scheduler für die Oracle Grid Engine

Alexander Ditter, Steffen Limmer, Dietmar Fey
Universität Erlangen-Nürnberg, Erlangen, Deutschland, {alexander.ditter, steffen.limmer, dietmar.fey}@cs.fau.de

Kurzfassung

Die effiziente und bedarfsgerechte Nutzung von Rechenressourcen ist einer der entscheidenden Vorteile des Cloud-Computing gebenüber dem herkömmlichen Cluster- und Grid-Computing. Abhängig von den Anforderungen einer Anwendung können Nutzer nur die tatsächlich benötigte Rechenleistung abrufen und diese bei Bedarf auch im laufenden Betrieb erweitern oder verringern. Wir führen diesen Ansatz durch den gleichzeitigen Einsatz von Cluster- und Cloud-Computing auf der selben Hardware-Infrastruktur noch einen Schritt weiter. Ermöglicht wird dies durch den von uns für die Cloud-Middleware OpenNebula, sowie die Oracle Grid Engine (vormals Sun Grid Engine) entwickelten Scheduler, *i3sched*, der den Standard-Scheduler von OpenNebula ersetzt. Er ermöglicht dadurch die Integration und den Betrieb von OpenNebula auf einem bestehenden Cluster, ohne die Batch-Verarbeitung des Clusters zu stören. Wir beschreiben den Aufbau und die Funktionsweise von *i3sched*, sowie die Integration in unsere bestehende Infrastruktur.

1 Einleitung

In diesem Papier stellen wir den für die Cloud-Middleware *OpenNebula* (ON) [1] entwickelten Scheduler, *i3sched*, vor. Wir beschreiben die durch seinen Einsatz mögliche Integration mit der *Oracle Grid Engine* (OGE) [2] und die dadurch ermöglichte parallele Nutzung von Cluster- und Cloud-Computing auf der selben Hardware-Infrastuktur. Die dafür notwendigen Anpassungen, sowie Fehlerbehandlungsszenarien, die durch das Zusammenspiel von OpenNebula und der OGE berücksichtigt werden müssen, werden ebenfalls diskutiert.

Im Bereich des Cloud-Computing gibt es bereits eine Vielzahl von Software und Werkzeuten, die einem beim Einrichten, Betreiben und Warten sowohl von privater als auch öffentlicher Cloud-Infrastruktur unterstützen [1, 3]–[5]. Eine dieser sogenannten Cloud-Middleware-Plattformen ist OpenNebula. Es gehört, zusammen mit OpenStack [4], aktuell zu den populärsten Vertretern und wird von uns für den Betrieb einer Cloud-Infrastruktur verwendet.

1.1 Motivation

In der heutigen Zeit werden, neben den Beschaffungskosten, die Betriebskosten von Großrechenanlagen ein immer größerer Kostenfaktor. Die Betreiber sind daher an einer möglichst hohen Auslastung ihrer Systeme interessiert. Häufig ändern sich jedoch die Anforderungen an die eingesetzten Systeme während der Verwendungsdauer. Der Einsatz von *i3sched* bietet hier den entscheidenden Vorteil, dass eine bestehende Hardware-Infrastruktur zum einen weiterhin als Cluster genutzt werden kann, zum anderen aber auch virtualisierte Ressourcen für OpenNebula zur Verfügung stellen kann.

1.2 Hintergrund

Als Teil einer jeden Cloud-Middleware besteht die Aufgabe eines Schedulers darin, die von einem Nutzer oder dem System angeforderten *virtuellen Maschinen* (VMs) auf vorhandene Rechenressourcen zu verteilen. Dabei ist es erforderlich, dass der Scheduler exklusiv über die entsprechenden Ressourcen verfügen kann bzw., dass diese nicht anderweitig ausgelastet werden. Dieser Ansatz ist jedoch nur dann zweckmäßig, wenn eine Hardware-Infrastruktur ausschließlich für die Cloud-Middleware zur Verfügung steht. Nicht selten kommt es vor, dass bereits bestehende Systeme auch für neue Anwendungen verwendet werden müssen, ohne dabei die bisherige Funktionalität zu beeinträchtigen.

OpenNebula ist, wie bereits erwähnt, neben OpenStack die aktuell am weitest verbreitete und sich am schnellsten weiter entwickelnde Cloud-Plattform. Die Hauptfunktionalität einer solchen Cloud-Middleware besteht darin, die vorhandenen Rechenressourcen zu verwalten, virtuelle Maschinen (VMs) zu Starten, zu überwachen und diese auch wieder zu beenden.

Der typische Lebenszyklus einer VM, wie in vereinfachter Form in Bild 1 dargestellt, besteht bei OpenNebula aus acht Phasen. Zuerst wird für die durch den Nutzer bzw. das System angeforderte VM überprüft ob das entsprechende VM-Image in der Datenbank vorhanden und sich im Zustand Ready befindet. Ist das der Fall, so wird vom Scheduler ein Host, entweder nach Vorgabe des Nutzers oder nach Verfügbarkeit ausgewählt auf dem die VM zur Ausführung kommt. Bis zu diesem Punkt befindet sich die VM im Status Pending. Anschließend wird das Image auf den lokalen Speicher des entsprechenden Hosts kopiert um von dort gestartet zu werden. Diese Phase wird als Prolog bezeichnet und ist abgeschlossen sobald der Hypervisor auf dem Host mit der Ausführung der VM beginnt. Dies wird als Boot Phase bezeichnet. Während ihrer Ausführung befindet sich die VM im Zustand

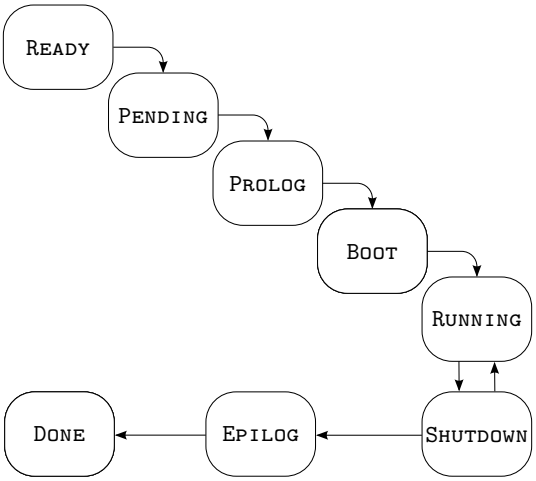


Bild 1. Vereinfachte Darstellung der Zustände einer VM in OpenNebula.

Running; dazu zählen sowohl das interne Hochfahren der Maschine, die Ausführung des Gast-Betriebssystems, sowie das Herunterfahren der VM. Nachdem die VM von OpenNebula mittels ACPI das Signal zum Herunterfahren erhalten hat, ändert sich ihr Status in Shutdown. Dabei ist zu beachten, dass der ON-Scheduler für den Fall, dass die VM ihre Ausführung nicht beendet und sich nach einem gewissen Timeout noch immer im Zustand Shutdown befindet, ihren Status wieder auf Running setzt. War das Herunterfahren hingegen erfolgreich, so wechselt der Status der VM erst in den Zustand Epilog, der andauert bis alle Daten der VM zurück geschrieben sind und die VM wieder vom Host entfernt ist. Danach wechselt der Zustand der VM zu Done. In diesem Zustand taucht die VM nicht mehr in der Liste des Schedulers auf, die damit verküpften Daten, wie z. B. die Konfiguration mit der der Hypervisor die VM erzeugt hat oder die Ausführungsdauer, werden z. B. zu Abrechnungszwecken gespeichert und können auch nachträglich noch abgerufen werden.

Die Problematik der gleichzeitigen Nutzung der Rechenressourcen liegt darin begründet, dass auch die OGE „Jobs“ auf die Hosts verteilt. Hierfür verwaltet sie eine Liste der verfügbaren sowie der belegten Ressourcen. Das parallele Betreiben von ON und OGE wäre funktional zwar ohne Probleme möglich, würde jedoch potentiell zur Überauslastung von Hosts führen. In einem solchen Fall würden ON und die OGE vermeintlich einen Host exklusiv nutzen, während tatsächlich nur jeweils 50% der Rechenleistung bzw. Rechenzeit zur Verfügung stehen.

Für die gemeinsame Nutzung, wie in Bild 2 angedeutet, ist es daher notwendig auch eine gemeinsame Liste für die Verwaltung der Ressourcen zu nutzen. In unserem Ansatz erreichen wir dies indem wir der OGE die Verwaltung der gesamten Ressource überlassen und der ON-Scheduler durch den von uns entwickelten *i3sched* ersetzt wird. Dieser gibt seine Ressourcen-Anforderungen direkt an die OGE weiter. Somit ist sichergestellt, dass es nicht zur Überauslastung einzelner Hosts kommen kann und die OGE zu jedem Zeitpunkt den Überblick über die

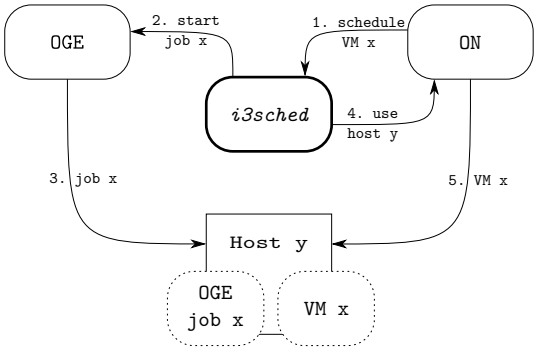


Bild 2. Funktionsweise von *i3sched*.

Auslastung des Gesamtsystems hat.

1.3 Vorarbeiten

Es gibt bereits Lösungen, die einen ähnlichen Ansatz verfolgen. In der Cloud-Plattform Nimbus [5] gibt es z. B. den sogenannten „workspace pilot“, der es ermöglicht Nimbus in ein bestehendes *Portable Batch System* (PBS) Cluster zu integrieren. Hierfür werden sogenannte „pilot jobs“ verwendet, die ebenfalls im Batch-System als Platzhalter für die virtuellen Maschinen fungieren.

Weitere alternative Scheduler für ON sind z. B. *Haizea* [7], der *Green Cloud Scheduler* (GCS) [8] oder das *Energy-aware Multi-start Local Start Metaheuristic for Scheduling VMs in the OpenNebula Cloud* (EMLS-ONC) System [9]. Während Haizea offenbar nicht mehr weiter entwickelt wird, sind von GCS und EMLS-ONC noch Versionen aus dem Jahr 2012 verfügbar. Allerdings ist auch hier, angesichts der rasanten Entwicklung von OpenNebula davon auszugehen, dass keine aktive Weiterentwicklung mehr stattfindet.

2 Architektur

Unser Scheduler ist vollständig in Python (Version 2.7) implementiert. Bild 3 zeigt den Aufbau von *i3sched* anhand eines Klassendiagramms.

Die zentrale Klasse, `Scheduler`, stellt die Hauptfunktionalität bereit – insbesondere die Schleife für den kontinuierlichen Ablauf von *i3sched*. Weiterhin sind vier Hilfsklassen vorhanden: `Log`, `OGE`, `OneRPC` sowie `Config`. Die Klasse `Log` stellt die Logging-Funktionalität zur Verfügung. Die mitprotokollierten Ereignisse dienen hauptsächlich dem Debugging um z. B. feststellen zu können auf welchem Host eine VM ausgeführt werden soll. Die Klasse `OGE` stellt die Schnittstelle zur Oracle Grid Engine bereit. Die Anbindung erfolgt durch die Kommandozeilen Tools *`qsub`*, *`qstat`*, *`qdel`* und *`qhost`* und ist mit der OGE Version 6.2u5 kompatibel. Die Klasse `OneRPC` im `client`-Modul enthält die Schnittstelle zu OpenNebula; die Kommunikation erfolgt über die XML-RPC API und ist aktuell kompatibel bis zur OpenNebula Version 4.0. Die Klasse `Config` stellt Methoden zur Verwendung von Konfigurationsdateien zur Verfügung. Dafür wird das Modul `configobj` genutzt, welches von Michael Foord und Nicola Larosa [10] entwickelt wurde und als Open

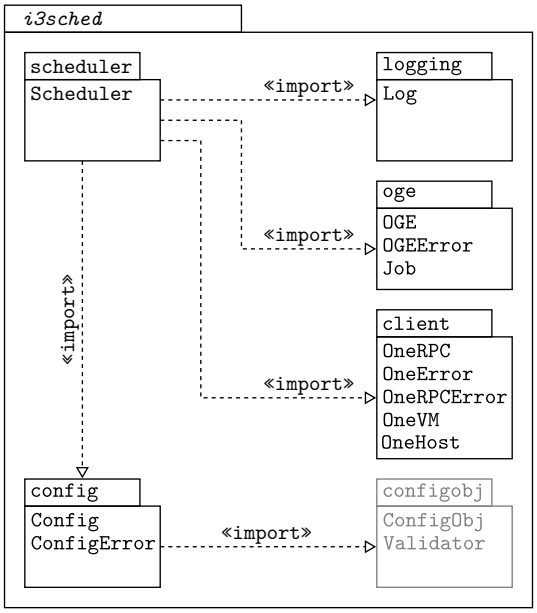


Bild 3. *i3sched* Klassendiagramm.

Source unter der Berkeley Software Distributed (BSD) Lizenz verfügbar ist.

3 Funktionsweise

Beim Start von *i3sched* wird als erstes ein Objekt der Klasse `Scheduler` initialisiert, welches wiederum je ein Objekt der Klassen `Log`, `OGE`, `OneRPC` sowie `Config` erzeugt. Anschließend werden die Informationen der Konfiguration gelesen und die Hauptschleife des Schedulers gestartet. Bild 4 zeigt die Aktivitäten eines Durchlaufs der Hauptschleife.

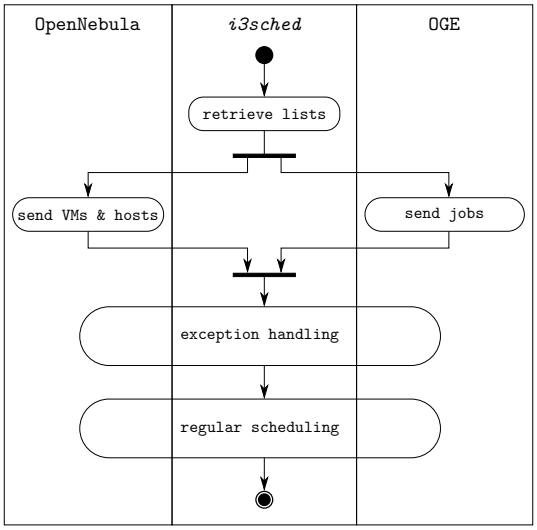


Bild 4. Ablaufdiagramm von *i3sched*.

Am Anfang jeder Iteration aktualisiert *i3sched* seine Informationen, d. h. er ruft die Liste aller VMs und Hosts von OpenNebula, sowie die aller Jobs von der OGE ab. Daran anschließend wird die Fehlerbehandlungsphase durchlaufen, in dieser wird nach Inkonsistenzen gesucht und diese, falls vorhanden, aufgelöst. Danach folgt die Scheduling Phase in der zum starten bereite VMs auf den verfügbaren Hosts, die über ausreichend freie Kapazitäten verfügen, eingeplant und gestartet werden.

Bild 5 zeigt den Ablauf des Einlastens einer VM durch *i3sched*. Als erstes wird für jede VM die sich im Zustand PENDING befindet überprüft, ob ein entsprechender Job in der OGE vorhanden ist und falls ja, auf welchem Host dieser platziert wurde. Anschließend wird OpenNebula angewiesen die entsprechende VM auf diesem Host zu erstellen.

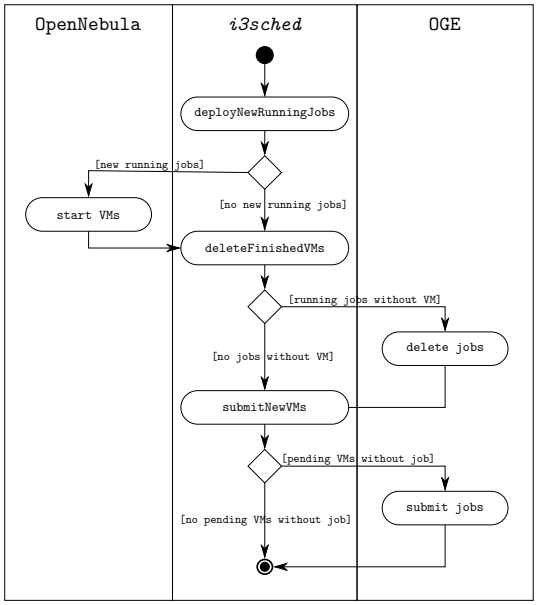


Bild 5. Einlasten einer VM durch *i3sched*.

Danach wird für jede beendete VM überprüft, welchem OGE-Job sie zugeordnet ist. Dies ist vorallem dann relevant, wenn die entsprechende VM seit dem letzten Durchlauf durch OpenNebula beendet wurde. In diesem Fall wird der entsprechende OGE-Job entfernt, damit die entsprechenden Ressourcen auf dem Host wieder freigegeben werden und für neue Jobs zur Verfügung stehen.

Als letztes werden alle sich im Zustand PENDING befindlichen VMs identifiziert, für die noch kein OGE-Job vorhanden ist. Dieser wird dann entsprechend der Anforderungen der VM durch die OGE erstellt. Diese OGE-Jobs sind „Dummy-Jobs“, d. h. in ihnen wird außer einem „Sleep“, nichts getan. Dies kann dazu führen, dass eine VM mit fehlerhafter Konfiguration (z. B. zu viele CPUs) nie gestartet wird, da von der OGE kein entsprechender Job eingelastet und an *i3sched* weitergegeben werden kann. Weiterhin werden von der OGE nur Hosts für ein Einlastung von „Dummy-Jobs“ zugelassen die auch von OpenNebula überwacht werden.

Bild 6 zeigt den Ablauf der Ausnahmebehandlung. Zuerst werden in dieser alle laufenden VMs die der OGE nicht bekannt sind gesucht. Dafür ist es notwendig die gesamte Liste der OpenNebula bekannten VMs zu durchlaufen und diese mit den laufenden Jobs der OGE abzugleichen. Ob ein Job zu einer laufenden VM gehört

oder nicht, kann anhand der bei OpenNebula vermerkten ID sowie dem Namen des OGE-Jobs festgestellt werden, da der Name des OGE-Jobs die entsprechende ID enthält. Falls die Funktion `shutdownRunningZombies` tatsächlich eine VM ohne entsprechenden Eintrag in der OGE finden sollte, so wird diese durch den entsprechenden XML-RPC Aufruf durch OpenNebula beendet. Dieses Vorgehen ist von essentieller Bedeutung für die reibungslose Kooperation von OpenNebula und der OGE.

Im zweiten Schritt der Fehlerbehandlung wird überprüft, dass sich keine VM länger als eine bestimmte Zeit im Zustand SHUTDOWN befindet. Dieses Vorgehen ist notwendig, da VMs vereinzelt nicht vollständig von ON beendet werden können und in diesem Fall als „Leichen“ sowohl in OpenNebula als auch in der OGE verbleiben würden. Dies könnte mit steigender Betriebsdauer dazu führen, dass Hosts mit VMs belegt erscheinen die eigentlich schon längst beendet sind und damit das gesamte System zu Stillstand bringen. Wenn *i3sched* eine VM findet, die sich bereits länger als das vorgegebene Zeitlimit im entsprechenden Status befindet, so wird diese direkt auf dem Host durch den Hypervisor (z. B. bei XEN mit `xm destroy`) beendet. Nach dem erfolgreichen Beenden werden OpenNebula und die OGE über den neuen Zustand informiert.

Im letzten Schritt der Fehlerbehandlung werden alle sich im Zustand UNKNOWN befindlichen VMs beendet, da diese nicht mehr von OpenNebula angesprochen werden und daher auch nicht mehr durch OpenNebula beendet werden können. Auch diese VMs werden direkt auf dem entsprechenden Host durch den Hypervisor (via SSH) beendet.

Nach Durchlauf der Fehlerbehandlung sind alle Inkonsistenzen aufgelöst und das normale Scheduling wird durchlaufen. Wie man ebenfalls in Bild 6 sieht, ist die OGE an der Fehlerbehandlung nicht beteiligt, da sich diese nur auf die VMs bezieht.

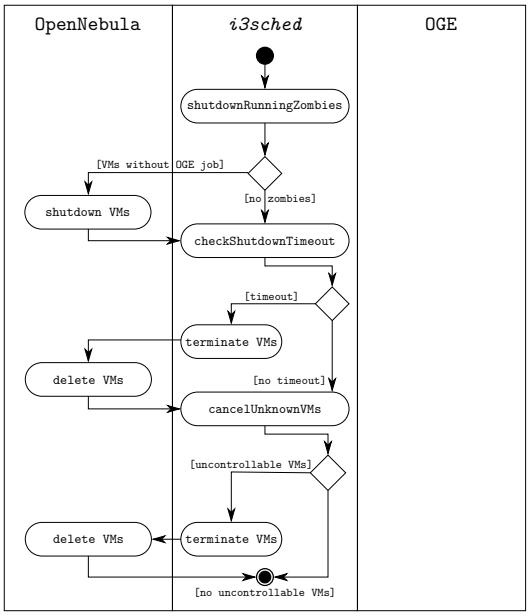


Bild 6. Fehlerbehandlung in *i3sched*.

4 Zusammenfassung und Ausblick

Mit *i3sched* haben wir eine einfache, aber wichtige Erweiterung zum Standard-Scheduler von OpenNebula entwickelt. Der Einsatz ist aktuell nur im Zusammenspiel mit der OGE möglich, allerdings ist eine Erweiterung für andere Batch-Scheduling-Systeme, wie z. B. Slurm [11], geplant, de diese in Zukunft auch in unserem Cluster zum Einsatz kommen sollen. Eine Erweiterung für weitere Cloud-Middleware, wie z. B. OpenStack, ist aktuell nicht geplant. Zum einen, weil OpenNebula mittelfristig bei uns weiter im Einsatz bleiben wird und zum anderen die Machbarkeit einer Erweiterung für andere Cloud-Middleware maßgeblich von deren Architektur abhängig ist.

5 Danksagung

Diese Arbeit entstand im Rahmen des vom Bundesministeriums für Wirtschaft und Technologie (BMWi), durch Beschluss des deutschen Bundestags, geförderten Projekts Cloud4E.

Literatur

[1] OpenNebula Webseite: *http://opennebula.org/* (Abgerufen am 20. September 2013).

[2] Oracle Grid Engine Webseite: *http://www.oracle.com/us/products/tools/oracle-grid-engine-075549.html* (Abgerufen am 20.09.2013).

[3] Eucalyptus Webseite: *http://www.eucalyptus.com/* (Abgerufen am 20. September 2013).

[4] OpenStack Webseite: *http://www.openstack.org/* (Abgerufen am 20. September 2013).

[5] Nimbus Webseite: *http://www.nimbusproject.org/*, (Abgerufen am 20. September 2013).

[6] T. Freeman and K. Keahey, *Flying Low: Simple Leases with Workspace Pilot*, Euro-Par 2008 – Parallel Processing, LNCS, vol. 5168, pp. 499–509, Springer Berlin Heidelberg, 2008.

[7] B. Sotomayor, K. Keahey, and I. Foster, *Combining batch execution and leasing using virtual machines*, Proceedings of the 17th international symposium on High performance distributed computing, pp. 87–96, ACM, New York, NY, USA, 2008.

[8] T. Cioara, I. Anghel, I. Salomie, G. Copil, D. Moldovan, and A. Kipp, *Energy Aware Dynamic Resource Consolidation Algorithm for Virtualized Service Centers Based on Reinforcement Learning*, 10th International Symposium on Parallel and Distributed Computing (ISPDC), 2011, pp.163–169, 2011.

[9] Y. Kessaci, N. Melab, and E. Talbi, *An energy-aware multi-start local search heuristic for scheduling VMs on the OpenNebula cloud distribution*, International Conference on High Performance Computing and Simulation (HPCS), 2012, pp. 112–118, 2012.

[10] M. Foord and N. Larosa, *Reading and Writing Config Files – ConfigObj 4 Introduction and Reference*, verfügbar unter: *http://www.voidspace.org.uk/python/configobj.html*, (Abgerufen am 20. September 2013).

[11] SLURM Webseite: *http://slurm.schedmd.com/slurm.html*, (Abgerufen am 20. September 2013).

Serviceorientierte Simulation auf Basis von OCCI am Beispiel der Finite Elemente Methode

Maik Srba, Gesellschaft für wissenschaftliche Datenverarbeitung mbH Göttingen (GWDG), Am Faßberg 11, 37077 Göttingen, maik.srba@gwdg.de
Dr. Simon Schmitz, ERAS GmbH, Hannah-Vogt-Straße 1, 37085 Göttingen, info@eras.de

Kurzfassung

Das Open Cloud Computing Interface (OCCI) und seine Erweiterbarkeit auch während der Laufzeit bieten die Möglichkeit bestehende Anwendungen als Service bereitzustellen. Eine Implementierung des OCCI Modells ist der rOCCI Server. Dieser wurde erweitert um das AMQP Protokoll, damit ein schneller und asynchroner Nachrichtenaustausch zwischen den verteilten Anwendungen möglich ist. Mit einer Kombination aus OCCI Service Adapters und einer simplen DSL zur Beschreibung der Anwendungsschnittstelle ist es möglich, aus bereits vorhandenen Simulationen serviceorientierte Simulationen zu implementieren, die über AMQP und OCCI vernetzt werden können. Diese sind zugleich portierbar auf andere Cloud Infrastrukturen, da sie unabhängig von den proprietären APIs der Dienstanbieter implementiert werden. Der Portierungsaufwand bestehender Simulationslösungen zwischen den Cloud Infrastrukturen wird so verringert und teilweise sogar unnötig. Als Beispiel für eine Umsetzung als serviceorientierte Simulation wurde eine bestehende FEM-Simulation ausgewählt.

FEM-Simulationen stellen an die Rechenleistung von Computern hohe Ansprüche. Dadurch ist die Berechnung von FEM-Modellen – gerade für kleine und mittelständische Unternehmen (KMUs) – mit hohen Kosten allein für die Anschaffung und den Unterhalt der Hardware verbunden. Ebenso sind Lizenzkosten für die Software nötig, die unabhängig von ihrer Auslastung anfallen. Die ERAS GmbH arbeitet im Rahmen von „Cloud4E - Trusted Cloud Computing for Engineering“ (als Projekt des Technologieprogramms „Trusted Cloud“ des BMWi) an der Bereitstellung eines Dienstes für FEM-Simulationen, in der Cloud. Dadurch können rechenintensive Analysen mechanischer Problemstellungen in die Cloud ausgelagert werden.

1 Übersicht OCCI

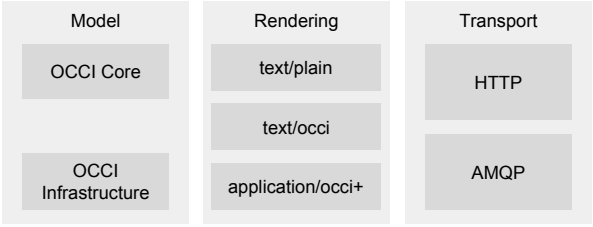


Abbildung 1: Übersicht der OCCI Komponenten.

Das Open Cloud Computing Interface (OCCI) [1] ist einer der ersten Ansätze, um eine offene Schnittstelle für Cloud Service Provider zu standardisieren. In erster Linie ist es eine API, welche von Klienten und Brokern genutzt werden kann, um auf Cloud Services zuzugreifen. OCCI bietet ein klares Interface, welches an existierende APIs angepasst werden kann. OCCI bemüht sich die Interoperabilität zwischen den verschiedenen Service Providern herzustellen und koexistiert dabei neben den bestehenden proprietären APIs.

OCCI bietet einige einzigartige Features, wie die Auflistung von Fähigkeiten des Dienstleisters und die Erweiterbarkeit während der Implementierung des Services bzw. dessen Laufzeit. Dies ermöglicht es den Dienstleistern eine Sprache zu sprechen, um alle Arten von Services (IaaS bis SaaS) auf einem einheitlichen Weg anzubieten.

1.1 OCCI-Modell

1.1.1 OCCI Core

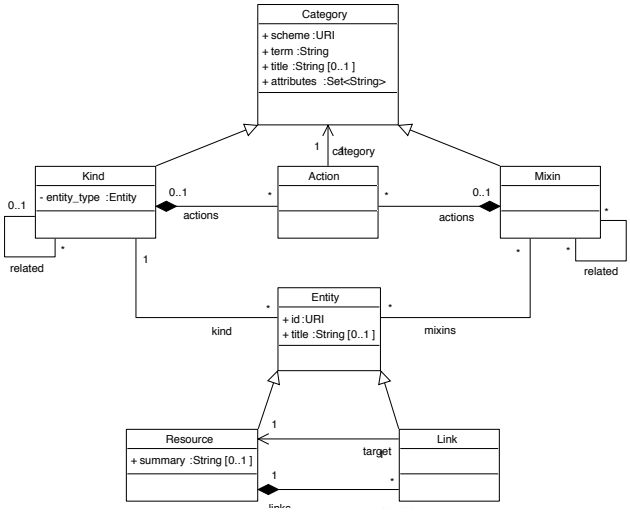


Abbildung 2: Aufbau des OCCI Core Modells.

Das OCCI Core Modell beschreibt die verfügbaren OCCI Klassen. Das OCCI Core Modell stellt ein minimales Set an notwendigen Klassen zum Erstellen eines

OCCI Service zur Verfügung. Es ist eine Ansammlung von Kinds, Mixins und Actions, die von der abstrakten Klasse „Category“ abgeleitet sind. OCCI Kinds definieren den Typ Identifier und die verfügbaren Aktionen und Attribute der OCCI Entities. Die Klasse OCCI Entity ist die abstrakte Basisklasse für die Klassen OCCI Ressource und OCCI Link. Eine OCCI Ressource beschreibt eine tatsächliche Cloud Ressource durch Attribute als Key-Value-Paare sowie anwendbare Aktionen. OCCI Links sind Verbindung zwischen OCCI Ressourcen und anderen Ressourcen. Die Verbindung selber kann Attribute und anwendbare Aktionen enthalten. Die verfügbaren Attribute und Aktionen einer OCCI Entity können durch OCCI Mixins erweitert werden. Diese enthalten zusätzliche Definitionen für Attribute und Aktionen. Solche Erweiterungen können während der Implementierung und der Laufzeit des OCCI Services realisiert werden.

1.1.2 OCCI Infrastruktur

Das OCCI Infrastruktur Modell [2] beschreibt eine OCCI Core Model Erweiterung zum Verwalten einer Infrastructure as a Service. Es ist eine Sammlung von Klassen zur Bereitstellung von grundlegenden Speicher-, Netzwerk- und Compute-Ressourcen. Diese basiert auf dem Infrastructure as a Service-Modell der „NIST Definition of Cloud Computing“ [3].

Die grundlegenden Klassen im OCCI Infrastructure Model sind Compute, Network, Storage Networkinterface und Storagelink. Compute, Network und Storage sind als OCCI Ressourcen implementiert. Networkinterface und Storagelink sind als OCCI Link implementiert, sie ermöglichen es jeweils Speicher oder Netzwerkkomponenten zu den Compute Ressourcen hinzuzufügen.

Diese OCCI Ressourcen und OCCI Links beinhalten die Attribute und Aktionen, die notwendig sind, um die zur Verfügung gestellte Infrastruktur zu verwalten.

1.1.3 OCCI Erweiterbarkeit

OCCI bietet die Möglichkeit mit Hilfe von Mixins die Fähigkeiten der angebotenen Services zu erweitern. Dies kann auch während der Laufzeit geschehen. Mixins können somit genutzt werden, um die Anbieter-spezifischen Besonderheiten in die OCCI Schnittstelle einzufügen, ohne den Standard zu brechen. Darüber hinaus eignet sich dieser Mechanismus, um die Schnittstellen eines Services aus der Software as a Service Ebene heraus zu beschreiben.

OCCI Links sind eine weitere Möglichkeit das OCCI Model zu erweitern. Diese erlauben neben dem Verlinken von OCCI Ressourcen auch das Anbinden externer Ressourcen, welche keine OCCI Ressourcen sind. Speziell ist dies nützlich, um das Nachrichtenprotokoll AMQP [4] anzubinden.

1.2 Transport

OCCI ist ein RESTful Protocol und eine API. Die Definition für den HTTP Transport ist im OCCI HTTP Rendering Dokument [5] beschrieben. Das REST Interface erlaubt es Klientanwendungen mit den verschiedenen OCCI Services über HTTP zu interagieren. Die verfügbaren Operationen entsprechen den HTTP Request Operationen POST, GET, PUT und DELETE.

1.3 Rendering

Das OCCI Rendering schreibt verschiedene HTTP Content Typen vor. Diese Content-Typen sind *text/plain*, *text/occi*, *text/uri-list*, *Application/occi-json* und *application/occi-xml*. Der Standard Content-Type und Fallback ist *text/plain*. Der Aufbau der einzelnen Content-Types ist im OCCI HTTP Rendering Dokument beschrieben [5].

2 Simulation as a Service mit rOCCI

2.1 rOCCI Übersicht

rOCCI ist ein OCCI Service Implementierung in Ruby und wurde gefördert von der European Commission Information Society. Es ist ein Open Source Projekt unter der Apache2 Lizenz. Der Quellcode ist erreichbar unter der Adresse [6].

2.1.1 Architektur

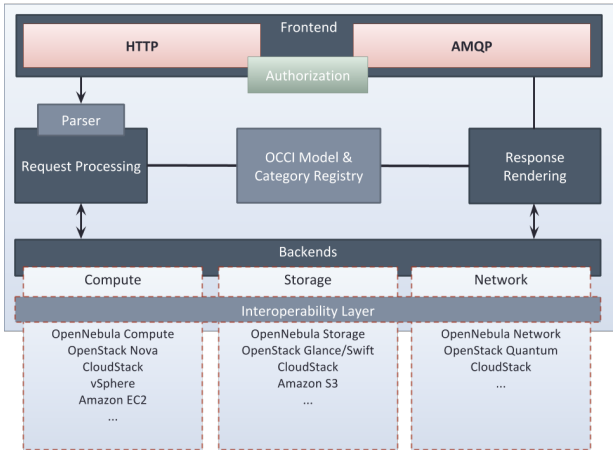


Abbildung 3: Übersicht der rOCCI Server Architektur mit Frontend, Core und Backend.

rOCCI besteht im Wesentliche aus zwei Projekten. Das rOCCI-Server Projekt und das rOCCI Projekt. Das rOCCI Projekt implementiert das OCCI Protokol und die klientseitige Anbindung an OCCI, wie es im Kapitel 1 beschrieben ist. Der rOCCI-Server implementiert einen Server, der die Anbindung an verschiedene proprietäre APIs der Dienstanbieter vereinfacht. Die Architektur von rOCCI ist unterteilt in Frontend, Core und Backend.

Das Frontend implementiert das im Kapitel 1 beschriebene OCCI Transport und Rendering des OCCI Models. Das Frontend ist durch weitere Transportprotokolle erweiterbar. Derzeit ist HTTPs und AMQPs als Transportprotokoll umgesetzt. Im Frontend ist die Authentifizierung der Klientanwendungen implementiert. Es wird die HTTP-Basis-Authentifizierung, HTTP Digest Authentifizierung, X.509 Authentifizierung und Keystone unterstützt. Mit der X.509 Authentifizierung ist auch eine

VOMS-Unterstützung implementiert, wie sie im Grid Umfeld bekannt ist.

Im Core ist das OCCI Model umgesetzt. Alle Anfragen an das Frontends werden durch den OCCI Parser in das OCCI Model übersetzt. Dabei werden die aktuellen OCCI Ressourcen, OCCI Links und OCCI Mixins in der „Category Registry“ gespeichert. Das Model ist stateless und wird mit den Informationen aus den angebundenen proprietären APIs aufgebaut. Das Model wird dann an das Backend übergeben. Alle Antworten des Backends werden durch den Core in den jeweiligen Content Type gerendert. Das Ergebniss wird an das Frontend übergeben.

Durch das Backend wird das OCCI Model auf die verschiedenen proprietären APIs umgesetzt. Es bildet daher die Interoperabilitätsschicht. Damit ist es möglich, OCCI mit unterschiedlicher proprietärer Cloud Middelware wie Opennebula, Openstack Nova, Openstack Swift, CloudStack, Amazon EC2 oder Amazon S3 einzusetzen. Die Unterteilung des Backends orientiert sich an den von der „NIST Definition of Cloud Computing“ beschriebenen Komponenten des Service-Modells.

2.1.2 AMQP Erweiterung

Das Advanced Message Queuing Protocol (AMQP) [4] ist ein offenes Internet Protokoll für den Nachrichtenaustausch zwischen Anwendungen oder Organisationen. Es wird vorwiegend genutzt, um verteilte Anwendungen auf einem standardisierten Weg zu verknüpfen. AMQP bietet dabei Mechanismen zur Absicherung, Skalierung, Interoperabilität und Standardisierung des Nachrichtenaustausches.

Die Erweiterung des rOCCI Frontends durch AMQP erlaubt rOCCI einen asynchronen Nachrichtenaustausch.

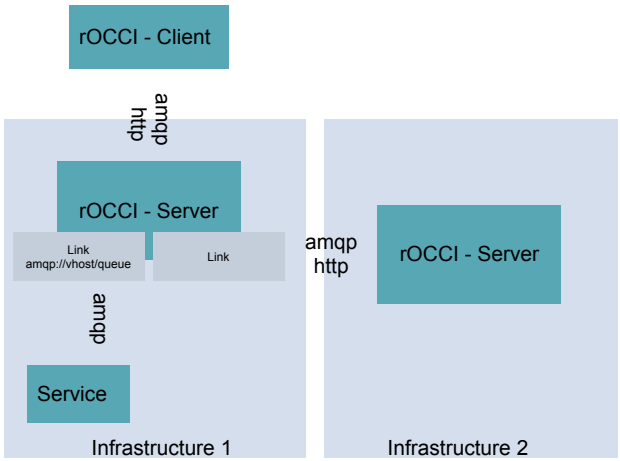


Abbildung 4: AMQP als Transportprotokoll und OCCI Link.

Dadurch werden die zumeist langläufigen Backendprozesse von der Kommunikation der Klientanwendung entkoppelt. Änderungen, die durch das Backend erfolgen, können an den Klient übermittelt werden. Dies verhindert unnötige Request der Klientanwendung an den OCCI Server. Eine weiterer Vorteil liegt darin, dass durch AMQP die Verarbeitung der OCCI Anfragen auf mehrere rOCCI Server verteilt werden kann und die Nachrichten priorisiert werden können, wodurch hier eine Implementierung von SLA gesteuerten Services möglich ist.

AMQP wird im Backend des Servers eingesetzt. Der rOCCI Server ist hier durch eine AMQP Implementierung des OCCI Links erweitert worden. Dies ermöglicht es OCCI Ressourcen, andere rOCCI Server oder nicht-OCCI Ressourcen über AMQP an bestehende OCCI Ressourcen anzubinden. Ein Beispiel hierfür ist der OCCI Service Adapter.

2.2 OCCI Service Adapter

Der OCCI Service Adapter ist ein Beispiel für die Anwendung der OCCI AMQP Erweiterung. Er ermöglicht die Integration bestehende Anwendungen über AMQP in das OCCI Model. Damit kann der rOCCI Server als Service Broker eingesetzt werden, der über die Grenzen eines Dienstanbieters hinaus Services anbieten kann.

2.2.1 Aufbau

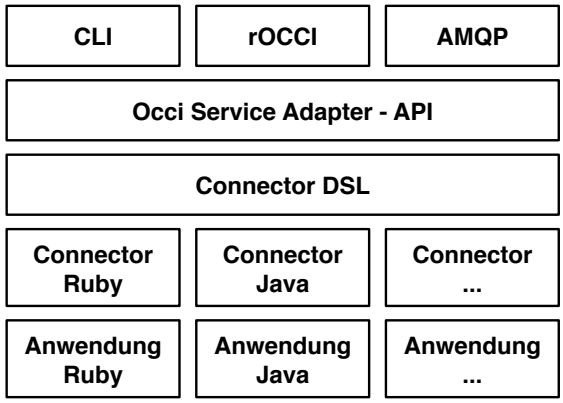


Abbildung 5:Aufbau des OCCI Service Adapters.

Der OCCI Service Adapter besteht im Wesentlichen aus 3 Ebenen. Die erste Ebene besteht aus dem Command Line Interface, der rOCCI API und dem AMQP Protokoll. Diese ermöglichen den Zugriff auf den OCCI Service Adapter über die Kommandozeile oder über das OCCI Protokoll.

Die API nutz die Informationen aus den Connectoren, um diese in das OCCI Model zu integrieren. Beim Start eines Connectors verbindet der OCCI Service Adapter den Connector mit einer AMQP Queue und meldet den Connector als neue OCCI Ressource an den rOCCI Server; damit ist der Connector erreichbar. Alle Anfragen an den Connector über diese Queue geschehen im OCCI Format und werden auf die Methoden des jeweiligen Connectors übertragen.

Die Connector DSL dient zur Beschreibung einer Anwendung als Connector. Derzeit können Connectoren in Ruby und Java geschrieben werden. Die Connectoren beinhalten die Schnittstelle zur Applikation und eine Beschreibung, wie diese Installiert werden können. Mit Hilfe der Connectoren können existierende Anwendungen

mit minimalem Aufwand als Cloudapplikation breitgestellt werden.

Die CLI des OCCI Service Adapters hilft bei der Erstellung des Connectors. Er erzeugt dazu eine Verzeichnisstruktur, in der die Anwendung mit der Installationsroutine abgelegt und der Connector für die Anwendungsschnittstelle beschrieben wird. Ein Connector besteht immer aus einem Header und einer Ansammlung von definierten Actions und Parametern. Der Header beinhaltet Informationen, welche die Anwendung als Service identifiziert. Diese Informationen enthalten Daten über den Dienstanbieter und deren Anwendung. Die Abschnitte „Actions“ und „Parameter“ beinhalten die Beschreibung der Schnittstelle und das Mapping auf die angebundene Anwendung. „Actions“ sind dabei Methoden, die Parameter beinhalten und die einen Rückgabewert haben können. Die „Actions“ können asynchron ausgeführt werden. Die Rückgabewerte werden im asynchronen Fall dann über eine Callback-Methode an den Client zurückgeliefert. Durch die Möglichkeit innerhalb eines Connectors einen weiteren Connector anzufordern können mehrere Anwendungen miteinander vernetzt werden.

Der OCCI Service Adapter ist nicht nur in der Lage einen Connector zu starten, sondern er kann diesen vom rOCCI Server auch abrufen. So kann ein Client die Schnittstelle zu einer Anwendung über den Connector holen. Der Connector wird dann dem Klient als Interface bereitgestellt. Alle Methoden- oder Parameteraufrufe werden dann über AMQP im OCCI Format an den verteilten Connector geschickt, wodurch die Anwendung gesteuert werden kann.

2.2.2 Integration bestehender Simulationen

Im Folgenden wird an Hand eines einfachen Beispiels beschrieben, wie man eine Anwendung mit Hilfe des OCCI Service Adapters als Cloud Service bereitstellen kann. Ausgangspunkt ist eine Anwendung, die bisher nur als lokale Shell Anwendung existiert. Man kann diese Anwendung bereits über Shell starten und stoppen.

Über den Befehl „occi-service-adapter connector init shellbeispiel“ erzeugt man sich einen neuen Connector. Man erhält ein Verzeichnis mit dem Aufbau aus Abbildung 6.

```
drwxr-xr-x  9 msrba  staff  306 27 Sep 14:33 .
drwxr-xr-x  9 msrba  staff  306 27 Sep 14:33 ..
drwxr-xr-x  2 msrba  staff   68 27 Sep 14:33 .occi
drwxr-xr-x  2 msrba  staff   68 27 Sep 14:33 application
drwxr-xr-x  3 msrba  staff  102 27 Sep 14:33 connector
drwxr-xr-x  3 msrba  staff  102 27 Sep 14:33 install
drwxr-xr-x  2 msrba  staff   68 27 Sep 14:33 log
drwxr-xr-x  5 msrba  staff  170 27 Sep 14:33 test
total 8
-rw-r--r--  1 msrba  staff  120 27 Sep 14:33 connector.yml
```

Abbildung 6: Aufbau einer Connector Klasse.

Der Connector ist sofort als Beispiel lauffähig. Im Verzeichnis „test“ wurden Testskripte erzeugt, die man mit dem Befehl „occi-service-adapter connector test <script>“ aufrufen kann. Es gibt zu Beginn 3 Skripte, für einen direkten Test, sowie für einen lokalen und einen remote Test des Connectors. Die Unterschiede zwischen den Tests liegen in der Art, wie der Connector aufgerufen wird. Im direkten Test wird die Klasse des Connectors als Objekt in die Testumgebung geladen. In einem lokalen Test wird der Connector als eigenständiger Prozess in der lokalen Umgebung gestartet und im remote Test wird der Connector auf der ausgewählten Cloud Plattform gestartet. Im Verzeichnis Connector befindet sich die Klasse für

```
$LOAD_PATH.unshift File.dirname(__FILE__)

require 'occi_service_adapter/connector/service2'
module OcciServiceAdapter
  module Connector
    class Shellbeispiel < OcciServiceAdapter::Connector::Service2
      provider_namespace 'http://gwdg.de/simulation'
      version '0.1'
      term 'Shellbeispiel'
      title 'My First Connector'

      attribute :example_parameter,
        {:type => 'String', :required => false, :default => 'old value'}

      action :my_action,
        {:my_parameter => {:type => 'String', :require => true}}
      def my_action(params={})
        @example_parameter='new value'
        puts 'parameter: ' + params[:my_parameter]

        h = {:test => 'test value', :test2 => 'test2 value'}
        a = [1, 2, 3, 4, 5, 6]
        a[1] = h
        return a
      end
    end
  end
end
```

Abbildung 7: Verzeichnisstruktur eines Connectors.

den Connector. In unserem Beispiel ist diese wie folgt aufgebaut:

In der Abbildung 7 ist zu sehen, wie durch die Schlüsselwörter „actions“ und „parameter“ die Schnittstelle der Anwendunge in einem Connector definiert werden. Die Attribute müssen nicht implementiert werden, da diese automatisch als Getter und Setter bereitgestellt werden. Die Actions werden durch Methoden definiert, die den angegeben Actionnamen haben („start“ im Beispiel aus Abbildung 7). Hier ist die Implementierung enthalten, wie auf die Schnittstelle der Anwendung zugegriffen wird. Alle Rückgaben der Methode werden über AMQP an den Klient zurückgeliefert.

Der Connector wird dann an einen Service Register geliefert. Der Befehl hierfür ist „occi-service-adapter connector push“. Damit steht er der Cloud Plattform zur Verfügung. Durch Verwendung des OCCI Service Adapters in einer eigenen Client Anwendung kann die im Connector angebundene Anwendung als Service gestartet werden. In unserem Fall würde man den folgenden Code nutzen, um einen Service über OCCI zu starten und einen Service als Objekt zu bekommen.

```
command = OcciServiceAdapter::Command::Service::Start.new
options = {
   :service_identifier =>
        'http://gwdg.de/simulation#shellbeispiel'
}
service = command.run options
service.wait_for OcciServiceAdapter::Service.SERVICE_READY

response = service.connector.start '1'

puts response
```

Abbildung 8: Erstellen eines Service Objektes an Hand des Beispielconnectors.

Über das Service-Objekt kann man auf den Connector zugreifen. Dieses beinhaltet die Parameter und Methoden des Connectors, welcher auf der Cloud Plattform ausgeführt wird. Damit ist die Schnittstelle in der Umgebung des Klienten verfügbar und kann wie ein lokales Objekt verwendet werden.

3 Anwendungsbeispiel

Die Finite-Element-Methode (FEM) [7] ist ein Werkzeug, um physikalische Prozesse zu beschreiben. Computerbasierte Simulationen auf Basis von FE-Modellen sind etablierte Verfahren in weiten Bereichen der Technik. Mittels einer FEM-Software werden aus dem FE-Modell die zur Lösung notwendigen Matrizen assembliert und durch den eingebauten Matrix-Löser invertiert. Auch dieser Schritt wird, obwohl er oft große Mengen an CPU-Zeit benötigt, bei KMUs regelmäßig auf der Workstation des Ingenieurs durchgeführt. Eine vertrauenswürdige und leistungsfähige Cloud-Lösung bietet hier ganz neue Möglichkeiten.

Im Rahmen des Projektes „Cloud4E - Trusted Cloud Computing for Engineering“ [8] des Technologieprogramms „Trusted Cloud“ arbeitet die ERAS GmbH an der Bereitstellung eines Programmpakets zur Assemblierung und Lösung von FEM-Systemmatrizen als Dienst in der Cloud. Dieses Paket basiert auf einem freien FORTRAN Quellcode. Im Gegensatz zu kommerzieller FEM-Software ist der Funktionsumfang allerdings eingeschränkt. Zum einen stehen nicht alle Analysetypen (z.B. nicht-lineare Analysen) zur Verfügung, die ein kommerzielles Programm wie Nastran [9] bietet. Zum anderen ist die Bibliothek der verwendeten Elementtypen im Vergleich zu Nastran beschränkt. Dessen ungeachtet reicht das Programmpaket für die meisten mechanischen Problemstellungen aus und kann somit eine Alternative für KMUs sein.

3.1 Workflow in der Cloud

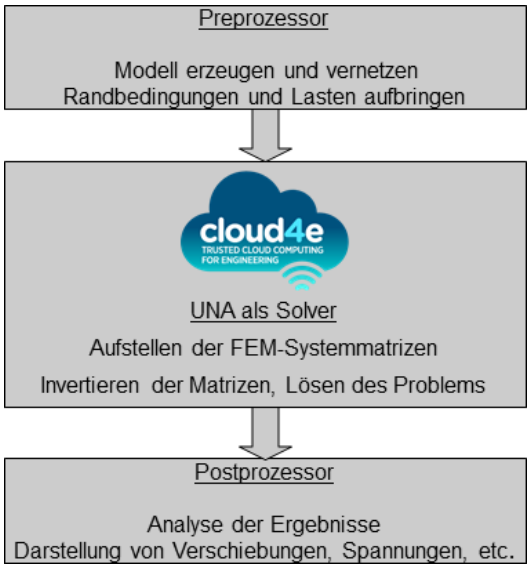


Abbildung 9: Workflow einer FEM-Simulation in der Cloud.

Der Workflow (Abbildung 9) einer FEM-Simulation lässt sich in drei Schritte unterteilen:

- Aufbau/Import des Modells.
- Berechnung mittels FEM-Software.
- Analyse des Simulationsergebnisses.

Diese drei Schritte werden gewöhnlich mit kommerziellen Programmen auf der Hardware des Ingenieurs durchgeführt. Bei Verwendung eines Cloud-basierten FEM-Dienstes wird der zweite Schritt in die Cloud ausgelagert. Dazu wird das Simulationsprogramm über den OCCI-Service-Adapter in die Simulationsinfrastruktur integriert. Das ist die Aufgabe des Softwareanbieters.

Für den Anwender auf der anderen Seite sollte die Auslagerung von FEM-Berechnungen in die Cloud mit möglichst wenig Mehraufwand im Vergleich zur Einzelplatzlösung verbunden sein. Die Akzeptanz einer Cloud-basierten FEM-Software hängt im Wesentlichen von zwei Punkten ab:

- Der (an seine lokale Software gewöhnte) Anwender erwartet von einem Cloud-Dienst einen ähnlichen Komfort, den er bei sich lokal hat. Das bedeutet, dass das Lösen der FE-Modelle in der Cloud den gewohnten Workflow nicht zu stark verkomplizieren darf.
- Der Benutzer erwartet eine höhere Rechenleistung im Vergleich zur Einzelplatzlösung, d.h. die Leistungssteigerung durch einen Cloud-Dienst muss spürbar sein.

Auf den ersten Punkt wird in diesem Stadium des Projektes noch nicht eingegangen, da die Voraussetzungen dafür noch nicht geschaffen sind. Der zweite Punkt wird anhand des einfachen Beispiels eines Biegebalkens untersucht.

3.2 Simulation am Beispiel „Biegebalken“

In Abbildung 10 (oben) ist der einseitig eingespannte Biegebalken schematisch dargestellt. Am freien Ende greift eine Kraft an, die zu einer Auslenkung führt. In der Abbildung 10 (unten) ist der Biegebalken als unausgelenktes FE-Modell gezeigt. Diesem Bild überlagert ist eine Darstellung der Spannung im Material, die bei der Auslenkung entsteht. Auf das Modell soll nicht näher eingegangen werden, ausführliche Informationen finden sich in der einschlägigen Literatur [10].

Für das in Abbildung 10 zu sehende Modell soll eine Frequenzganganalyse berechnet, d.h. es soll die Fre-

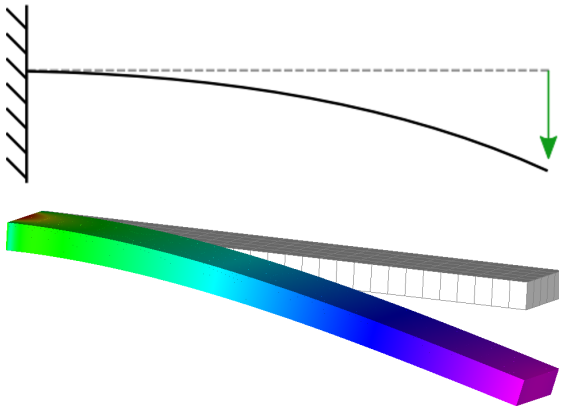


Abbildung 10: Beispiel "Biegebalken" mit schematischer Darstellung (oben) und Darstellung als FE-Modell (unten).

quenzantwort im Bereich von 0 Hz bis 1000 Hz ermittelt werden. Dazu wird das Intervall [0 Hz; 1000 Hz] durch viele Stützstellen zerlegt, an denen jeweils die Frequenzantwort berechnet wird. Für dieses Beispiel ist die Aufstellung/Invertierung der FEM-Systemmatrizen nur einmal nötig, die Matrixgleichung wird danach für jede einzelne Stützstelle des Frequenzbereichs gelöst. Aus diesem Grund ist die Aufgabe gut auf mehrere Instanzen innerhalb der Cloud verteilbar: Jede Instanz erhält dabei das volle Modell zur Aufstellung der benötigten Matrixgleichung, aber nur einen Ausschnitt des Frequenzbereichs. Nachdem alle Instanzen ihren Frequenzbereich berechnet haben, werden alle diese Teilergebnisse zur Bereitstellung des endgültigen Ergebnisses gesammelt. Das Ergebnis für das Modell aus Abbildung 10 ist in Abbildung 1111 zu sehen.

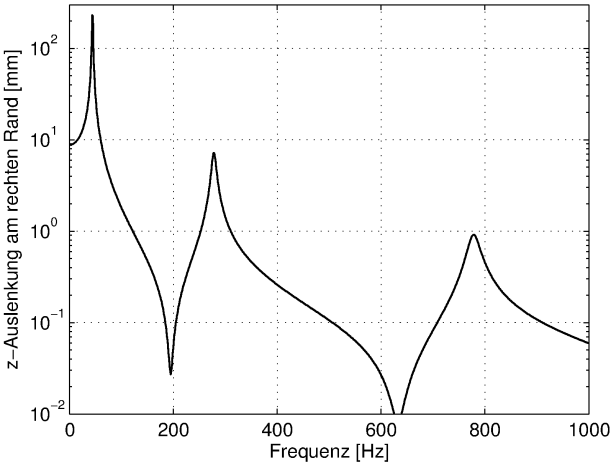


Abbildung 11: Frequenzgang des Biegebalken-Modells mit drei Resonanzen.

Aufgetragen ist die Auslenkung am rechten äußeren Rand des Balkens. Dort tritt die maximale Schwingungsamplitude auf. Gut erkennbar sind die Resonanzen des Biegebalkens bei 44.5, 278.4 und 779.2 Hz.

3.3 Leistungssteigerung durch die Cloud

Die Auswirkungen der Parallelisierung auf mehrere Instanzen in der Cloud wurden untersucht, indem dasselbe Modell auf unterschiedlich viele Instanzen aufgeteilt wurde. Dabei wurde darauf geachtet, dass jede Instanz die gleiche Anzahl an Stützstellen zur Berechnung erhält.

Es wurden die Zeiten gemessen, die die Ausführung der gesamten Berechnung in der Cloud erfordert, also vom Anfordern der Berechnung bis zum Ergebnis. Dies beinhaltet

- Die Zeit, die vergeht, bis die Instanz verfügbar ist. Darin ist die Anforderung des richtigen VM-Images, dessen Bootvorgang und die Initialisierung des Service auf der Instanz enthalten.
- Die Zeit, die vom Start der Simulation bis zu ihrem Ende vergeht. Das beinhaltet für jede Simulation das Lesen des Modells mit Aufstellung der Systemmatrix, die Lösung für die angeforderten Frequenz-Stützstellen und das Schreiben des Ergebnisses.

In Abbildung 12 ist die Zeit bis zur Verfügbarkeit des Services dargestellt. Wie zu erwarten ist diese Zeit unabhängig von der Anzahl der Instanzen. Im Mittel dauert es ca. 95 Sekunden, bis die komplette Instanz bereitgestellt ist. Allerdings ist die Streuung (angezeigt durch die Fehlerbalken, die die Standardabweichung angeben) recht hoch.

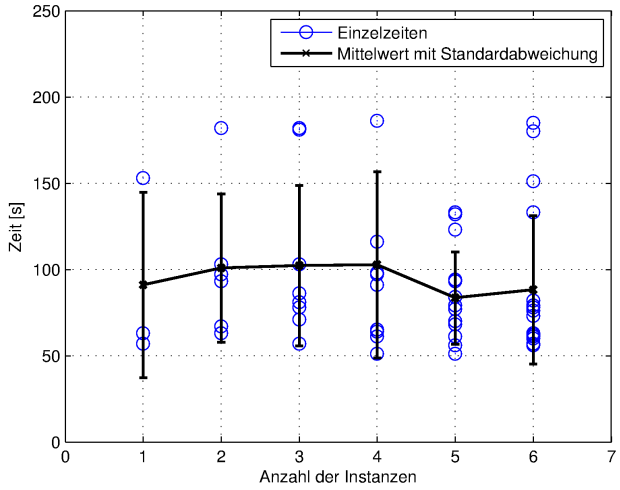


Abbildung 12: Zeit bis zur Verfügbarkeit des Services.

An dieser Stelle ist noch Optimierungsbedarf. Die Streuung resultiert aus dem Verhalten des Schedulers, der auf der Cloud Infrastruktur für die Bereitstellung der Ressourcen verantwortlich ist. Dieser startet je nach Auslastung der Cloudknoten die Virtuellen Maschinen unterschiedlich schnell. Durch die Wahl geeigneter QoS Merkmale und unter Einbeziehung dieser in den Schedulermechanismus der Cloud Infrastruktur können die Startzeiten deutlich verkürzt werden.

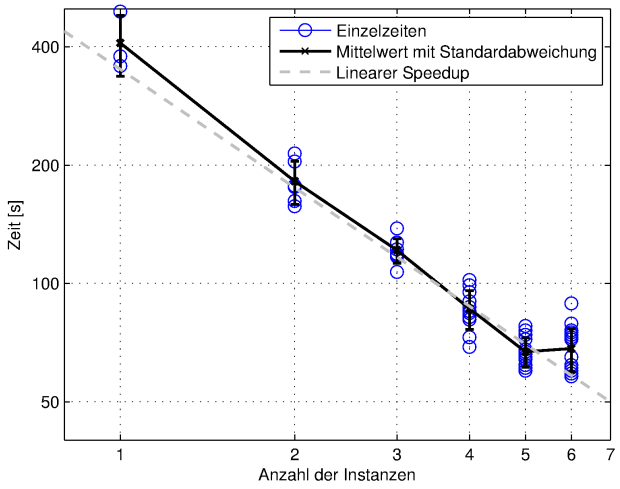


Abbildung 13: Simulationszeit in Abhängigkeit der Anzahl verwendeter Instanzen.

Abbildung 13 zeigt die Simulationszeit. Diese ist doppelt-logarithmisch aufgetragen. Zusätzlich ist ein theoretischer linearer Speedup (grau gestrichelte Linie) eingezeichnet. Es ist gut zu erkennen, dass die Simulation (im Mittel) dieser Linie folgt, das bedeutet, dass eine Verdoppelung der Instanzen tatsächlich eine Halbierung der Simulationsdauer bewirkt.

Werden allerdings zu viele Instanzen gestartet, dann tritt der Fall ein, dass der Aufwand die Simulation vorzubereiten (Einlesen des Modells, Aufstellen der Matrizen) größer ist als die zur Lösung benötigte Zeit. In diesem Fall ist durch eine Erhöhung der Zahl der Instanzen keine Beschleunigung mehr zu erreichen. Es ist jedoch anzumerken, dass das gerechnete Beispiel sehr klein war. Wie die Performance für größere Modelle aussieht, lässt sich noch nicht abschätzen.

4 Fazit

FEM-Simulationen als Dienst in der Cloud bereitzustellen bietet dem Ingenieur eine attraktive Alternative zu einer Einzelplatzlösung am eigenen Arbeitsplatz. Es konnte gezeigt werden, dass für das gerechnete Modell eine spürbare Beschleunigung der Berechnung erzielt werden konnte. Hinzu kommt die Tatsache, dass die Auslagerung der Simulation die lokalen Ressourcen des Ingenieurs entlastet. Die Portierung der FEM-Simulation als Serviceorientierte Simulation ließ sich durch den OCCI Service Adapter zudem einfach bewältigen.

Literatur

[1] Metsch, Thijs, et al. Open Cloud Computing Interface-Core. In: Open Grid Forum, OCCI-WG, Specification Document. Available at: http://forge.gridforum.org/sf/go/doc16161

[2] Metsch, Thijs, et al. Open Cloud Computing Interface-Infrastructure. In: Standards Track, no. GFD-R in The Open Grid Forum Document Series, Open Cloud Computing Interface (OCCI) Working Group, Muncie (IN). 2010

[3] Mell, Peter; GRANCE, Timothy. The NIST definition of cloud computing (draft). NIST special publication, 2011, 800. Jg., Nr. 145, S. 7.

[4] Advanced message queuing protocol: https://amqp.org

[5] Metsch, Thijs. Open Cloud Computing Interface-RESTful HTTP Rendering. 2011.

[6] rOCCI-Server: https://github.com/gwdg/rOCCI-server

[7] Schwarz, Hans R.: Methode der finiten Elemente Teubner Verlag , 3. Auflage (1991)

[8] “Cloud4E” als Projekt des Technologieprogramms “Trusted Cloud”: http://trusted-cloud.de/de/1699.php

[9] Nastran: http://www.mscsoftware.com/

[10] István Szabó: Einführung in die Technische Mechanik. Springer, Berlin 2001

Secure Algorithms for Biomedical Research in Public Clouds

Martin Beck*, V. Joachim Haupt†, Jan Moennich†, Janine Roy†, René Jäkel‡, Michael Schroeder†, Zerrin Isik†
*TU Dresden, Institute of Systems Architecture, Germany
martin.beck1@tu-dresden.de
†TU Dresden, BIOTEC, Germany
michael.schroeder@biotec.tu-dresden.de
‡TU Dresden, Center for Information Services and High Performance Computing (ZIH), Germany
rene.jaekel@tu-dresden.de

Abstract

Algorithms from the biomedical domain have to face a rapid growth of biological data and therefore a rising demand for computing time. The predictive power of such algorithms is also further improving and becomes increasingly interesting for commercial applications. Cloud Computing – as an already established paradigm to elastically allocate computing resources on demand – offers flexible solutions to deal with the increasing request for compute power. However, security concerns remain when valuable research or business data are being processed in a Public Cloud. Herein, we describe – from the application and security perspective – three biomedical case studies from different domains: Patent annotation, cancer outcome prediction, and drug target prediction developed within the GeneCloud project. Our approach is to realize a data-centric security method to be able to compute on encrypted or blinded data in any non-trustworthy environment accessible by the user.

***Index Terms*—Cloud Computing, data security, privacy, text-mining, outcome prediction, drug repositioning**

1 Introduction

Data in life sciences – such as sequence, structural, pharmacological data and biomedical literature – are ever growing and its usage demands appropriate computational resources. Cloud services offer such resources, but require adjusted algorithms and data management. Security concerns rise when valuable research, business data or personally identifiable information is transferred to a Public Cloud. Moreover, resulting data should be protected in such a way, that nobody except from the submitting party is able to evaluate the results.

Several proposals were made over the last years, like the use of trusted platform modules (TPM) for trusted computing [17], effective separation of data depending on the secrecy level [6], the complete use of fully homomorphic encryption [9] or multi-party computation between several non-colluding parties [20].

Nevertheless, these techniques possess advantages and disadvantages, that need to be considered before application. Some methods like homomorphic encryption provide a high security level, but data transformation is extremely time consuming, thus increasing execution time dramatically. In contrast, techniques – like anonymization – have only a small impact on the computational complexity, but do not provide a high level of security. The balance between different security mechanisms and the efficiency of data transformation is shown in Figure 1.

All authors contributed equally to this work.

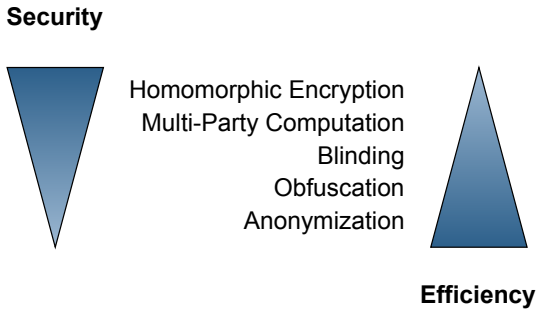


Figure 1. Balance between security and efficiency for different privacy preservation mechanisms

2 Infrastructure

In this section we briefly describe the architectural model as well as the infrastructure used to run Cloud services based on algorithms from biomedical research on a Public Cloud environment. In principal this means to operate on highly vulnerable data from the pharmaceutical domain. Therefore, any Cloud environment used for calculations not under our direct control, e.g. concerning its access rights, is regarded as a potentially non-trustworthy environment.

In our project we have started to construct a Private Cloud solution in order to realize secure versions of the use cases, which allows us to provide services able to compute on encrypted data. Firstly, we have evaluated widely used Open-Source based Cloud middlewares, which operate on top of a virtualization layer. This layer

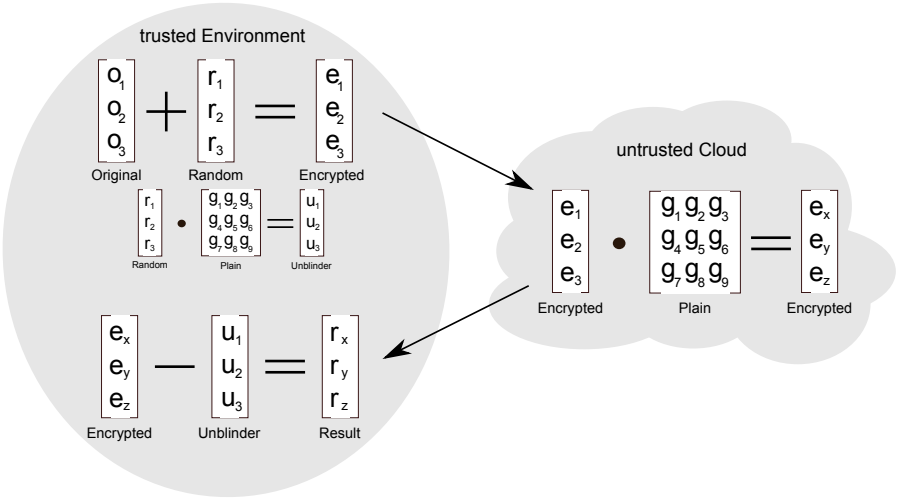


Figure 2. Privacy-preserving matrix-vector multiplication using a blinding technique to hide sensitive information

is accessible via hypervisor instances. In our reference installation we rely on KVM [11], which is a stable project and widely distributed, but we have also examined Xen [19] as a possible alternative. As middleware we have chosen OpenNebula [14] as default, but could also switch alternatively to OpenStack [15] instead. The basic management functionalities of the state-of-the-art Cloud middlewares have evolved towards a rather mature state and provide a broad set of management tools, such as user handling and rights management, full virtual machine (VM) life cycle, monitoring, as well as user and developer interfaces.

Beside the basic functions to interact with the underlying virtualized hardware layer, those middlewares (OpenNebula, OpenStack, Eucalyptus, Nimbus) support elastic scaling of needed computing demands via established quasi-standards, either by using the proprietary EC2 interface (e.g. to submit VMs to the Amazon Cloud) or the open standard OCCI [13].

We are currently evolving the use cases (see sections 4) towards secure Cloud services. The data encryption has to be performed in a fully trustworthy environment, such as our Private Cloud, where we also provide storage for those encrypted data sets. Based on the actual Cloud infrastructure, more complex Cloud models can be realized later on. In the first place, we use the system as a Private Cloud for testing purposes, but since the computing needs are potentially very large, other models, including a fully-grown Hybrid Cloud, acting than as a hybrid Cloud solution, can be realized as well among our partners.

3 Security

The bioinformatics applications, which will be described in the following chapters, utilize very different essential building blocks. Some utilized algorithms are not accessible and must be treated in a black-box setting, while others are using simpler primitives like matrix multiplications. Depending on the actual algorithm, its accessibility and of course privacy requirements regarding the input data, only a subset of available privacy preservation techniques can be applied. We will shortly discuss the three main scenarios of this project regarding possible security solutions over the next sections.

3.1 Secure Text Mining

A very basic requirement to be able to perform text mining is the possibility to compare text or character strings with each other. The input string of one member should however not be known to other participating parties. In case a search over confidential documents is performed, the content of these documents must be protected against access from unauthorized parties.

Beck *et al.* implemented a privacy-preserving string comparison algorithm, which has exactly these requirements fulfilled and is efficient by means of having a linear complexity in the length of the longest compared string [3]. The main parts of this solution are a conversion of the input strings into sets using variable length grams [12], a representation of these sets using Bloom Filters [5] and a privacy-preserving comparison of these representations using additive homomorphic encryption [4].

3.2 Secure Cancer Outcome Prediction

There are two main components of the cancer outcome prediction algorithm presented in 4.2, first a matrix-vector multiplication and second a machine learning algorithm. Several methods were proposed within the security community to securely outsource matrix multiplications to untrusted parties [1, 2]. We use an efficient blinding solution as is shown in Figure 2.

The blinding and all subsequent operations can be done over real numbers, which results in a very efficient protocol with very low computational overhead, but which leaks information about the hidden data, due to the differences in the expected and observed distributions over the input values. Instead of using floating point numbers

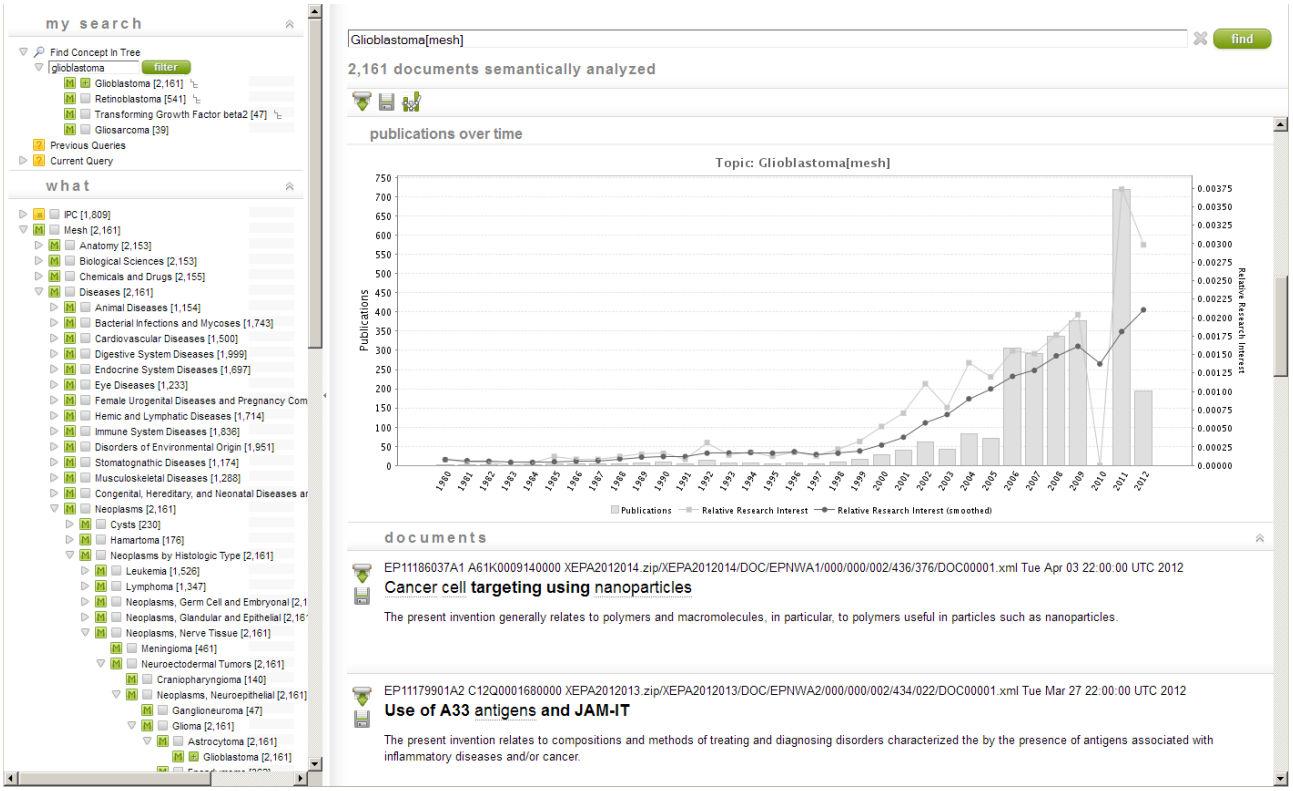


Figure 3. GoPatents: automatic extraction, sorting and statistical evaluation of biomedical entities - Left: Tree representing all assigned document to entities - Upper right: statistical evaluation of the document set - Down right: document set

we use modular arithmetic over a finite prime field $\mathbb{F}_p$. The input values are first converted to integers within this field, using an arbitrary but fixed precision. Blinding is then done by adding a uniformly chosen random element from $\mathbb{F}_p$. Another implemented solution uses an additive homomorphic cryptographic system to encrypt either the matrix or the vector of the multiplication. The actual operation is then performed over the ciphertext, returning an encrypted result, which only the client is able to decrypt.

3.3 Secure Drug Target Prediction

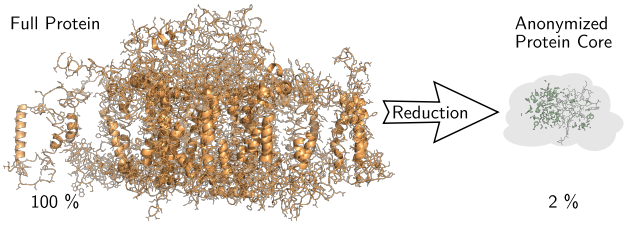


Figure 4. Minimizing and at the same time anonymization of a protein structure to a predefined ligand

Some of the algorithms used for predicting drug targets are not available as source code and thus can only be treated as a black box, which can not be converted directly into a privacy-preserving version. One part of the prediction pipeline, where we find such a black box, is the task to perform a binding site alignment of two protein structures in three dimensional space. The inputs of such an algorithm are descriptions of two protein structures, which contain mainly coordinates with three dimensions, some properties for the referenced atoms and further general descriptions. We strip all the unnecessary information out of the structure to make identification of compared proteins as hard as possible.

Figure 4 shows the result of such a minimization. Only atoms along and around the specified ligand are kept. The atom positions them self are further moved and rotated by some random value to make identification more complex.

4 Use Cases

In this section we describe three domains of biomedical research using big data. This data is at least partially mission-critical, underlies copyright restrictions or contains personally identifiable information and must therefore be protected against illegal access. The large amount of required processing resources, as well as the necessity for privacy protection thus builds a common ground for all of the following use cases.

4.1 Text Mining

The text mining algorithms will be implemented as two use cases.

The first use case is the creation of GoPatents (Figure 3), which automatically analyzes European patents for biomedical entities and sorts them into the IPC ontology. In contrast to the search engine by Doms *et al.* GoPubMed [7] it does not only analyze the abstracts, but

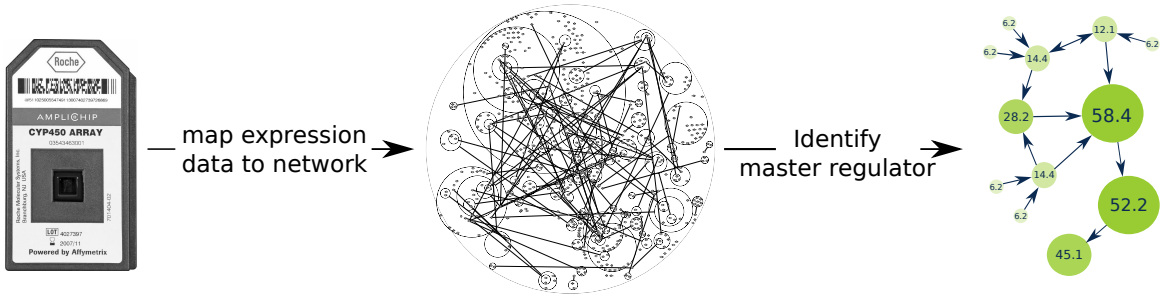


Figure 5. The work flow of network-based biomarker discovery

whole documents. Due to the longer texts, much more processing power is required (approx. 6,000 CPU h for 2 million patents instead of 300 CPU h for 20 million abstracts), hence it requires massive parallel processing to analyze these patents within a rather short time frame. Further the patents have to be protected due to copyright restrictions.

The second use case processes the relations between different entities, such as proteins, drugs and diseases. However, this information is usually not easily accessible from a database, it has to be collected from many different journals. We have developed a system that retrieves a list of statements about the relation between two different entities of the given drugs, proteins, and diseases. After a search for two entities, all documents are automatically fetched and all sentences with both entities are extracted. These sentences are ranked according to their importance using a maximum entropy classifier. While such analysis for combinations of drugs, proteins or diseases is already available, we are developing an online application that identifies all counterparts for one known entity. As these calculations are computationally intensive (approx. 3 CPU months for 22 million Medline abstracts), access to a Cloud environment can shorten the overall execution time considerably by parallel processing of search tasks. Currently, a demonstrator for a few thousand abstracts is available. However, we need to adjust this application for the Cloud environment to run the analysis for the whole Medline database.

4.2 Cancer Outcome Prediction

Cancer outcome prediction aims to address the problem of learning how to forecast disease progression from gene expression and thus allowing refined treatment options. Thus, gene expression measured via microarrays helps to reveal underlying biological mechanisms of tumor progression, metastasis, and drug-resistance in cancer studies. The gene signatures obtained in such analysis could be addressed as biomarkers for cancer progression.

The experimental or computational noise in data and limited tissue samples collected from patients might reduce the predictive power and biological interpretation of such signature genes. Network information (e.g. about protein-protein interactions) efficiently helps to improve outcome prediction and reduce noise in microarray experiments [8]. For this purpose, network information has been integrated with microarray data in various studies in the last decade.

Winter *et al.* developed an algorithm – called NetRank – that employs protein-protein interaction networks and ranks genes by using the random surfer model of Google's PageRank algorithm [18]. Figure 5 shows the general approach of NetRank. Firstly, gene expression values measured by microarrays are mapped to network data, which might be either physical protein-protein interaction or regulatory information. Afterward, NetRank is applied on the network to identify master regulators based on gene expression data and network information. Hence, the algorithm provides an integration of the topological information (i.e. connectivity and random walk) and microarray data (i.e. node score) to explore crucial signature genes for outcome prediction.

The performance of the algorithm was evaluated in two studies. In the first study, NetRank was applied on gene expression data obtained from 30 pancreas cancer patients and seven candidate biomarker genes could be identified that are able to predict the survival time of patients after tumor removal with 70% accuracy [18]. The second study was a systematic assessment of network-based outcome prediction on 25 cancer data sets, where the authors could show the general applicability of the algorithm on different cancer types [16].

Due to the effectiveness of the approach, implementation in the Cloud as a publicly available software could accelerate as well as simplify cancer outcome prediction.

4.3 Drug Target Prediction from Binding Site Similarity

Drug repositioning applies established drugs to new disease indications with increasing success. A prerequisite for drug repositioning is drug promiscuity (polypharmacology) – a drug's ability to bind to several targets. There is a long standing debate on the reasons for drug promiscuity. Based on large compound screens, hydrophobicity and molecular weight have been suggested as key reasons. However, the results are sometimes contradictory and leave space for further analysis. Protein structures offer a structural dimension to explain promiscuity: Can a drug bind multiple targets because the drug is flexible or because the targets are structurally similar or even share similar binding sites?

Haupt *et al.* contributed to the discussion of causes of

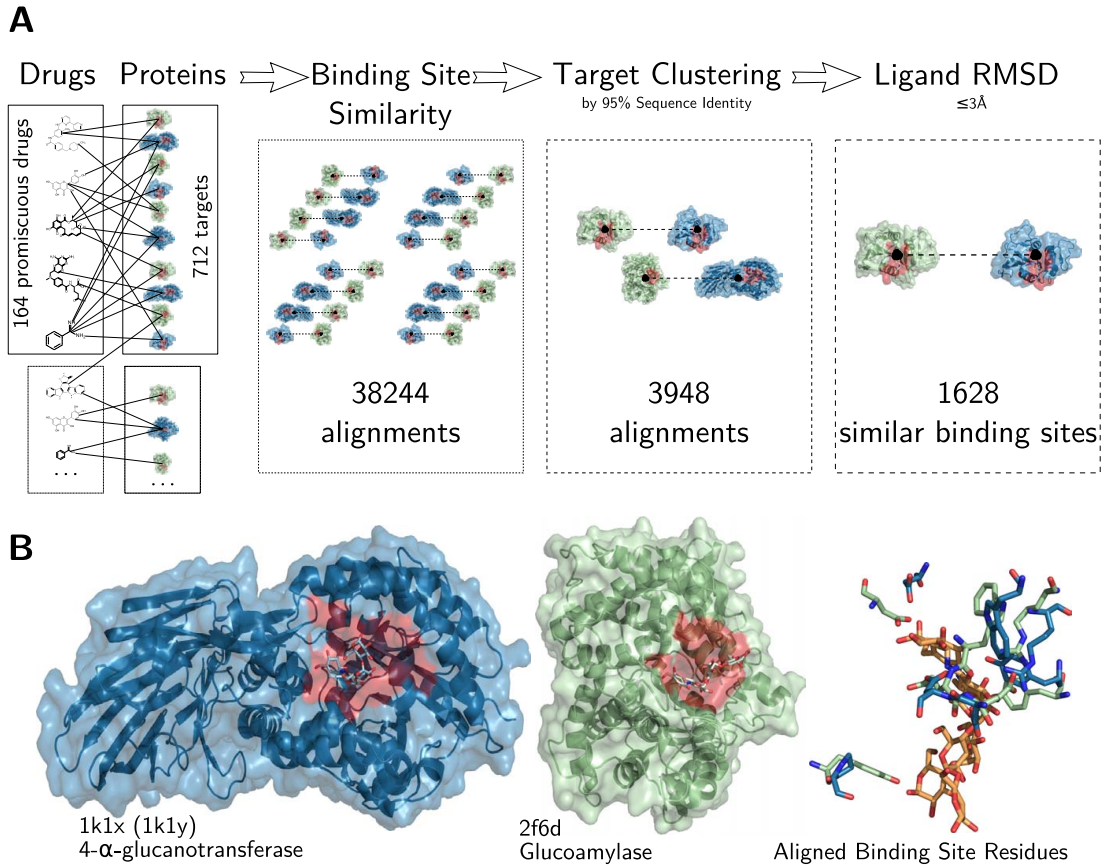


Figure 6. (A) The work flow resulting in 1628 similar binding site pairs with well superposed ligands. (B) Similar binding sites in a pair of proteins targeted by acarbose. The two bound structures of acarbose are shown as orange sticks on the right with superposed binding site residues. Figure adapted from [10]

drug promiscuity by pursuing a comprehensive structural approach and by investigating two more possible sources for drug promiscuity: ligand flexibility and binding site similarity [10]. However, the drug data set is limited in size due to the restriction to PDB. They analyzed a structural data set of 164 promiscuous drugs bound to 712 unique protein targets from the PDB (Figure 6.A). Regarding drug physicochemical properties, they found no correlation between hydrophobicity or molecular weight and the degree of promiscuity of a drug in contrast to other studies on large screening libraries. Subsequently, they clustered the drug conformers and compared all binding sites of a drug. As a result, they found a weak correlation of the degree of drug promiscuity to ligand flexibility (r = 0.2), a correlation to structural similarity (r = 0.76) and even higher to the number of similar binding sites (r = 0.81, example in Figure 6.B). Furthermore, they found that for 71 % of the drugs at least one pair of their targets' binding sites is similar and for 22 % all are similar. Thus, they conclude that binding site similarity is the most important prerequisite for a promiscuous PDB drug to bind to multiple PDB targets and that ligand flexibility has a minor impact. Molecular weight and hydrophobicity do not seem to influence whether a drug is promiscuous or not. Global structural similarity is also reflected in the pairs of similar binding sites but misses the important examples of similar binding sites in globally structurally dissimilar proteins. In particular, 15 % of all target pairs with a similar binding site are dissimilar in global structure and would have not been detected by other approaches on sequence or global structure level. As supported by their findings, protein local structural alignments bare a huge potential to infer so-far unknown drug-target relationships. Implementation in the Cloud might speed up drug development by uncovering off-targets and thus causes of adverse drug reactions early in the development pipeline. On the other Hand, predicted targets are starting points for drug repositioning.

5 Conclusions

The different biomedical applications described are designed as services for a Cloud environment. The infrastructure to support these services was chosen to fit the requirements given by the applications. Further the used Cloud middleware is solely based on open standards and capable of easy deployment of virtual machines to a local cluster or other Cloud providers. The infrastructure itself is not obliged to provide security and privacy for the private data.

By choosing and designing algorithms that preserve privacy of the input data through mechanisms like blinding, homomorphic encryption and anonymization, the desired security levels can be achieved without the necessity

to trust the Cloud provider or any other third party. For those algorithms, which could not be changed due to license restrictions, we applied a black box method and minimized the input and output information to the absolute necessary amount.

Further, as the proposed security solutions are directly targeting the developed algorithms and are therefore independent of the actual Cloud infrastructure, the services can run on virtually any Cloud platform. This allows more flexibility, interoperability and efficiency compared to platform dependent solutions. The ideas, methods and tools used to construct these privacy-preserving solutions can easily be adapted to other similar algorithms. As a result, the security risks in using public Cloud Computing upon private data decreases and new applications might arise.

6 Acknowledgments and Funding

Funding by the German Federal Ministry of Economics and Technology is kindly acknowledged as GeneCloud is part of the Trusted Cloud Initiative. Furthermore, the authors would like to thank the Center for Information Services and High Performance Computing (ZIH) at TU Dresden for generous allocations of computing time.

References

[1] Atallah, M.; Pantazopoulos, K.; Rice, J.: Secure outsourcing of scientific computations „Advances in Computers 54“, 2001.

[2] Atallah, M.; Frikken, K.: Securely outsourcing linear algebra computations „Proceedings of the 5th ACM Symposium on Information, Computer and Communications Security - ASIACCS ’10“, New York, April 2010.

[3] Beck, M.; Kerschbaum, F.: Approximate Two-Party Privacy-Preserving String Matching with Linear Complexity „Proceedings of the 2nd IEEE International Congress on Big Data (BIGDATA)“, July 2013.

[4] Boneh, D.; Goh, E.; Nissim, K.: Evaluating 2-DNF formulas on ciphertexts „Theory of Cryptography (TCC) ’05“, 2005.

[5] Bloom, B.: Space/time trade-offs in hash coding with allowable errors „Communications of the ACM“, July 1970.

[6] Bugiel, S.; Nürnberger, S.; Sadeghi, A. *et al.*: Twin Clouds: Secure Cloud Computing with Low Latency. „12th Communications and Multimedia Security Conference (CMS’11) “, 2011.

[7] Doms, Andreas and Schroeder, Michael: GoPubMed: exploring PubMed with the Gene Ontology. „Nucleic Acids Res“, July 2005.

[8] Fortney, K.; Jurisica, I.: Integrative computational biology for cancer research. „Journal of Human Genetics“, October 2011.

[9] Gentry, C.: Fully homomorphic encryption using ideal lattices „Proceedings of the 41st annual ACM symposium on Theory of computing“, New York, 2009.

[10] Haupt, V.J.; Daminelli, S.; Schroeder, M.: Drug Promiscuity in PDB: Protein Binding Site Similarity Is Key. „PloS one“, January 2013.

[11] Kernel Based Virtual Machine, http://www.linux-kvm.org

[12] Li, C.; Wang, B.; Yang, X.: VGRAM: improving performance of approximate queries on string collections using variable-length grams „Proceedings of the 33rd international conference on Very large data bases (VLDB)“, September 2007.

[13] Open Grid Forum Working Group, „OCCI – Open Cloud Computing Interface“, 2009.

[14] Open Source Data Center Virtualization, information online: http://opennebula.org

[15] Open Stack Cloud Software, online via: http://openstack.org

[16] Roy, J.; Winter, C.; Isik, Z. *et al.*: Network information improves cancer outcome prediction. „Briefings in Bioinformatics“, December 2012.

[17] Santos, N.; Gummadi, K.; Rodrigues, R.: Towards trusted cloud computing „Proceedings of the 2009 conference on Hot topics in cloud computing“, Berkeley, 2009.

[18] Winter, C.; Kristiansen, G.; Kersting, S. *et al.*: Google goes cancer: improving outcome prediction for cancer patients by network-based ranking of marker genes. „PLOS Computational Biology“, 2012.

[19] Xen Project: http://xenproject.org

[20] Yao, A.: How to generate and exchange secrets „27th Annual Symposium on Foundations of Computer Science“, 1986.

Cloud4health – On effective ways to deal with sensitive patient data in a secure Cloud environment

Steffen Claus[1], Horst Schwichtenberg[1], Julian Laufer[2], Florian Berger[2]

[1]Fraunhofer SCAI, Schloss Birlinghoven, 53754 Sankt Augustin
[2]RHÖN-KLINIKUM AG, Schlossplatz 1, 97616 Bad Neustadt a. d. Saale

{steffen.claus, horst.schwichtenberg}@scai.fraunhofer.de
{julian.laufer, florian.berger}@rhoen-klinikum-ag.com

Abstract

The cloud4health project researches secondary analysis of clinical patient data, such as surgery- and discharge-reports in a secure and trusted Cloud infrastructure. Given the data's sensitive nature, a main emphasis rests on guaranteeing its confidentiality during the course of the analysis. The paper outlines infrastructure developments of the first year of the cloud4health project and highlights requirements towards a secure Cloud environment. The first solution architecture is sketched and the lifecycle of data processing is presented.

1 Introduction

During the course of a patient's treatment in clinics, several documents emerge along the way; from initial diagnostics to surgery- and discharge reports. Partly, this documentation consists of unstructured information, i.e. free text, whose style and semantic clearness varies widely between the respective responsible doctors. Utilizing such patient data for so-called secondary usage scenarios, e.g. for retrospective studies on drugs' side effects or effectiveness of certain surgery methods, exhibits a great potential but requires sophisticated text mining tools. Depending on the size of the study, respectively the number of relevant documents, increasing requirements towards processing power and memory cannot be met by single clinic infrastructures. Utilizing Cloud computing resources for these scenarios offers a promising approach but guaranteeing the patient data's confidentiality throughout the whole lifecycle of data processing represents a challenging task.

1.1 The cloud4health project

The cloud4health (c4h) project is funded by the German Federal Ministry of Economics and Technology in the funding program "Trusted Cloud" (FKZ 01MD11009) and researches secure and trusted secondary use of clinical patient data in a Cloud infrastructure. The project demonstrates practical relevance by concentrating on real-world use cases. For example, cloud4health analyzes narrative surgery reports in implantations to extract information about the type of prosthesis and the kind of operation technique. Thus, it allows statistical analysis on the effectiveness and durability of implants.

To support such secondary usage scenarios, the project develops several Cloud services. The main service – the text mining - applies Natural Language Processing (NLP) techniques for analyzing clinical patient data, more specifically the contained free text. This service utilizes study-specific medical terminologies provided by a terminology management service. Analyzed - i.e. semantically annotated - patient data can be further processed through data mining services. These Cloud services can be offered as Software-as-a-Service (SaaS). Depending on the considered secondary usage scenarios or specific clinic requirements, different deployment scenarios are envisaged and supported.

In this context, the project proposes three solution architectures. In the first solution, patient data is pre-processed and anonymized before it is sent to the text mining services. This pre-processing step removes personal-identifiable attributes from the patient data, thereby transforming it into so-called de facto anonymized data. The second solution applies clinic-internal pseudonymization which allows tracing patient's documents after the analysis in the Cloud. The third solution enables across-clinics pseudonymization. Figure 1 shows an overview of the implementation of the first solution and sketches its three central building blocks: the hospital infrastructure, the text mining Cloud and the study portal. A more detailed description of the individual components is found in section 3.

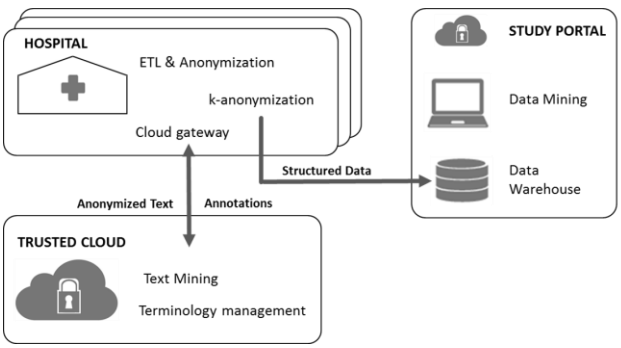


Figure 1 cloud4health (c4h) architecture

1.2 Related Work

There are several projects in the area of secondary use of clinical data, some of them making use of NLP techniques [1], [2], [3]. Some undertakings are focused on specific use cases [4]; others support a wide range of potential research scenarios [5]. Secondary use of clinical data within the German regulatory framework is researched by [6]; special focus on privacy enhancing tools is laid by [7].

In the area of collaborative use and management of Electronic Health Records (EHR) initiatives such as [8] try to facilitate patient treatment across clinics, while [9] concentrates on underlying security aspects of such processes. A clinical data management system with a strong emphasis on data protection and security measures is sketched in [10]. Challenges introduced by legal and ethical aspects - especially in multinational clinical data management – are discussed in [11].

Projects such as [12] and [13] research secure ecosystems for use cases within the health sector. Other undertakings concentrate on fundamental principles of securing clinical data in Cloud usage scenarios [14], [15]. To secure clinical patient data before delivering them to secondary analysis workflows in external infrastructures, one approach is to pre-process and remove patient identifying information [16]. Potential re-identification risk of such pre-processed data is researched by [17].

On the one hand, most of the aforementioned projects don't apply to the German data protection framework, including its diverse regional regulations and hospital laws. On the other hand, some of the initiatives focus on German regulations, but don't incorporate challenges that arise through the use of Cloud computing. Security measures are often only applied for a specific use case or only fit to a certain part of the whole ecosystem, including clinic infrastructures, public transport networks and clinic-external Cloud infrastructures. Summing up, there is no integrated solution for a secure and trusted secondary use of clinical data in Cloud infrastructures that is explicitly focused on the German regulatory framework and includes all relevant players – clinics, data protection officers, legal experts and Cloud providers. Cloud4health approaches to close this gap.

2 Secure Cloud environment

The compute- and memory-intensive processing of patient – i.e. the text mining - is handled by a Cloud testbed infrastructure provided by the Fraunhofer institute SCAI. This testbed is used by the partners and clinics within cloud4health and thus represents a community Cloud. It constitutes a potential deployment model for the Cloud services and can furthermore serve as a best practice implementation for processing patient data in a Cloud in a secure and trustworthy manner. Given the sensitive nature of patient data, the Cloud provider should meet high security requirements regarding the data's confidentiality. As clinics are the essential data providers and users of the Cloud services, security measures not only have to be aligned to applicable data protection regulations, such as the BDSG (German Federal Data Protection Act) or the diverse LDSGs (Regional Data Protection Acts), but also to hospital laws and clinics' specific regulations and guidelines for data processing in external infrastructures. On that basis and in close cooperation with the clinics and legal experts within cloud4health, fundamental requirements for secure and trusted patient data processing in an external Cloud have been gathered. These aspects – outlined in section 2.1 – are incorporated into the current operated testbed infrastructure.

2.1 Fundamental security requirements

Only in case of personal data the aforementioned legislative regulations – e.g. BDSG and LDSG - apply. But under certain conditions, also non-personal and de facto anonymized data can become personal data again, e.g. through inside knowledge (keyword: k-anonymity). Thus, the Cloud provider should implement appropriate security measures in order to protect the data's confidentiality and thereby reducing the risk of data re-identification.

As a first step, the Cloud provider should perform a detailed risk analysis of the respective infrastructure and its components. This analysis should specify and examine diverse worst-case scenarios regarding breaches of data confidentiality and has to group the infrastructure components into different categories related to their protection needs. Based on this classification, the provider can implement technical security measures accordingly. Guidelines from the working groups of the German federal data protection officers [18], [19], the procedures from the German Federal Office for Information Security (BSI) [20], [21], [22] or the recommendations from the European Network and Information Security Agency (ENISA) [23] are advised to be followed. In the course of the analysis, it is beneficial to directly construct an information security management system, e.g. by following the BSI standard 100-2 [24], which is closely related to the well-known ISO standards 27001 [25] and 27002 [26].

Furthermore, the concrete purpose of data processing not only has to be explicitly defined but also maintained appropriately. Thus, security measures must guarantee that personal patient data won't be used for any other purpose than the predetermined one. Therefore, unauthorized users – respectively their processes - mustn't access other users' data; multi-tenancy has to be supported and separation of data processing has to be ensured on all levels during the complete lifecycle of data handling - from transfer to storage over processing to deletion. Ideally, personal patient data should not be stored in the Cloud infrastructure at all. At least, it should be deleted after a reasonable or agreed upon period of time. Furthermore, an incident response process is required, preferably integrated in the aforementioned overarching concept for security management. Finally, the user of the Cloud infrastructure has to be able to check and verify the security measures, e.g. through a certification executed by trusted third party experts. Even though there are standards for

self-assessment and self-certification, an independent verification of security measures is always preferred.

2.2 Technical security measures in the Cloud

The project performed an in-depth risk analysis of all components of the Cloud testbed following abovementioned state-of-the-art techniques [18], [19], [20], [21], [22], and [23]. This analysis was oriented at the procedures as described by the BSI in its standard "100-2: IT-Grundschutz-Vorgehensweise". Given its broad view on potential risks and accompanied suggestions for mitigating measures, data confidentiality is ensured throughout the whole workflow of data processing in the Cloud.

The following list shows an excerpt of the fundamental security measures applied in the Cloud testbed:

- **Secure data transfer over public networks:** Confidentiality of the data transfer from the clinic to the virtual machines hosting the text mining services is guaranteed by encryption (through OpenVPN, [27]). The selection of cipher suites and the definition of key lengths closely follow the respective BSI guidelines [21], [22].
- **Exclusive text mining services for each user:** The virtual machines hosting the text mining services are directly initialized by the user. Only he maintains control over the virtual machine, other users aren't authorized to access any of the contained services. Exclusive allocation of Cloud nodes is supported as well to further strengthen the separation of different Cloud users.
- **Limited lifetime of virtual machines:** Virtual machines are immediately terminated once the data processing has finished. Thus, patient data only resides in the Cloud as long as it is needed for the respective text mining processes.
- **No persistence of personal patient data:** All data generated during the document processing, such as log files and temporary files, is discarded during the shutdown of the respective virtual machine. No patient data or any data related to the input documents is stored in the image of the virtual machine or anywhere else in the Cloud infrastructure, e.g. in an object storage or a shared file system.
- **Secure virtual machine image storage:** The basic images for the virtual machines are kept in a secure central storage. Once an image is requested for start-up, it is copied to the respective execution node via an encrypted channel (SCP). Depending on the users' preferences, encryption techniques and key lengths can be adjusted accordingly. The basic images serve as templates and don't contain any sensitive patient data or other information related to the respective user.
- **Separation of Cloud-internal communication:** The text mining service is distributed and scalable: multiple virtual machines containing the text mining services can be started, the services themselves support multithreading. Thus, patient data processing can potentially be spread over multiple Cloud nodes. The required communication between those nodes, respectively the virtual machines, is supported by dynamically set-up VLANs. Each user gets its own VLAN, which maintains a separation of the Cloud-internal data transfer.

With these security measures integrated in the overarching security concept, cloud4health addresses the requirements and challenges gathered in section 2.1. Even though the Cloud testbed hasn't been certified yet, the carried out steps support the preparation for certification processes based on e.g. EuroPriSe [28] or [24].

3 Life cycle of data processing

Patient data processing starts within the clinic's infrastructure (see overview in Figure 1). Data from clinic-internal systems such as hospital information systems or data warehouses is extracted and harmonized. Subsequently, this patient data is anonymized. This step includes the replacement of certain identifying attributes, such as names, dates of birth and post codes in unstructured text as well as in the document's meta-data. Preprocessed documents are then stored in a so-called transfer database. The transfer database enables clinical data protection officers or another personal to control and verify the data set that is intended for analysis in the Cloud. All these abovementioned steps – extraction, harmonization, anonymization and storing - are integrated into an ETL workflow (Extract, Transform, Load). A more detailed description of this workflow can be found in [29].

Figure 2 shows the sequence of data processing in the Cloud. The clinic's gateway to the Cloud is embodied by a component that maintains an encrypted connection to the text mining services in the Cloud. As well, this gateway temporarily assigns random IDs to the documents before sending them to the text mining services. Thus, documents inside the Cloud cannot be traced back to the underlying patient. Before the data processing can be started, the Cloud gateway initializes and configures the required virtual machines and their integrated text mining services via a Cloud management interface. The initialization is controllable by parameters such as the number of documents or the document type, resulting in an adjusted number of started virtual machines or different text mining services. Within the Cloud, documents are processed and annotated based on study-specific terminologies. For example, documents within a hip surgery follow-up study would be searched for occurrences of types of hip joint prostheses, applied surgery methods or other medical terminology e.g. describing the initial bone structure of the patient. These terminologies are dynamically provided by the terminology management service.
Once all documents have been processed, annotated and transferred back to the hospital, the Cloud gateway termi-

nates all virtual machines. No temporal or any other patient-related data is persisted in the Cloud. Before the processed documents are uploaded to the study portal for further data mining and study-specific processes, the data's confidentiality can be further improved by computing k-anonymous sets of the input documents.

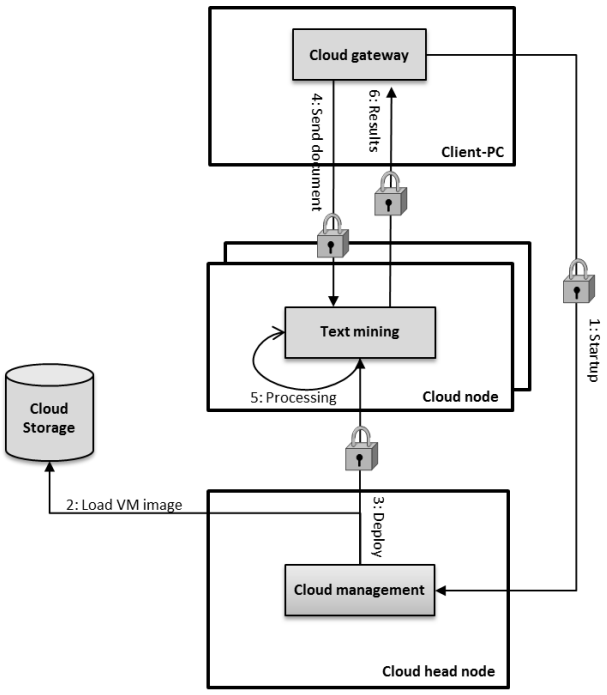


Figure 2 Sequence of data processing in the Cloud

4 Conclusion and future work

The cloud4health project successfully implemented a prototype of the first solution architecture in March 2013. Based on this initial architecture, further use-cases will be realized. The initialization procedure is extended to support dynamic up- and down-scaling of virtual machines during runtime. Load tests with hundreds of thousands of patient documents will be used to benchmark this solution. With the integrated approach and the first prototypical architecture, cloud4health demonstrated the feasibility of processing sensitive patient data in a Cloud in a secure and trusted manner. Data protection and security concepts have been composed in close cooperation with clinical data protection officers and legal experts and thus serve as best practices for similar projects.

The project cloud4health is funded by the German Federal Ministry of Economics and Technology in the funding program "Trusted Cloud" (FKZ 01MD11009).

5 References

[1] Chard, K.; Russell, M.; Lussier, Y.; A; Mendonça, E. A; Silverstein, J. C.: A Cloud-based Approach to Medical NLP. AMIA Annual Symposium proceedings, 2011, S. 207–216.

[2] Carrell, D.: A Strategy for Deploying Secure Cloud-Based Natural Language Processing Systems for Applied Research Involving Clinical Text. Proceedings of the 44th Hawaii International Conference on System Sciences, IEEE Computer Society, 2011, S. 1-11.

[3] Chute, C.; Pathak, J.; Savova, G.: The SHARPn project on secondary use of Electronic Medical Record data: progress, plans, and possibilities. AMIA Annual Symposium Proceedings, 2011, S. 248–256.

[4] Weber, G. M.; Murphy, S. N.; McMurry, A. J.; Macfadden, D.; Nigrin, D. J.; Churchill S.; Kohane, I. S.: The Shared Health Research Information Network (SHRINE): a prototype federated query tool for clinical data repositories. Journal of the American Medical Informatics Association 16(5), 2009, S. 624–630.

[5] EHR4CR Consortium: Electronic Health Records for Clinical Research, http://www.ehr4cr.eu, last visited 10.06.2013.

[6] Berliner Forschungsplattform Gesundheit, http://medinfo.charite.de/forschung/berliner_forschungsplattform_gesundheit/, last visited 21.06.2013.

[7] Ganslandt, T.; Mate, S.; Helbing, K.; Sax, U.; Prokosch, H. U.: Unlocking data for clinical research - The German i2b2 experience, Applied Clinical Informatics 1(4), 2011, S. 116-127.

[8] Elektronische Fallakte, http://www.fallakte.de/, last visited 21.06.2013.

[9] Hupperich, T.; Löhr, H.: Flexible patient-controlled security for electronic health records. International Health Informatics Symposium, 2012, S. 1–5.

[10] Deserno, T. M.; Deserno, V.; Lowitsch, V.; Franck, W.; Willems, J.; Löbner, H.: Aspekte des datenschutzgerechten Managements klinischer Forschungsdaten, GI-Jahrestagung, 2012, S. 1491-1505.

[11] Rahmouni, H. B.; Solomonides, T.; Mont, M. C.; Shiu, S.: Privacy compliance and enforcement on European healthgrids: an approach through ontology, Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences 368(1926), 2010, S. 4057-4072.

[12] Cloudi/o – Sicheres Cloud-basiertes Datenmanagement im Umfeld der klinischen Forschung, http://www.cloudi-o.de/, last visited 21.06.2013.

[13] TRESOR – Trusted Ecosystem for Standardized and Open cloud-based Resources, http://www.cloud-tresor.com/about-tresor/, last visited 21.06.2013.

[14] Neuhaus, C., Wierschke, R., Löwis, M. von, Polze, A.: Secure Cloud-based Medical Data Exchange, GI-Jahrestagung, 2011

[15] Li, M.; Yu, S.; Zheng, Y.; Ren, K.; Lou, W.: Scalable and secure sharing of personal health records in cloud computing using attribute-based encryption, IEEE Transactions on parallel and distributed systems, *24*(1), 2013, S. 131–143.

[16] Pommerening, K.; Helbing, K.; Ganslandt, T.; Drepper, J.: Identitätsmanagement für Patienten in medizinischen Forschungsverbünden, GI-Jahrestagung, 2012, S. 1520–1529.

[17] Dankar, F. K.; El Emam, K.; Neisa, A.; Roffey, T.: Estimating the re-identification risk of clinical data sets. BMC medical informatics and decision making, 2012, 12(1), 66.

[18] Arbeitskreis Technik und Medien: Orientierungshilfe – Cloud Computing, 2011.

[19] Arbeitskreis Technische und Organisatorische Datenschutzfragen: Technische und organisatorische Anforderungen an die Trennung von automatisierten Verfahren bei der Benutzung einer gemeinsamen Infrastruktur, 2012.

[20] Bundesamt für Sicherheit in der Informationstechnik: Eckpunktepapier - Sicherheitsempfehlungen für Cloud Computing Anbieter, Bonn, 2011.

[21] Bundesamt für Sicherheit in der Informationstechnik: Kryptographische Verfahren: Empfehlungen und Schlüssellängen, Bonn, 2013.
[22] Bundesamt für Sicherheit in der Informationstechnik: Kryptographische Verfahren: Empfehlungen und Schlüssellängen – Verwendung von TLS, Bonn, 2013.
[23] ENISA: Cloud Computing - Information Assurance Framework, 2009.
[24] Bundesamt für Sicherheit in der Informationstechnik: BSI-Standard 100-2, IT-Grundschutz-Vorgehensweise, Version 2.0, Bonn, 2008.
[25] ISO/IEC 27001:2005 - Information technology – Security techniques – Information security management systems requirements specification, 2005.
[26] ISO/IEC 27002:2005 - Information technology – Code of practice for information security management, 2005.
[27] OpenVPN, Version 2.3.1, 03/2013.
[28] Unabhängiges Landeszentrum für Datenschutz Schleswig-Holstein: EuroPriSe Criteria, Kiel, 2011.
[29] Griebel, L.; Leb, I.; Christoph, J.; Laufer, J.; Marquardt, K.; Prokosch, H.U.; Toddenroth, D.; Sedlmayr, M.: Cloud-Architektur für die datenschutzkonforme Sekundärnutzung strukturierter und freitextlicher Daten, Proceedings of the eHealth2013, 2013, S. 59-64.

FPGAs in der Cloud: Integration und Bereitstellung von rekonfigurierbaren Hardware-Ressourcen in einer Cloud-Infrastruktur

Oliver Knodel, Rainer G. Spallek
Institut für Technische Informatik, Technische Universität Dresden, Dresden, Germany
oliver.knodel@tu-dresden.de

Kurzfassung

Cloud-Computing findet eine immer weitere Verbreitung und hat mittlerweile eine große wirtschaftliche Bedeutung. Durch die flexible Bereitstellung von Ressourcen und Diensten kann eine deutliche Kostenersparnis auf Nutzerseite erreicht werden. Die Einsatzgebiete reichen hierbei von einfachen Web-Technologien und Datenspeichern über komplexe Geschäftsprozesse bis hin zu datenintensiven wissenschaftlichen Anwendungen.
Auch im Bereich von Systementwurf und -analyse gewinnt die Auslagerung komplexer Synthese- und Simulationsprozesse in eine Cloud zunehmend an Bedeutung. Insbesondere beim Entwurf von Anwendungen für die immer größer werdenden programmierbaren Schaltkreise werden leistungsfähige Synthese- und Simulationssysteme benötigt. Neben der einfachen Auslagerung von Synthese und Simulation ist in vielen Fällen auch der Test auf einer realen Hardware, wie einem FPGA, von großer Bedeutung.
Die Investitionskosten für FPGAs als Plattform für Prototypen, welche zum Teil nur über einen kurzen Zeitraum genutzt werden, werden durch eine Integration dieser Komponenten in eine Cloud vermieden. Dieser Beitrag erläutert, wie rekonfigurierbare Schaltkreise in eine Cloud-Infrastruktur eingebettet werden können, um die Ressource FPGA als Service bereitstellen zu können und somit den gesamten Prozessablauf von der Synthese über die Simulation bis hin zum Test auf der realen Hardware als on-demand Dienst anzubieten. Darauf aufbauend soll es ebenfalls möglich sein, den FPGA als Hardwarebeschleuniger einzusetzen. Eine flexible Einbettung des FPGAs in die Architektur ist daher wesentlicher Bestandteil dieses Beitrages.

1 Einleitung

Auf vielen Gebieten wird zunehmend auf Cloud-Dienste zurückgegriffen. Da eine Cloud-Infrastruktur auf einfachem Wege und jederzeit einen Zugang zu den aktuell benötigten Ressourcen bereitstellt, können die Investitionskosten in eigene IT-Systeme signifikant reduziert werden. Durch die Umsetzung des Utility Computing [1], bei dem die jeweils aktuell benötigte Menge an Ressourcen bereitgestellt und abgerechnet wird, ist das Cloud-Computing von großer wirtschaftlicher Bedeutung für viele kleine und mittelständische Unternehmen, aber ebenso für Forschungseinrichtungen [2].

Im Bereich von Systementwurf und -analyse ist die Auslagerung komplexer Synthese- und Simulationsprozesse in eine Cloud von zunehmender Bedeutung. Die Arbeit mit erheblichen Datenmengen und einer Vielzahl von Parametern oder die Untersuchung unterschiedlicher Varianten erfordert, dass Prozessor- und Speicherressourcen praktisch unbegrenzt zur Verfügung stehen, ohne dass die Investitionskosten in die dafür erforderliche Hardware beim eigentlichen Nutzer liegen.

Neben einer einfachen Auslagerung von Synthese und Simulation ist auch der Test auf der realen Hardware von großer Bedeutung. Daher ist es naheliegend diesen Test auf rekonfigurierbaren Schaltkreisen wie Field-Programmable Gate Arrays (FPGAs), ebenfalls in die Cloud auszulagern. Beim Testen von Prototypen auf realer Hardware entstehen erhebliche Investitionskosten, obwohl die Hardware nur für einen begrenzten Zeitraum benötigt wird. Durch eine Integration von FPGAs in eine Cloud können dem Nutzer die benötigten Ressourcen in beliebiger Größe und Menge bereitgestellt werden, ohne dass dieser in neue Hardware investieren muss.

Der Entwurf eines System-on-Chip (SoC) oder eines vollständigen Prozessors und dessen anschließender Test auf realer Hardware mit unterschiedlichen Parametern erfordert viel Zeit, wenn nur eine einzelne Testplattform zur Verfügung steht. Da mittels einer Cloud Ressourcen in deutlich größerem Umfang angeboten werden können, ist eine Parallelisierung dieser Vorgänge und damit eine erhebliche Zeitersparnis möglich. Die Anschaffung dieser Menge an Hardware für einen einmaligen Test hingegen ist wirtschaftlich oft nicht vertretbar. Darauf aufbauend ist es möglich, den FPGA auch als Hardwarebeschleuniger für eine Vielzahl von Anwendungen einzusetzen. Die Einbettung des FPGAs in eine sinnvolle Infrastruktur ist ein weiterer wesentlicher Bestandteil dieses Beitrages.

In Abschnitt 2 diese Beitrages wird der aktuelle Einsatz von Hardwarebeschleunigern und FPGAs in Computing-Clouds beschrieben. Darauf aufbauend wird in Abschnitt 3 die angestrebte Architektur vorgestellt, bevor in Abschnitt 4 verschiedene Einsatzgebiete für diese Cloud-Architektur erläutert werden. Abschnitt 5 gibt einen Ausblick.

2 Überblick – Clouds und FPGAs

Das Auslagern von Synthese- und Simulationsprozessen in eine Cloud, welche einen flexiblen und hoch skalierbaren Zugang zu verschiedenen Entwurfs- und Analysewerkzeugen ermöglicht, bieten bereits seit längerem unterschiedliche kommerzielle Anbieter auf Basis der Amazon Webservices an [3]. Hierbei reicht eine einfache Infrastruktur mit hoher Rechen- und Speicherkapazität aus, um insbesondere bei umfangreichen Entwürfen und zahlreichen Parametervariationen die erforderliche Leistung bereitzustellen. Neben der Infrastruktur, ist auch das Bereitstellen einer Plattform mit den erforderlichen Werkzeugen und der Software zum Systementwurf notwendig (SaaS – Software as a Service).

In Abschnitt 2.1 wird auf die Integration spezieller Hardware in Cloud-Systeme eingegangen. Bevor in Abschnitt 3 die angestrebte Integration von FPGAs vorgestellt wird, erläutert Abschnitt 2.2 zunächst übliche FPGA-Plattformen und ihre unterschiedlichen Anforderungen.

2.1 Hardwarebeschleuniger in der Cloud

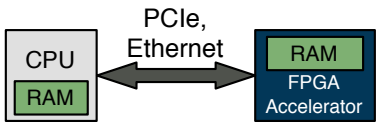
In Anwendungen, die eine hohe Rechenleistung erfordern und einfach zu parallelisieren sind, werden häufig Grafikkarten (GPUs) eingesetzt, um die Verarbeitungsgeschwindigkeit und den Datendurchsatz zu erhöhen. Viele Computing-Clouds bieten dem Nutzer mittlerweile solche Infrastrukturen (IaaS – Infrastructure as a Service) mit Grafikkarten als Hardwarebeschleuniger an [4], [5]. Im Bereich des Hochleistungsrechnens (HPC) gewinnt die Bereitstellung der reinen HPC-Infrastruktur für die Entwicklung eigener Anwendungen ebenfalls immer mehr an Bedeutung (HPCaaS – HPC as a Service [6]).

FPGAs werden in Rechenzentren ebenfalls als Hardwarebeschleuniger für spezielle Anwendungen mit einfachen Datenstrukturen und Programmflüssen eingesetzt, wie beispielsweise für einfache Datenbank-Anfragen [7]. Der Grund für den Einsatz von FPGAs liegt hierbei, neben der Verarbeitungsgeschwindigkeit, im Wesentlichen beim vergleichsweise geringen Energieverbrauch gegenüber Grafikkarten und Prozessoren. Für Cloud-Dienste ist es des Weiteren möglich, FPGAs zur Anonymisierung der Nutzeranfragen einzusetzen [8]. Die direkte Bereitstellung der eigentlichen Ressource FPGA wird derzeit jedoch nicht angeboten.

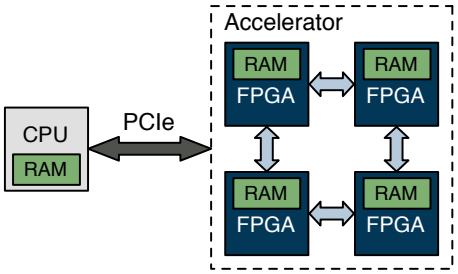
2.2 FPGA-Systeme und -Architekturen

Um eine Architektur zu entwickeln, welche FPGAs flexibel für die unterschiedlichsten Anwendungsfälle in eine Cloud integriert, wird im Folgenden eine Auswahl an Anwendungen und der dazugehörigen Konfiguration der Plattform betrachtet.

Ein Haupteinsatzgebiet für FPGAs stellt das Prototyping von SoCs oder einfachen Prozessoren dar. Dabei wird deren grundlegendes Verhalten sowie Funktionsweise auf der rekonfigurierbaren Hardware getestet. Die Kommunikation zwischen dem FPGA und der Analysesoftware auf einem Host-System erfolgt über eine Schnittstelle wie Gigabit-Ethernet oder PCIe. Abbildung 1a zeigt ein solches System, bestehend aus einem Prozessor, welcher direkt mit einem FPGA gekoppelt ist. Durch die Nähe zum Host-Prozessor kann der FPGA in einem solchen System ebenfalls als Hardwarebeschleuniger für Anwendungen eingesetzt werden, die eine hohe Datenrate zwischen Prozessor und Beschleuniger benötigen [9].



(a) Direkte Host-FPGA Verbindung über PCIe oder Ethernet.



(b) Multi-FPGA System mit zusätzlichen seriellen Gigabit Verbindungen der FPGAs untereinander.

Bild 1: Häufig genutzte Systemkonfigurationen für FPGAs als Hardwarebeschleuniger oder als Prototyping Plattform.

Beim Entwurf von Prozessoren mit mehreren Kernen oder komplexeren Systemen ist ein einzelner FPGA oftmals nicht ausreichend. Statt dessen werden mehrere eng miteinander gekoppelte FPGAs zum Testen der Hardware benötigt, wie Abbildung 1b zeigt [10], [11]. Die Verbindung zum Host-Prozessor ist bei derartigen Systemen zu vernachlässigen. Entscheidend ist vielmehr die Kommunikation zwischen den FPGAs unterinander.

Für eine flexible Cloud-Architektur ist ein Ansatz erforderlich, der einerseits eine unabhängige Nutzung einzelner Prozessor/FPGA-Knoten zulässt und so mehrere Nutzerentwürfe oder Parametervarianten auf den FPGAs parallel abarbeiten kann, aber andererseits auch umfangreiche Entwürfe über mehrere FPGAs erlaubt.

3 Integration von FPGAs in eine Cloud-Infrastruktur

Die angestrebte Cloud-Architektur setzt sich aus unterschiedlichen Clustern zusammen, welche hybride Rechenknoten bestehend aus Prozessoren und FPGAs benötigt. Diese Cluster werden in Abschnitt 3.1 beschrieben. Das erforderliche Grunddesign auf dem FPGA wird in Abschnitt 3.2 vorgestellt. Anschließend wird die Gesamtarchitektur aus Top-Level Sicht in Abschnitt 3.3 beschrieben. Ein Überblick über die erforderlichen Softwareschichten gibt Abschnitt 3.4.

3.1 Architektur eines Clusters

Ein Cluster fasst die Charakteristiken der beiden in Abschnitt 2.2 vorgestellten Systeme, in einer Architektur zusammen [12]. Die kleinste Komponente ist ein Rechenknoten (Node), bestehend aus einem Prozessor welcher mittels PCIe mit einem FPGA gekoppelt ist.

Vier oder acht dieser Rechenknoten können zu einem Cluster zusammengefasst werden, wobei die Prozessoren über ein Verbindungsnetzwerk gekoppelt sind. Die Besonderheit der Architektur, welche sie von anderen Systemen unterscheidet, besteht in der separaten Verbindung der FPGAs untereinander über serielle Kanäle mit einer Datenrate von bis zu 40 GBit/s (siehe Abbildung 2). Dadurch können auch komplexe Entwürfe getestet werden, welche den Einsatz spezieller Multi-FPGA Karten erfordern würden. Die Verwaltung und der Zugang zu einem Cluster erfolgt über einen Serviceknoten.

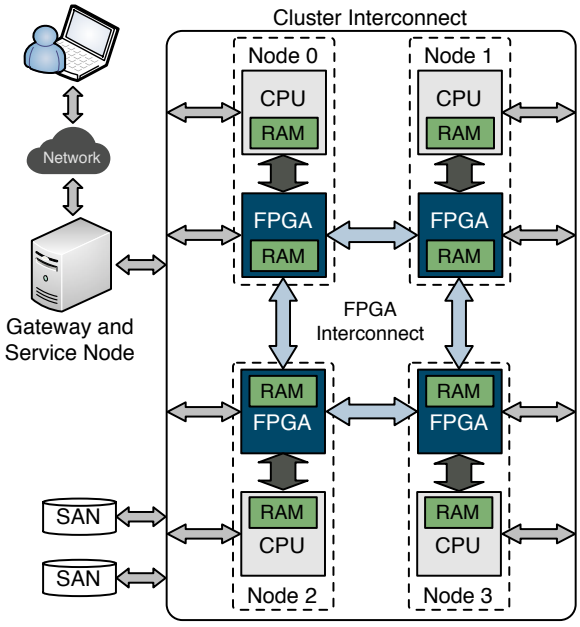


Bild 2: Systemarchitektur eines Clusters innerhalb der Cloud mit vier Knoten. Jeder Knoten besteht aus einem Host-Prozessor und einem FPGA. Die FPGAs sind über ein separates Netzwerk untereinander verbunden. Der Zugang und die Verwaltung eines Clusters erfolgt über einen Serviceknoten.

Für die Integration in eine Cloud stellt die Elastizität der Ressourcen ein wesentliches Kriterium dar. Innerhalb eines Clusters können einem Nutzer bis zu acht Rechenknoten zugeteilt werden. Bei kleineren Allokationen ist innerhalb eines Clusters ein Betrieb mit maximal acht unterschiedlichen Nutzern, die jeweils nur einen FPGA benötigen, möglich. Abbildung 6 zeigt ein Cluster mit vier Rechenknoten und unterschiedlichen Nutzern. Die von ihnen jeweils genutzten Ressourcen sind durch die unterschiedlichen Farben gekennzeichnet.

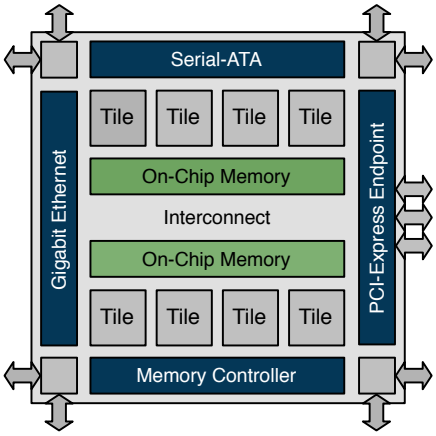


Bild 3: Architektur des FPGA-Designs mit Schnittstellen zu Host, Memory Controller und den anderen FPGAs.

3.2 FPGA-Grunddesign

In einer Architektur mit fest in das System eingebetteten FPGAs, ist es erforderlich, sowohl auf dem FPGA als auch auf dem Host-System die erforderlichen Schnittstellen in Hardware beziehungsweise in Software bereitzustellen. Abbildung 3 zeigt das vorgegebene Design auf dem FPGA.

Die Verbindung zum Host-System erfolgt über PCIe. Die Schnittstellen lassen dabei auf Seiten des FPGAs Statusabfragen von internen Registern sowie Blocktransfers in Speicherbereiche zu. Über einen Speicherkontroller kann weiterhin mit dem bis zu 8 GByte großen Hauptspeicher auf dem FPGA-Board kommuniziert werden. Die Schnittstellen zu den paketbasierten Verbindungen zwischen den FPGAs sind ebenfalls vorkonfiguriert. Mittels einer partiellen Rekonfiguration kann der Nutzer seinen Entwurf direkt auf den FPGA übertragen und die vorgefertigten Schnittstellen nutzen.

3.3 Top-Level Schicht

Ein einzelner Cluster kann als Cloud mit bis zu acht Knoten eingesetzt werden. Zugang und Verwaltung erfolgen in diesem Fall über den Serviceknoten. Die Verbindung mehrerer Cluster über ein Netzwerk erfolgt ebenfalls über die Serviceknoten der einzelnen Cluster. Diese wiederum werden bei einer solchen Konfiguration von einem zentralen System zum Management der gesamten Cloud-Infrastruktur verwaltet. In diesem Fall können sich die einzelnen Cluster in verschiedenen Rechenzentren befinden. Abbildung 4 zeigt die Top-Level-Architektur mit zentralem Management-System und zwei Clustern.

3.4 Integration der Software

Um die Cloud zu verwalten und die notwendigen Schnittstellen für die Arbeit mit der Hardware anzubieten, sind mehrere Software-Schichten notwendig. Abbildung 5 gibt einen Überblick über die Schichten, welche in zwei funktionale Bereiche eingeteilt werden können.

Die Management-Ebene ist für die Verwaltung der verteilten Ressourcen (bei mehreren Clustern) zuständig

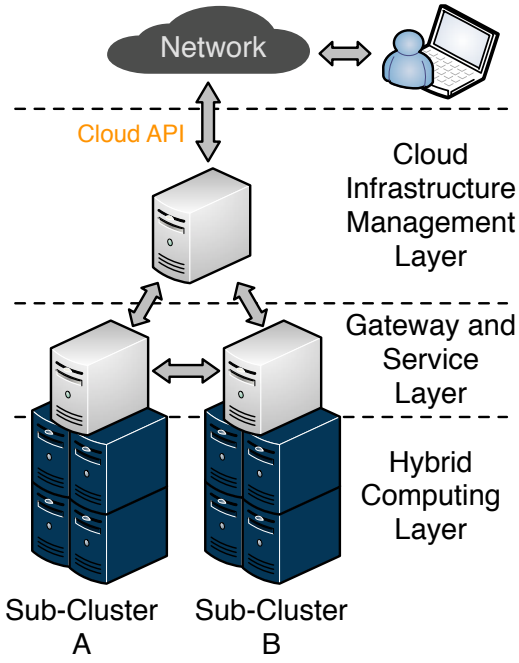


Bild 4: Cloud-Infrastruktur mit zwei Rechenzentren und den erforderlichen Ebenen, welche die Verwaltung der gesamten Infrastruktur sowie der Teilkomponenten beinhalten.

und stellt die Nutzerschnittstelle (Cloud-API) bereit. Die Management-Ebene beinhaltet ebenfalls die Authentifizierung der Nutzer und die Verschlüsselung der Datentransfers. Die Verteilung der Nutzer-Anfragen (Jobs) erfolgt über ein zweistufiges Batch-System. Die Management-Ebene stellt hierbei die erste Stufe dar und verteilt die Jobs auf die Cluster.

Die zweite Stufe des Batch-Systems befindet sich auf dem Serviceknoten der einzelnen Cluster und verteilt die Jobs auf die Rechenknoten. Ein Job beinhaltet hierbei entweder die erforderlichen Kommandozeilenbefehle in

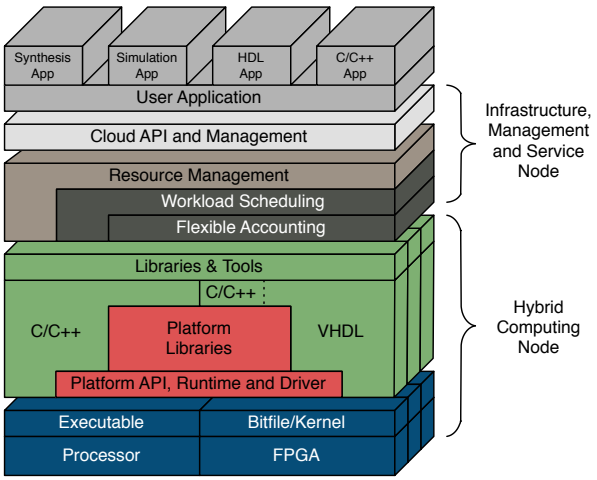


Bild 5: Software-Komponenten innerhalb der Cloud-Architektur, verteilt auf Service- und Rechenknoten.

einer Batch-Datei oder die Anforderung einer kompletten Virtuellen Maschine mit den erforderlichen Ressourcen. Bei kleinen Systemen, die nur aus einem Cluster bestehen, übernimmt der Serviceknoten des Clusters die vollständige Verwaltung und bietet die Nutzerschnittstelle an (oberer Bereich in Abbildung 5).

Die Rechenknoten bilden den unteren Bereich in Abbildung 5. Neben den eigentlichen Entwurfs- und Analysewerkzeugen befinden sich hier die erforderlichen Bibliotheken und Treiber, um eine Interaktion mit den FPGAs zu ermöglichen. Dazu gehören beispielsweise die Rekonfiguration und die Kommunikation mit dem Entwurf auf dem konfigurierten FPGA. Der Nutzer hat auf dieser Ebene nur Zugang zu höheren Schichten und muss sich nicht mit der konkreten Hardware im Detail auseinandersetzten.

Eine entscheidende Anforderung besteht in der Sicherheit der Daten und dem Schutz des Know-Hows über sämtliche Ebenen hinweg [13]. Um dieses zu gewährleisten werden sowohl die Kommunikation als auch die gesamten Nutzerdaten verschlüsselt.

4 Einordnung und Einsatzgebiete

Die Hauptanwendung, für welche die Cloud konzipiert ist, bildet der Systementwurf mit den Prozessen Synthese, Simulation und Test. Des Weiteren ist es möglich, die FPGAs als reine Hardwarebeschleuniger zu nutzen. Im Folgenden werden die Einsatzgebiete und potentiellen Anwendungen über das Servicemodell nach [14] erläutert.

4.1 SaaS – Software as a Service

Werden die für den Systementwurf notwendigen Anwendungen wie Synthesewerkzeuge und Simulatoren genutzt, ist die Cloud als Anbieter von reinen Services einzuordnen (SaaS). Des Weiteren ist es aber möglich, die FPGAs als Hardwarebeschleuniger für weitere Dienste einzusetzen, so dass sie für den eigentlichen Nutzer des Dienstes nicht sichtbar sind.

4.2 PaaS – Platform as a Service

Beim Test realer Hardware sowie bei der Entwicklung von Software auf den bereitgestellten Schnittstellen wird die Anwendung vom Nutzer in der Cloud entwickelt. Somit kann bei der Arbeit mit eigenen Entwürfen auf dem FPGA auch die Bereitstellung der Plattform als Service bezeichnet werden (PaaS).

4.3 IaaS – Infrastructure as a Service

Ausschließlich die Infrastruktur als Service anzubieten ist prinzipiell möglich, aufgrund der komplexen Einbettung des FPGAs in das Betriebssystem aber nur für spezielle Anwendungen erforderlich, die einen tieferen Eingriff in das System erfordern.

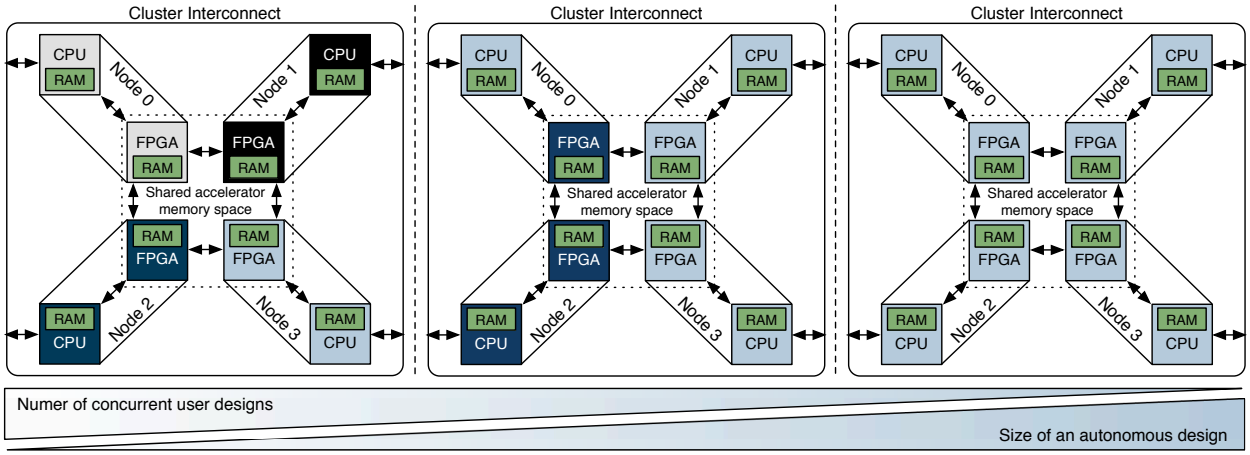


Bild 6: Hybride Multi-FPGA Plattform mit unterschiedlicher Belegung der Ressourcen. Konfiguration mit mehreren kleinen Designs (links) und mit einem großen Design verteilt über vier FPGAs (rechts). Die Farben repräsentieren die Nutzer und die ihnen zugeteilten Ressourcen.

5 Zusammenfassung

Die Integration rekonfigurierbarer Hardware in eine Cloud-Infrastruktur ist für viele komplexe Entwürfe und deren anschließenden Test auf realer Hardware von großer Bedeutung. Die vorgestellte Cloud-Architektur ist flexibel, skalierbar und kann in der Menge der Ressourcen an die unterschiedlichsten Anforderungen der Nutzer elastisch angepasst werden. Ebenso wird beim Betrieb mit mehreren Nutzern eine möglichst hohe Auslastung der Ressourcen ermöglicht. Die Anforderungen verbreiteter FPGA-Plattformen werden in einem einzigen System zusammengefasst, so dass zahlreiche Konfigurationen ermöglicht werden. Des Weiteren werden auf unterer Ebene die Schnittstellen auf Hardware- sowie Softwareseite bereitgestellt.

Das System kann durch seine modulare Architektur einfach erweitert und ebenso auf räumlich getrennte Rechenzentren verteilt werden. Aus Sicht des Nutzers werden ähnliche Schnittstellen angeboten wie bei anderen Computing-Clouds [15].

Durch den Zugriff auf FPGA-Ressourcen beliebiger Anzahl und Größe können Unternehmen und wissenschaftliche Einrichtungen Anschaffungskosten sparen und die Hardware kann durch die zentrale Bereitstellung für verschiedenste verteilte Nutzer effizient ausgelastet werden. Ähnlich wie High Performance Computing as a Service (HPCaaS) kann analog dazu von Reconfigurable Computing as a Service (RCaaS) gesprochen werden.

Literatur

[1] S. L. Garfinkel, *Architects of the Information Society: Thirty-Five Years of the Laboratory for Computer Science at MIT*. The MIT Press, 1999.

[2] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica *et al.*, "A view of cloud computing," *Communications of the ACM*, vol. 53, no. 4, pp. 50–58, 2010.

[3] Plunify Pte Ltd, "Simplify your chip design," September, 2013. [Online]. Available: http://www.plunify.com

[4] W. Zhu, C. Luo, J. Wang, and S. Li, "Multimedia cloud computing," *Signal Processing Magazine, IEEE*, vol. 28, no. 3, pp. 59–69, 2011.

[5] F. Giunta, R. Montella, G. Laccetti, F. Isaila, and F. Blas, "A gpu accelerated high performance cloud computing infrastructure for grid computing based virtual environmental laboratory," *Advances in Grid Computing*, 2011.

[6] V. Mauch, M. Kunze, and M. Hillenbrand, "High performance cloud computing," *Future Generation Computer Systems*, 2012.

[7] L. Woods, Z. István, and G. Alonso, "Hybrid fpga-accelerated sql query processing."

[8] K. Eguro and R. Venkatesan, "Fpgas for trusted cloud computing," in *Field Programmable Logic and Applications (FPL), 2012 22nd International Conference on*. IEEE, 2012, pp. 63–70.

[9] O. Knodel et al., "Next-generation massively parallel short-read mapping on FPGAs," in *Application-Specific Systems, Architectures and Processors (ASAP), 2011 IEEE International Conference on*. IEEE, 2011.

[10] J. Wawrzynek et al., "RAMP: Research accelerator for multiple processors," *Micro, IEEE*, vol. 27, no. 2, 2007.

[11] R. Sass et al., "Reconfigurable computing cluster (RCC) project: Investigating the feasibility of FPGA-based petascale computing," in *Field-Programmable Custom Computing Machines, 2007. FCCM 2007. 15th Annual IEEE Symposium on*. IEEE, 2007.

[12] O. Knodel and R. Spallek, "Integration of a multi-fpga system in a common cluster environment," *The International Conference on Field Programmable Logic and Applications (FPL)*, 2013.

[13] M. Hansen, "Datenschutz im cloud computing," in *Daten- und Identitätsschutz in Cloud Computing, E-Government und E-Commerce*. Springer, 2012, pp. 79–95.

[14] P. Mell and T. Grance, "The nist definition of cloud computing (draft)," *NIST special publication*, vol. 800, no. 145, p. 7, 2011.

[15] The OpenStack project, "Open source software for building private and public clouds," September, 2013. [Online]. Available: http://www.openstack.org

Ein Cloud-basierter Workflow für die effektive Fehlerdiagnose von Loop-Back-Strukturen

Matthias Gulbins, André Schneider, Steffen Rülke
Fraunhofer IIS/EAS Dresden
{matthias.gulbins|andre.schneider|steffen.ruelke}@eas.iis.fraunhofer.de

Kurzfassung

Eine hochkomplexe und zeitaufwändige Aufgabe beim Entwurf integrierter Mixed-Signal-Schaltkreise ist die Fehlerdiagnose. Der vorliegende Beitrag stellt einen auf Cloud-Technologien basierenden Lösungsansatz vor, der Fehler in für solche Schaltkreise typischen Strukturen aus Analog-Digital- und Digital-Analog-Wandlern lokalisiert. Das Diagnoseverfahren (Ergebnis des BMBF-Projektes *DIANA*) beruht auf dem sogenannten Loop-Back-Test, der zwar die Generierung von Testdaten vereinfacht, aber eine Vielzahl von Variantensimulationen mit verschiedenen Simulationsprinzipien und erheblichen Datenmengen erfordert. Diese sollen nunmehr problemangepasst und damit effizient in der Cloud realisiert werden. Für die entsprechende Informationsverarbeitung in der Cloud wurde das in dem Projekte *OptiNum-Grid* entwickelte Framework *GridWorker* adaptiert. Experimente mit ersten Anwendungsbeispielen bestätigen die Leistungsfähigkeit und Praktikabilität des Ansatzes für daten- und verarbeitungsintensive Schaltkreisentwurfsaufgaben.

1 Einleitung

Analog-Digital- und Digital-Analog-Wandler (ADC/DAC) sind typische Schaltungskomponenten vieler Mixed-Signal-Schaltkreise, z.B. im Bereich der Automobilelektronik. Der Test solcher Strukturen und die Diagnose der Fehlerursachen gestalten sich als überaus komplex und aufwendig. Eine Strategie zur Reduktion der Test- und Diagnosekosten ist ein kombiniertes Vorgehen aus strukturellen und simulations-methodischen Ansätzen. Neben einer Loop-Back-Struktur für den Test kommen hierarchische Ansätze und Cloud-Technologien für die Fehlerdiagnose zur Anwendung.

Bei einer Loop-Back-Struktur werden für den Test DAC und ADC in Reihe geschaltet, so dass das Ausgangssignal des ADCs mit dem Eingangssignal des DACs verglichen werden kann. Vorteile sind, dass

- der Test digital abläuft, was eine einfache Stimuligenerierung und eine reproduzierbare Fehlerdiagnose ermöglicht,
- die Fehler der Wandler direkt auf der elektrischen Netzwerkebene definiert werden und
- Schwankungen der Herstellungsparameter von Schaltkreisen unmittelbar beim Test berücksichtigt werden.

Für die Beherrschbarkeit der hochkomplexen und (entsprechend der Parametervariationen und der Vielfalt von möglichen Fehlerorten und -werten) sehr umfangreichen Fehlersimulationen ist folgendes vorgesehen:

- Einerseits ist ein hierarchisches Vorgehen erforderlich: Alle Fehlersimulationen der Wandler werden auf Netzwerkebene ausgeführt, die Ergebnisse in die funktionelle Ebene überführt und die Simulationen in der Loop-Back-Umgebung sowie die Fehlerdiagnose auf funktioneller Ebene durchgeführt.
- Andererseits werden alle Operationen (Simulationen und Konvertierungen auf Netzwerk- und Funktionsebene) in einer Cloud unter Verwendung geeigneter Virtualisierungstechnologien (*GridWorker*) ausgeführt.

Mit dem vorgestellten Lösungsansatz wird die Leistungsfähigkeit der Cloud-Technologien für die zunehmend daten- und verarbeitungsintensiven Schaltkreisentwurfsaufgaben demonstriert. Gerade aus der immer höheren Integrationsdichte und der zunehmenden Verarbeitungsbreite der Schaltungskomponenten erwachsen diesbezüglich erhebliche Anforderungen.

Weiterhin ist der beschriebene Lösungsansatz auch die Grundlage für künftige Dienstleistungsangebote zur Varianten- bzw. Fehlersimulation, Analyse und Diagnose beim Mixed-Signal-Schaltkreisentwurf. Dabei könnten auch die von den Autoren entwickelten Werkzeuge zur Informationsverarbeitung in der Cloud (*GridWorker*) und zur analogen Fehlersimulation (*aFSIM*) als *Software-as-a-Service* (SaaS) Verwendung finden.

Schließlich wird mit dem Lösungsansatz auch die Qualität der Fehlerdiagnose signifikant erhöht. Das wird möglich, weil dank der Cloud-Technologien anstelle von relativ groben Verhaltensmodellen nunmehr auch genauere Strukturmodelle auf der elektrischen Netzwerkebene verwendet werden können, die das auszuwertende Datenvolumen und Anzahl der erforderlichen Simulationsläufe drastisch ansteigen lassen.

Der weitere Beitrag ist wie folgt gegliedert: Abschnitt 2 erläutert das Fehlerdiagnoseverfahren von Loop-Back-Strukturen, Abschnitt 3 stellt den Cloud-basierten Workflow dafür vor und Abschnitt 4 die Adap-

tion des *Gridworker*s an diesen Workflow. Abschnitt 5 diskutiert die bei Experimenten erreichten Ergebnisse. Eine Zusammenfassung sowie der Ausblick auf weiterführende Arbeiten schließen den Beitrag ab.

2 Prinzip der Fehlerdiagnose für Loop-Back-Strukturen

Wichtige Schritte bei der Fertigung integrierter nanoelektronischer Systeme (System-on-Chip – SoC) sind Fertigungs- und On-Chip-Test sowie – im Fehlerfall – die Fehlerdiagnose. Insbesondere bei SoC mit Mixed-Signal-Komponenten ist dies meist sehr zeit- und kostenaufwendig. Ein Ansatz zur Aufwandsreduktion ist der Loop-Back-Test.

Grundliegende Idee des Loop-Back-Tests ist es, zwei funktional zueinander inverse Blöcke in Reihe zu schalten und den eigentlichen Test dann für diese Zusammenschaltung durchzuführen. Der Testaufwand kann erheblich reduziert werden, weil Ein- und Ausgangssignale der Zusammenschaltung weitgehend einander entsprechen und damit einfach verglichen werden können

Im Projekt DIANA wurde ein Zugang entwickelt, um den Loop-Back-Test auch für Mixed-Signal-Komponenten vom Typ ADC und DAC anzuwenden, wenn diese gemeinsam in einem System-on-Chip-Halbleiterbaustein integriert sind. Die Zusammenschaltung von DAC und ADC zu einer Loop-Back-Struktur mit digitalen Ein- und Ausgängen ermöglicht den Test auf digitaler Basis. Dabei erfolgt ein statisches Vorgehen, bei dem der ADC mit einer Rampenfunktion und der DAC mit einer Treppenfunktion stimuliert werden. Nach der Stimulierung des DACs mit einer digitalen Treppenfunktion wird das analoge Ausgangssignal des DACs auf den analogen Eingang des ADCs geschaltet. Ein zwischengeschalteter Tiefpass erzeugt analoge Zwischenwerte, um die Rampenfunktion besser nachzubilden und die erforderliche Testgenauigkeit zu erreichen (vergl. Abbildung 1). Die Ausgabe des ADCs ist eine Folge aus Digitalwerten.

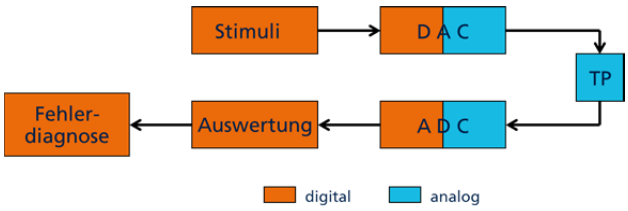


Abbildung 1: Loop-Back-Struktur aus Digital-Analog- und Analog-Digital-Wandler

Aufgrund der großen Datenmenge, werden aus der Ausgabe des ADCs, also der Loop-Back-Struktur, Kenngrößen berechnet, die es auch für jeden der beiden Wandler gibt. Statische Kenngrößen sind z. B. integrale Nichtlinearität (INL) und differentielle Nichtlinearität (DNL) [1]. Zur Verbesserung der Fehlerdiagnose können weitere Kenngrößen wie Monotonie oder fehlende Codes herangezogen werden. Alle Kenngrößen einer Folge aus Digitalwerten werden zu einer Signatur zusammengefasst, die diese Folge charakterisiert.

Im Unterschied zu den aus der Literatur bekannten Verfahren liegt der Schwerpunkt bei unserem Vorgehen auf der Fehlerdiagnose [2], [3]. Aus Stimulus und Ausgabe der Loop-Back-Schaltung soll nicht nur erkannt werden, ob ADC und DAC innerhalb einer vorgegebenen Genauigkeit korrekt funktionieren, sondern im Fehlerfall auch diagnostiziert werden, welcher der beiden Wandler fehlerbehaftet und wo der Fehler lokalisiert ist.

Dafür wird die fehlerfreie Loop-Back-Schaltung simuliert und aus den Simulationsergebnissen die Kenngrößen für die Signatur berechnet. Anschließend wird ein Fehler ausgewählt und in die Schaltung eingefügt, d. h., die Schaltung entsprechend diesem Fehler verändert. Die mit dieser Schaltung ausgeführte Simulation liefert eine Fehlersignatur. Die Signatur der fehlerfreien Schaltung und die Fehlersignaturen aller Fehler werden vor der eigentlichen Fehlerdiagnose berechnet und dann bei der Fehlerdiagnose mit der Signatur der konkreten zu testenden Schaltung verglichen.

Bei der Fehlerdiagnose wird die zu untersuchende Schaltung mit der Treppenfunktion stimuliert und aus der Ausgabefolge die Kenngrößen und damit die „gemessene“ Fehlersignatur berechnet. Diese „gemessene“ Fehlersignatur wird mit den berechneten Fehlersignaturen verglichen und aus dem Vergleich auf den vorliegenden Fehler geschlossen.

Um einen möglichst großen Hardware-Bezug zu erreichen, wurden die Wandler auf der elektrischen Netzwerkebene modelliert. Als Fehler wurden Kurzschlüsse und Unterbrechungen sowie parametrische Veränderungen der Widerstände und Kapazitäten betrachtet.

Da Simulationen auf elektrischer Netzwerkebene sehr zeitaufwändig sind, wurde ein hierarchisches Vorgehen gewählt. Dabei werden ADC und DAC unabhängig voneinander simuliert und die Simulationsergebnisse jeweils in Kennlinien umgewandelt. Die Simulation der Loop-Back-Zusammenschaltung erfolgt unter Verwendung der Kennlinien auf der Verhaltensebene z. B. mit MATLAB.

Bei der Durchführung der Fehlerdiagnose hat sich herausgestellt, dass sich viele Fehlersignaturen nur geringfügig unterscheiden, so dass viele Fehler als Fehlerkandidaten für eine gemessene Fehlersignatur in Frage kommen. Ein Abstandsmaß ist schwer zu definieren. Dabei stellt sich die Frage, wie genau die Kennlinien sind, denn diese werden von Parametern, z. B. Kennwerten von Bauelementen wie Widerständen und Transistoren oder von Zellen wie Operationsverstärker, beeinflusst. Infolge von kleinen Veränderungen im Produktionsprozess können diese Parameter variieren. Von ihnen seien jeweils Nominalwert und Varianz bekannt. Die Parametervariationen werden in Monte-Carlo-Simulationen nachgebildet und so Kennlinien erzeugt, die diese Parametervariationen berücksichtigen. Damit sind eine Quantifizierung der Fehlerabstände und eine Verbesserung der Fehlerdiagnose möglich.

Durch die Monte-Carlo-Simulationen erhöht sich der Simulationsaufwand drastisch. Um die damit gleichfalls wachsende Datenmenge zu reduzieren und für die Fehlerdiagnose anwendbar zu machen, wird neben den Nominalwerten nur die Datenverteilung in Form von Histogrammen sowie Min-Max-Intervallen verwendet.

Die Kenngrößen und die Min-Max-Intervalle einer Fehlersignatur bilden die Grundlage für die Fehlerdiagnose. Werden *n* Kenngrößen als Achsen eines kartesischen Koordinatensystems betrachtet, so liegen für jeden Fehler alle möglichen Kombinationen der *n* Kenngrößen innerhalb eines *n*-dimensionalen Quaders, der durch die Min-Max-Intervalle begrenzt wird. Zur Veranschaulichung werden die beiden Kenngrößen „Gain Error“ und „Offset Error“ betrachtet, wie in Abbildung 2 illustriert. Für den fehlerfreien Fall F0 und sechs mögliche Fehler F1, …, F6 wurden per Monte-Carlo-Simulation die Intervalle für Offset und Gain Error bestimmt, in denen diese Kenngrößen auftreten. Diese spannen unterschiedlich große Flächen auf. Aufgabe der Fehlerdiagnose ist es, einer gemessenen Fehlersignatur Ti Fehlerkandidaten zuzuordnen. Abbildung 2 zeigt, dass sich diese Aufgabe grafisch lösen lässt: Liegt die Fehlersignatur Ti innerhalb der aufgespannten Fläche bzw. im aufgespannten *n*-dimensionalen Quader eines Fehlers, so kommt dieser Fehler als Kandidat für die vorliegende Fehlersignatur in Frage. Gibt es nur einen Kandidaten, so ist die Fehlerdiagnose damit beendet. Im vorliegenden Beispiel gemäß Abbildung 2 wird bei den Fehlersignaturen T1 und T2 kein Fehler (F0); bei T3 Fehler F3; bei T4 F5 und bei T5 F4 klassifiziert. Gibt es mehrere Kandidaten, so wird zunächst der Fehlerort (Fehlerlokalisation) bestimmt.

Sind die Fehler F1 und F5 Fehler am gleichen Fehlerort mit unterschiedlichen Fehlerwerten, so ist der Fehlerort gefunden. Im Schritt der Fehleridentifikation muss der Fehlerwert, d. h. die Abweichung von einem Nominalwert, bestimmt werden.

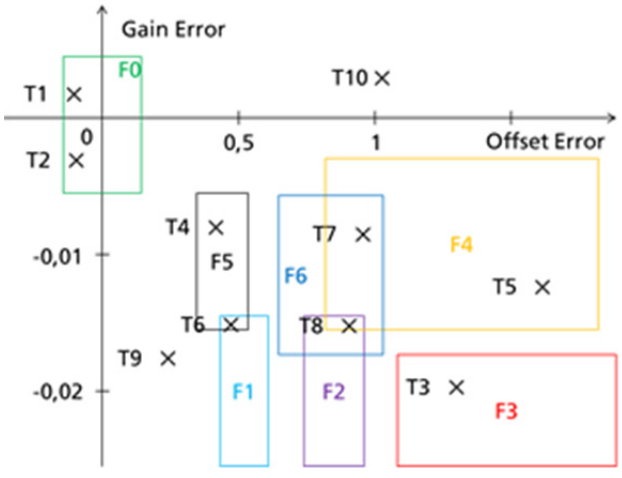


Abbildung 2: Fehlerintervalle Fi der Kenngrößen „Gain Error“ und „Offset Error“ für verschiedene Fehler

Haben die Fehlerkandidaten unterschiedliche Fehlerorte, so können weitere Kenngrößen in Simulation und Messung einbezogen werden, um so die Fehlerkandidaten voneinander zu unterscheiden.

Unabhängig davon ist eine Bewertung der Kandidaten unter Verwendung der berechneten Verteilungen möglich. Dabei werden Fehler gleichen Fehlerorts zusammengefasst.

In Abbildung 2 gibt es für die Fehlersignaturen T6 und T7 jeweils zwei, F1 und F5 bzw. F4 und F6, sowie für die Fehlersignatur T8 drei Kandidaten: F2, F4 und F6.

Falls einer gemessenen Fehlersignatur kein Fehler zugeordnet werden kann, gehört diese Signatur nicht zu den in den Simulationen untersuchten Fehlern. In Abbildung 2 gilt das für T9 und T10.

3 Workflow zur Erzeugung der Fehlersignaturen

Wie in Abschnitt 2 erläutert, erfordert die Detektion von Fehlern in den ADC/DAC-Komponenten eines konkreten gefertigten Mixed-Signal-Schaltkreises sogenannte Fehlersignaturen, die im Rahmen eines Loop-Back-Tests zu berechnen sind.

Der in diesem Abschnitt vorgestellte Workflow realisiert die algorithmische Umsetzung des Loop-Back-Tests durch Simulatoren und Skripte und liefert im Ergebnis die für die Fehlerdiagnose benötigten Fehlersignaturen. Für berechnungs- und datenintensiven Schritte werden dabei Cloud-Technologien eingesetzt.

Für die Arbeiten wurde zunächst der im Fraunhofer IIS/EAS entwickelte analoge Fehlersimulator *aFSIM* [4] in Verbindung mit dem Netzwerksimulator ngSPICE sowie Octave als Rahmenprogramm und Simulationsprogramm auf Verhaltensebene verwendet.

Der analoge Fehlersimulator *aFSIM* ist ein Programm zur analogen Fehlersimulation auf elektrischer Netzwerkebene, mit dem auf der Grundlage von symbolischen Fehlerbeschreibungen in Form von Pattern eine Fehlerliste erzeugt, die Fehler der Fehlerliste in die Schaltung injiziert, die Fehlersimulationen angestoßen und die Ergebnisse der Fehlersimulation ausgewertet werden können. Für unsere Zwecke werden nur die Funktionen Fehlerlistengenerierung und Fehlerinjektion verwendet. Die Simulationsaufrufe erfolgen vom *GridWorker* und die Auswertung der Simulationsergebnisse auf Verhaltensebene mit Hilfe von Octave, einer Open-Source-Alternative zu MATLAB.

Abbildung 3 zeigt den Workflow zur Erzeugung von erweiterten Fehlersignaturen für die Fehlerdiagnose. Um die große Anzahl der Dateien zu verdeutlichen, wird deren Anzahl in Abbildung 3 durch die Variablen x, y und m ausgedrückt. Dabei ist x die Anzahl der Fehler im ADC, y die der Fehler im DAC und m die Anzahl der Monte-Carlo-Simulationen. Für die 5-Bit-Varianten gilt: (x, y, m) = (9.137, 395, 1.000).

Die Kennlinien für den ADC und für den DAC werden unabhängig voneinander berechnet. Ausgangspunkt ist jeweils eine elektrische Netzliste und eine symbolische Beschreibung der Fehler. Im **Schritt 1** wird daraus

jeweils eine Fehlerliste erzeugt. Mit Hilfe eines Skripts werden die Fehler separiert, und es entsteht für jeden Fehler eine Einzelfehlerliste. Im folgenden **Schritt 2** wird für jeden Fehler aus elektrischer Netzliste und Einzelfehlerliste eine modifizierte Netzliste generiert. Die in dieser modifizierten Netzliste enthaltenen Widerstände und Kapazitäten werden per Skript einer Parametrisierung und damit der Monte-Carlo-Simulation zugänglich gemacht. Mit Hilfe des *GridWorkers* wird in **Schritt 3** jede der modifizierten Netzlisten mit ngSPICE einer Monte-Carlo-Simulation unterworfen. Die Dateien mit den SPICE-Simulationsergebnissen enthalten sehr viele Daten, während die Kennlinien selbst nur aus der Zuordnung analoger Eingangswert – digitaler Ausgangswert beim ADC und digitaler Eingangswert – analoger Ausgangswert beim DAC bestehen. In **Schritt 4** extrahiert deshalb ein Octave-Programm aus den SPICE-Simulationsergebnissen die Kennlinien.

Im Ergebnis erhält man für den fehlerfreien und jeden fehlerhaften Wandler ein Kennlinienbündel, bestehend aus der Nominalkennlinie und den Kennlinien der Monte-Carlo-Simulation. Die ersten vier Schritte beziehen sich überwiegend auf die elektrische Netzwerkebene und sind deshalb sehr zeitintensiv.

Sie liefern die Daten für **Schritt 5**, die Loop-Back-Simulation. Dabei kommt es darauf an, einen DAC und einen ADC, jeweils repräsentiert durch ein Kennlinienbündel, in der Loop-Back-Struktur zu simulieren. Aus Effektivitätsgründen wird die Simulation auf der Verhaltensebene durchgeführt. Die Kennlinien wurden schon entsprechend konvertiert.

Zunächst werden die Kenngrößen für den fehlerfreien Fall berechnet. Deren Nominalwerte ergeben sich bei Simulation der Nominalkennlinien der fehlerfreien Wandler. Durch Kombination von je einer Monte-Carlo-Kennlinie des einen mit einer Monte-Carlo-Kennlinie des anderen Wandlers erhält man eine entsprechende große Anzahl von variierten Werten für jede Kenngröße. Die große Anzahl von Werten wird in **Schritt 6** dadurch reduziert, dass nur die Nominalwerte der Kenngrößen aufgehoben und alle anderen in 11 Bins eingeordnet werden. So entsteht ein Histogramm für jede Kenngröße.

Die Fehlerdiagnose geht von einem Einzelfehlermodell aus. Das bedeutet, dass nur einer der beiden Wandler fehlerbehaftet und nur einer der Fehler wirksam ist. Bei der Fehlersimulation in der Loop-Back-Struktur muss deshalb jeder fehlerhafte Wandler mit dem fehlerfreien anderen Wandler kombiniert werden. Zur Berechnung der Nominalwerte der Kenngrößen eines Fehlers werden die Nominalkennlinien ausgewählt und in der Loop-Back-Struktur simuliert. Danach wird jede Monte-Carlo-Kennlinie des Fehlers mit einer zufällig ausgewählten Monte-Carlo-Kennlinie des fehlerfreien anderen Wandlers kombiniert. In Schritt 6 werden wie im fehlerfreien Fall die berechneten Kenngrößen der Monte-Carlo-Kennlinien in jeweils 11 Bins zu Histogrammen zusammengefasst. Auf diese Weise erhält man eine erweiterte Signatur für die fehlerfreien Wandler und erweiterte Fehlersignaturen für jeden Fehler des ADCs und jeden Fehler des DACs.

Abschließend erfolgt die in Abschnitt 2 beschriebene eigentliche Fehlerdiagnose, d. h., der Vergleich einer gemessenen Fehlersignatur mit den vorausberechneten Fehlersignaturen. Im beschriebenen Ansatz wird dieser Schritt lokal mit der Rechentechnik des Nutzers realisiert, ist also nicht an die Verfügbarkeit einer Cloud-Infrastruktur gebunden.

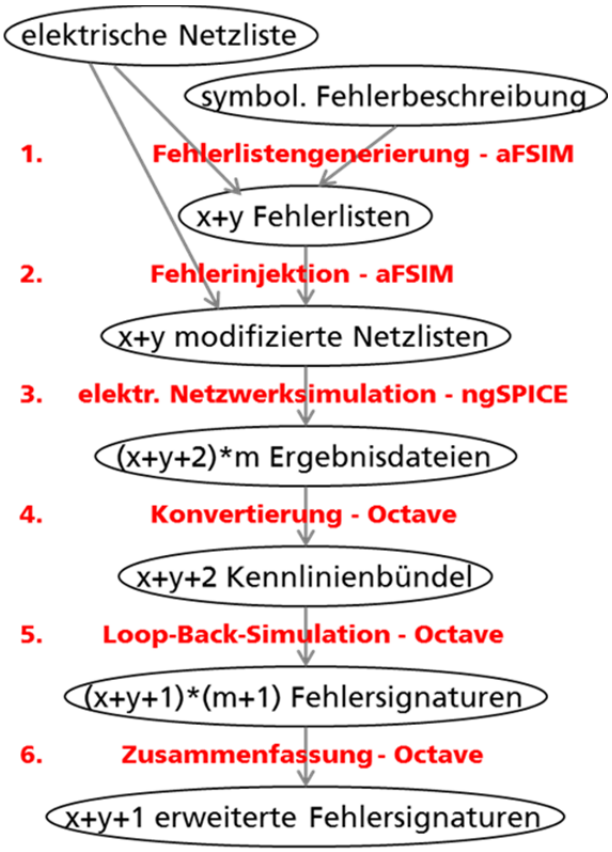


Abbildung 3: Workflow zur Erzeugung von erweiterten Fehlersignaturen. x Anzahl der ADC-, y Anzahl der DAC-Fehler, m Anzahl der Monte-Carlo-Simulationen

4 Adaption *GridWorker*

GridWorker ist ein am Fraunhofer IIS/EAS entwickeltes Framework [5], das auf Basis des Map/Reduce-Paradigmas [6] parallel ausführbare Bearbeitungseinheiten (Map-Phase) über angepasste Adapter auf Multi-Core-, Cluster-, Grid- und Cloud-Infrastrukturen verteilen und nach Beendigung die Ergebnisse wieder zusammenführen kann (Reduce-Phase). Ziel ist es dabei, die wachsenden Compute-Ressourcen insbesondere im Grid- und Cloud-Umfeld für Endanwender im Systementwurf zu erschließen. Diese haben in der Regel keine Möglichkeit, die vielfältigen Low-Level-Middleware-Schnittstellen wie gLite, UNICORE, Globus Toolkit, Grid Engine, PBS (Portable Batch System), OpenStack, OCCI (Open Cloud Computing Interface) etc. direkt aus ihren Applikationen

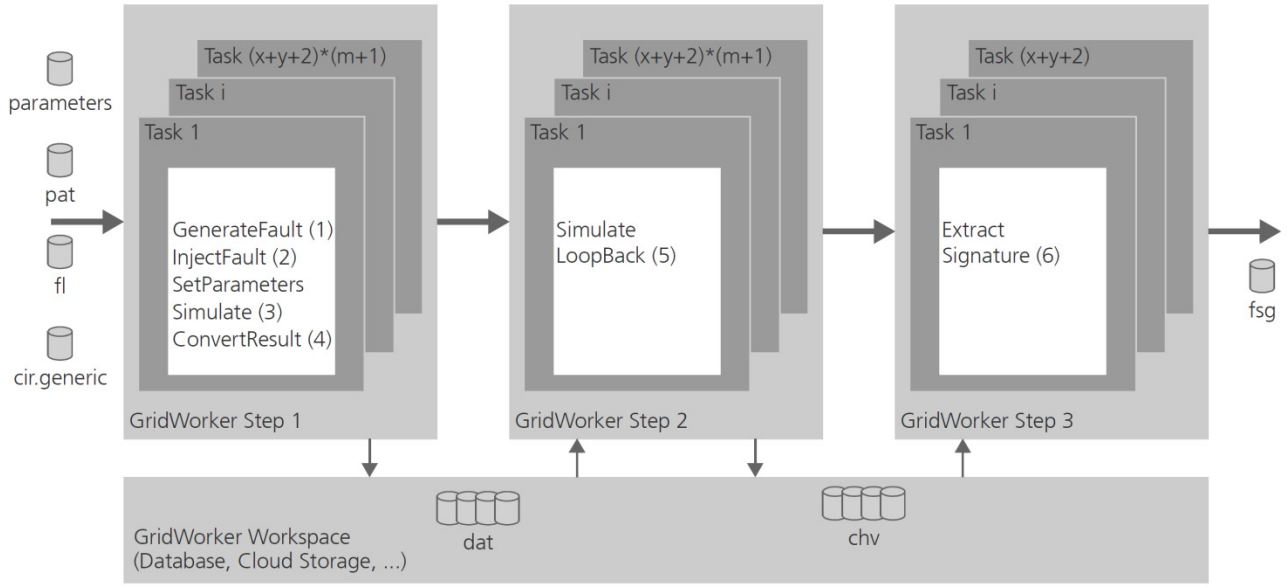


Abbildung 4: Gesamtablauf des Workflows aus *Gridworker*-Perspektive

heraus anzusprechen. *GridWorker* schließt die Lücke zwischen diesen Middleware-Diensten und den Entwurfsprogrammen und ermöglicht dem Systementwerfer das Konfigurieren von Simulations-, Optimierungs- oder Analyseexperimenten und deren Ausführung auf jeweils verfügbaren Backend-Ressourcen, wie sie beispielsweise beim Cloud-Computing von Providern zur Verfügung gestellt werden.

Bei der Fehlerdiagnose besteht die Aufgabe, für alle Fehler jeweils eine Monte-Carlo-Simulation durchzuführen. Da die Einzelsimulationen untereinander entkoppelt und damit unabhängig voneinander ausgeführt werden können, kann der Gesamtzeitbedarf drastisch reduziert werden, wenn Cluster-, Grid- oder Cloud-Ressourcen mit Hilfe von *GridWorker* eingesetzt und ein maximal möglicher Parallelisierungsgrad ausgenutzt werden.

Die Realisierung der Fehlerdiagnose mit *GridWorker* erfolgte in fünf Schritten:

1. Der gesamte Fehlerdiagnose-Workflow wurde in drei Phasen (Step 1 - 3) zerlegt, da es im gesamten Ablauf zwei Synchronisationspunkte gibt: Die Loop-Back-Simulationen (5) können erst ausgeführt werden, wenn die Ergebnisse aller Schaltungssimulationen vorliegen. Und die Fehlersignaturen können erst nach Vorliegen der einzelnen Signaturen zusammengefasst werden (6).

2. Alle pro Einzelsimulation erforderlichen Programmaufrufe wurden in einem Shellskript *map.sh* zusammengefasst. *GridWorker* ruft dieses Skript während der Map-Phase für einen konkreten Fehlerfall mit den jeweils durch den Monte-Carlo-Algorithmus bestimmten Parameterwerten auf, führt die Simulation durch und generiert als Ergebnis eine Kennliniendatei (Step 1).

3. Auch die für die Loop-Back-Simulation notwendigen Programmaufrufe (MATLAB oder Octave) wurden in einem Shellskript *map.sh* beschrieben und *GridWorker* für den Step 2 zur Verfügung gestellt. Analog wurde für Step 3 vorgegangen.

4. Da zwischen den einzelnen Phasen sehr viele Dateien (Tausend bis Millionen) übergeben werden müssen, wurde *GridWorker* um das Konzept der zentralisierten Speicherung erweitert. Dabei kann der Anwender bei Bedarf zwei Referenzen WORKSPACE und STORAGE konfigurieren und jeder *GridWorker*-Task die Möglichkeit geben, über diese Referenzen Daten zentralisiert abzulegen. Das Konzept erlaubt es, über die Standard-URL-Syntax Pfade in Dateisystemen, Datenbanken oder Referenzen innerhalb von Cloud ObjectStorages zu spezifizieren.

5. Schließlich wurden konkrete Anwendungsbeispiele für die Verwendung mit *GridWorker* aufbereitet. Dazu wird jeweils das Simulationsmodell für den fehlerfreien Fall und die zu untersuchenden Fehlermodelle in einem Verzeichnis *inputs/* bereitgestellt. Außerdem müssen die bei der Monte-Carlo-Simulation zu variierenden Parameter in einer Datei *parameters* spezifiziert werden. Die *GridWorker*-bezogenen Konfigurationen (Cloud-Adapter etc.) werden in einer Datei *configuration* zusammengefasst.

5 Anwendungsbeispiele und Experimente

Der Nachweis der Wirksamkeit des Workflows wurde zunächst anhand der 3-Bit-Varianten von Flash-ADC und R2R-DAC geführt (Blockschaltbilder s. Abbildung 5).

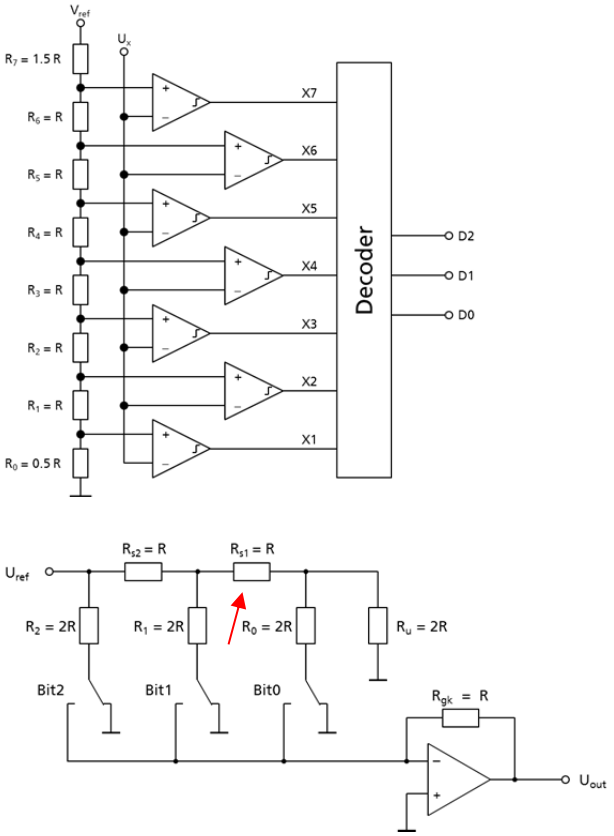


Abbildung 5: 3-Bit-Flash-ADC (oben) und 3-Bit-R2R-DAC (unten)

Ausgangspunkt für die Fehlerdiagnose ist die elektrische Netzliste, die den Aufbau jedes Wandlers aus Widerständen, Kondensatoren und Transistoren beschreibt. Die Netzliste des 3-Bit-Flash-ADCs enthält 85 Widerstände, sieben Kondensatoren und 161 Transistoren, die des 3-Bit-R2R-DACs 17 Widerstände, einen Kondensator und 21 Transistoren. Die symbolische Fehlerbeschreibung gibt an, wie sich diese Bauelemente im Fehlerfall verändern können. Bei den Widerständen wird deren Wert von Kurzschluss (sehr kleiner Wert) bis Unterbrechung (sehr großer Wert) variiert, wobei hier mit insgesamt acht verschiedenen Widerstandswerten gearbeitet wird. Das betrifft insbesondere die funktionsbestimmenden Widerstände, beim Flash-ADC die der Widerstandskette und beim R2R-DAC die des R2R-Netzwerkes. Bei den Kondensatoren und bei den Transistoren werden nur Kurzschluss- und Unterbrechungsfehler betrachtet.

Bei der Fehlerlistengenerierung werden die Fehlerdefinitionen der symbolischen Fehlerbeschreibung auf die elektrische Netzliste angewendet und auf diese Weise eine Fehlerliste erzeugt, auf die sich dann die Fehlerdiagnose bezieht. Da der Flash-ADC für jede Stelle im Thermometer-Code einen Widerstand und einen Komparator benötigt, nimmt die Anzahl der Fehler exponentiell mit der Stellenzahl zu. So hat der 3-Bit-Flash-ADC 1.197 Fehler, die 5-Bit-Variante hingegen 9.137 Fehler. Demgegenüber nimmt die Anzahl der Fehler beim R2R-DAC nur linear zu: 223 Fehlern bei der 3-Bit-Variante stehen 395 Fehler bei der 5-Bit-Variante gegenüber.

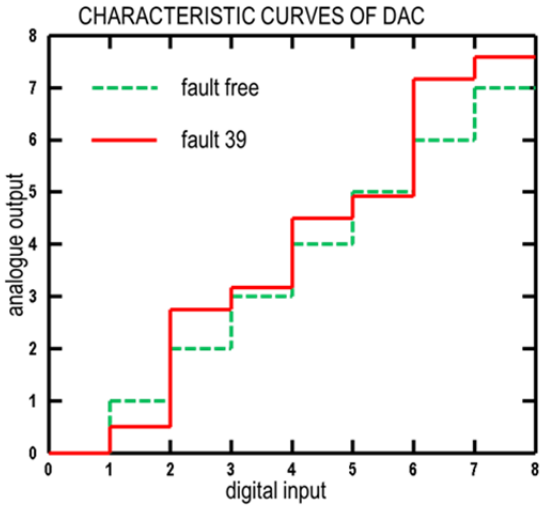


Abbildung 6: Kennlinie des DACs im fehlerfreien Fall (grün) und bei Fehler

Da alle Fehler der Fehlerliste in gleicher Weise behandelt werden, bietet sich eine Parallelisierung der Bearbeitung mit Cloud Computing geradezu an. Nach der Bildung von Einzelfehlerlisten werden modifizierte Netzlisten gebildet, in die einheitlich Parameter für die Monte-Carlo-Simulation eingefügt werden. Die Simulation mit ngSPICE und die Konvertierung in Kennlinienbündel werden in der Cloud durchgeführt.

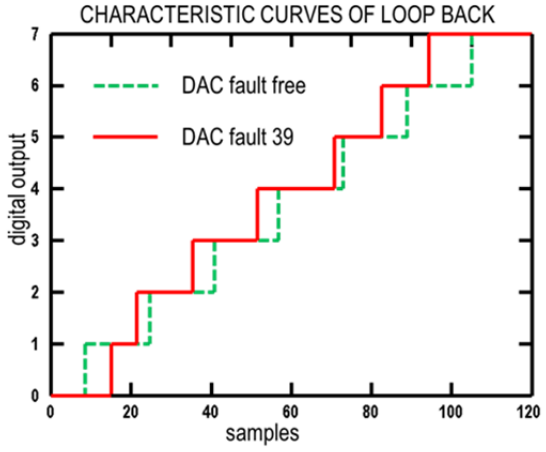


Abbildung 7: Ergebnis der Loop-Back-Simulation für DAC-Fehler 39

Für die Loop-Back-Simulation müssen alle Kennlinienbündel berechnet sein. Da die Loop-Back-Simulation besonders rechenintensiv ist, wird sie in der Cloud durchgeführt. Dabei können die Schritte 5 und 6 aus Abbildung 3 für jeden Fehler unmittelbar hintereinander

ausgeführt werden. Die Abbildungen 6 und 7 sowie die Tabellen 1-3 veranschaulichen die entstehenden Kennlinien, Kenngrößen und Fehlersignaturen.

In Abbildung 6 werden die mit ngSPICE simulierten Kennlinien für den fehlerfreien Fall und DAC-Fehler 39 gegenübergestellt. Bei Fehler 39 erhöht sich der in Abbildung 5 durch den Pfeil markierte Widerstand Rs1 auf den fünffachen Wert. Abbildung 7 zeigt das Ergebnis nach der Loop-Back-Simulation.

Daraus werden Kenngrößen, z.B. die in Tabelle 1 gezeigten berechnet. Diese Kenngrößen werden bei jeder Monte-Carlo-Simulation ermittelt. Zur Datenreduktion werden nur die Kenngrößen der Nominalparameter in der Fehlersignatur gespeichert. Alle anderen berechneten Kenngrößen werden zu Histogrammen zusammengefasst. Die Fehlersignatur für DAC-Fehler 39 wurde aus Platzgründen auf die Tabellen 2 und 3 aufgeteilt.

fnr	Δgain	offset	tue	dnl+	dnl-	inl+	inl-
0	0.00	0.00	0.00	0.00	0.00	0.00	0.00
39	0.05	0.03	0.69	0.67	-0.58	0.67	-0.61

Tabelle 1: Kenngrößen nach der Loop-Back-Simulation für fehlerfreien Fall und DAC-Fehler 39

fnr	char value	nom	min	max
39	Δgain	0.05	0.00	0.09
	offset	0.03	-0.22	0.23
	tue	0.69	0.33	1.59
	dnlmax	0.67	0.17	0.87
	dnlmin	-0.58	-0.59	-0.46
	inlmax	0.67	0.47	1.18
	inlmin	-0.61	-1.16	0.30

Tabelle 2: Fehlersignatur für DAC-Fehler 39 Teil 1, bestehend aus Nominal-, Maximal- und Minimalwert

fnr	char value	bin width	bins									
39	Δgain	0.01	1	2	5	11	18	25	19	12	6	2	0
	offset	0.05	1	3	7	14	20	22	17	10	5	1	0
	tue	0.18	0	0	0	1	8	61	15	5	4	4	2
	dnlmax	0.10	5	13	17	18	16	26	4	1	0	0	0
	dnlmin	0.02	0	0	0	0	0	74	0	0	18	0	8
	inlmax	0.10	0	0	0	6	20	27	31	12	2	1	1
	inlmin	0.19	0	0	1	3	10	42	9	11	13	9	2

Tabelle 3: Fehlersignatur für DAC-Fehler 39 Teil 2, bestehend aus den Histogrammen

Zur Validierung der Zeiteffektivität wurden die Rechnungen zunächst auf einem Cluster mit einem, fünf, zehn und 20 parallelen Jobs durchgeführt und dabei Zeitmessungen gemacht. Insgesamt wurden neben dem fehlerfreien Fall 1.197 Fehler im ADC und 223 Fehler im DAC berücksichtigt. Für jeden Fehler wurden zehn Monte-Carlo-Simulationen durchgeführt. Zusammen mit den Simulationen der Nominalwerte ergaben sich damit 15.642 Einzelsimulationen. In Abbildung 8 sind die Bearbeitungszeiten für verschiedene Zahlen von parallelen Jobs dargestellt. Für Step 1 und 2 ist die Trennung von Map- und Reduce-Phase markiert. Die Map-Phase (jeweils unterer Balkenabschnitt) ist nahezu ideal parallelisierbar. Bei der Reduce-Phase (jeweils obererer Balkenabschnitt) ist keine Parallelisierung möglich.

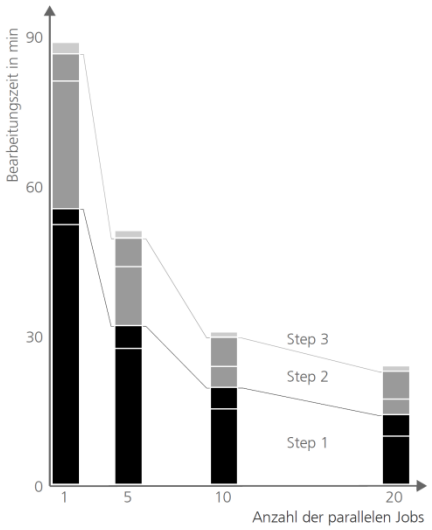


Abbildung 8: Dauer der Berechnungen für die verschiedenen Schritte: Step 1 (schwarz), Step 2 (grau) und Step 3 (hellgrau).

6 Zusammenfassung und Ausblick

Der im Beitrag vorgestellte Ansatz nutzt Cloud-Technologien für daten- und berechnungsintensive Aufgaben bei Entwurf und Test integrierter elektronischer Systeme. Für die Fehlerdiagnose von Mixed-Signal-Komponenten DAC und ADC konnte gezeigt werden, dass signifikante Verbesserungen der Diagnoseeffizienz in Form von Zeitgewinn, höherer Auflösung der Fehler (Netzwerk- anstelle Verhaltensebene) und Skalierbarkeit auf größere Verarbeitungsbreite der Komponenten erreichbar sind. Der Workflow zur Fehlerdiagnose basiert auf dem Ansatz, DAC und ADC in einer Loop-Back-Struktur zu analysieren und partitioniert die Diagnoseaufgabe in Teilschritte, die für eine Bearbeitung in der Cloud prädestiniert sind. Für die Organisation dieser Bearbeitung wurde das bei Fraunhofer entwickelte Werkzeug *GridWorker* weiterentwickelt. Als Software-Werkzeuge für die Bearbeitung der algorithmischen Teilaufgaben des eigentlichen Diagnoseprozesses kommen Public-Domain-Simulatoren und der bei Fraunhofer entwickelte analoge Fehlersimulator *aFSIM* zum Einsatz.

Ziel weiterführender Arbeiten ist es einerseits, die Skalierbarkeit unseres Ansatzes weiter zu verbessern und Voraussagen zur Laufzeit zu ermöglichen. Andererseits wird an einem nutzerfreundlichen Zugang gearbeitet, um bei Fraunhofer verfügbare Simulations-, Test- und Diagnoseverfahren (bzw. die darauf aufbauenden Werkzeuge) auch in der Cloud als Entwurfsdienstleistungen für externe Nutzer anbieten zu können.

Diese Arbeit wurde im Rahmen der BMBF-Projekte DIANA - Durchgängige Diagnosefähigkeit für Elektroniksysteme im Automobil (Kennzeichen 01M3188B) und OptiNum-Grid - Optimierung technischer Systeme und naturwissenschaftlicher Modelle mit Hilfe numerischer Simulationen im Grid (Kennzeichen 01IG0901D) gefördert.

7 Literatur

[1] *ADC and DAC Glossary*, MAXIM Application notes, Jul 22, 2012. http://www.maximintegrated.com/appnotes/index.mvp/id/641

[2] Gulbins, M., Vermeiren, V., Redlich, St.: *Fehlerdiagnose für AD- und DA-Wandler in einer Loop-Back-Struktur*. Dresdner Arbeitstagung Schaltungs- und Systementwurf (DASS 2012), Dresden, 3./4. Mai 2012

[3] Gulbins, M., Vermeiren, V., Redlich, St.: *Fehlerdiagnose für ADCs und DACs in einer Loop-Back-Struktur unter Einbeziehung von Parametervariationen*. Testmethoden und Zuverlässigkeit von Schaltungen und Systemen 25. GI/GMM/ITG-Workshop Dresden, 24. - 26. Februar 2013

[4] Straube, B.; Müller, B.; Vermeiren, W.; Hoffmann, C.; Sattler, S.: *Analogue fault simulation by aFSIM*. Design, Automation and Test in Europe Conference and Exhibition, DATE 2000 – User Forum, Paris, March 27-30, 2000, pp. 205-210.

[5] Schneider, A.: *Variantensimulation mit GridWorker*. ASIM-Workshop „Simulation technischer Systeme und Grundlagen und Methoden in Modellbildung und Simulation“, Krefeld, 24.-25. Februar 2011

[6] Dean, J.; Ghemawat, S.: *MapReduce*: *Simplified Data Processing on Large Clusters*. Sixth Symposium on Operating Systems Design and Implementation (OSDI '04), San Francisco, California, USA, December 6-8, 2004

[7] Schneider, A.: *Automatisierte Ressourcenbedarfsschätzung für Simulationsexperimente in der Cloud*. eingereicht zu Grid4Sys-Workshop „Grid-, Cloud- und Big-Data-Technologien für Systementwurf und -analyse“, Dresden, 27.-28. November 2013

Automatisierte Ressourcenbedarfsschätzung für Simulationsexperimente in der Cloud

André Schneider
Fraunhofer-Institut für Integrierte Schaltungen IIS, Institutsteil Entwurfsautomatisierung EAS
Zeunerstraße 38, 01069 Dresden, Deutschland
andre.schneider@eas.iis.fraunhofer.de

Kurzfassung

Mit Hilfe von Grid und Cloud Computing eröffnen sich heute vollkommen neue Möglichkeiten, komplexe, ressourcenintensive Berechnungen auszuführen. Skalierbarkeit und Elastizität spielen hierbei eine Schlüsselrolle. Die mit den Grids und Clouds gewonnene Flexibilität hat jedoch auch einen Preis. Während sich ein Anwender bei der Nutzung der eigenen, lokal installierten Infrastruktur keine oder wenige Gedanken über die Kosten für eine CPU-Stunde machen musste, wird bei kommerziellen Cloud-Anbietern jede in Anspruch genommene Ressource wie CPU, Speicher und Netzwerkbandbreite für den Zeitraum der Nutzung konsequent abgerechnet. Im vorliegenden Beitrag wird ein Ansatz vorgestellt, der für Simulationsexperimente auf Cluster-, Grid- und Cloud-Infrastrukturen den Ressourcenbedarf vorab automatisiert abschätzt. Der Anwender bekommt auf diese Weise beispielsweise eine Vorstellung von den zu erwartenden Bearbeitungszeiten und den dafür anfallenden Kosten. Die Ressourcenabschätzung wurde für das Framework *GridWorker* implementiert und mit Anwendungsbeispielen aus dem Systementwurf evaluiert.

1 Motivation

Mit Hilfe von Grid und Cloud Computing ist es heute möglich, einem Anwender Rechenkapazitäten in Größenordnungen, die bisher Rechenzentren vorbehalten waren, binnen weniger Minuten zur Verfügung zu stellen. Insbesondere der Aspekt der Skalierbarkeit und Elastizität ermöglicht es in vielen rechenintensiven Bereichen, Experimente in völlig neuen Dimensionen durchzuführen und dabei Probleme zu lösen, die in der Vergangenheit primär an einem zu hohen Bedarf an Rechenleistung gescheitert sind. Die mit den Grids und Clouds gewonnene Flexibilität hat jedoch auch einen Preis. Während sich ein Anwender bei der Nutzung der eigenen, lokal installierten Infrastruktur keine oder wenige Gedanken über die Kosten für eine CPU-Stunde machen musste, wird bei kommerziellen Cloud-Anbietern jede in Anspruch genommene Ressource wie CPU, Speicher und Netzwerkbandbreite für den Zeitraum der Nutzung konsequent abgerechnet. Der Anwender ist also künftig stärker gefordert, wenn es um einen sinnvollen Kosten-Nutzen-Kompromiss beim Zugriff auf On-Demand-Ressourcen geht. Und selbst dann, wenn mit Flat Rates alternative Kostenmodelle angeboten werden, wird es auch beim Cloud Computing Grenzen bei der Skalierbarkeit und Elastizität geben und der Anwender wird sich Gedanken machen müssen, wie er seine Experimente so konfiguriert, dass er mit minimalem Aufwand zum gewünschten Ergebnis kommt.

Im vorliegenden Beitrag wird ein Ansatz vorgestellt, mit dem der Ressourcenbedarf für Simulationsexperimente in der Cloud vorab geschätzt und so dem Anwender geholfen werden kann, angemessene Experimentkonfigurationen zu finden und das Kosten-Nutzen-Verhältnis sinnvoll zu gestalten.

Im Rahmen der BMWi-Förderinitiative *Trusted Cloud* werden gegenwärtig im Projekt *Cloud4E – Trusted Cloud Computing for Engineering* prototypische Dienste für diverse Ingenieursdisziplinen entwickelt. Ziel ist es unter anderem, Systementwerfern die Durchführung von komplexen Simulationsexperimenten in der Cloud über generische Dienste zu ermöglichen.

Simulationen spielen beim Entwurf und der Analyse von Systemen im Engineering-Bereich und auch in naturwissenschaftlichen Disziplinen wie der Physik und der Systembiologie eine zentrale Rolle. Dabei sind es weniger die Einzelsimulationen, die die aktuellen Herausforderungen bestimmen, sondern vielmehr neue Methoden, die meist auf einer Vielzahl von Einzelsimulationen aufbauen. So sind beispielsweise Monte-Carlo-Simulationen für statistische Analysen bei Empfindlichkeits- und Robustheitsuntersuchungen Voraussetzung für einen guten Systementwurf. Ebenso gehören Parameterstudien, Worst-Case-Analysen und Optimierung zum Alltag eines Entwurfsingenieurs. In der Regel werden für eine komplexe Entwurfs- oder Analyseaufgabe Experimente definiert, die die Durchführung vieler Hundert, Tausend oder gar Millionen Einzelsimulationen erfordern.

Damit derartige ressourcenintensiven Aufgaben für den Anwender beherrschbar bleiben, wurden in den letzten Jahren Middleware-Lösungen, Frameworks und Dienste entwickelt, die die Ausführung sehr vieler Berechnungen auf Cluster-, Grid- und Cloud-Infrastrukturen organisieren und automatisieren. Mit *GridWorker* [5][9] steht beispielsweise ein Framework zur Verfügung, das die parallele, fehlertolerante Ausführung umfangreicher Variantensimulationen auf Basis des Map/Reduce-Paradigmas gestattet. Der Anwender kann mit Hilfe einer sehr einfachen Syntax kompakt und flexibel Simulationsexperimente beschreiben [11], die über *GridWorker* vollständig automatisiert

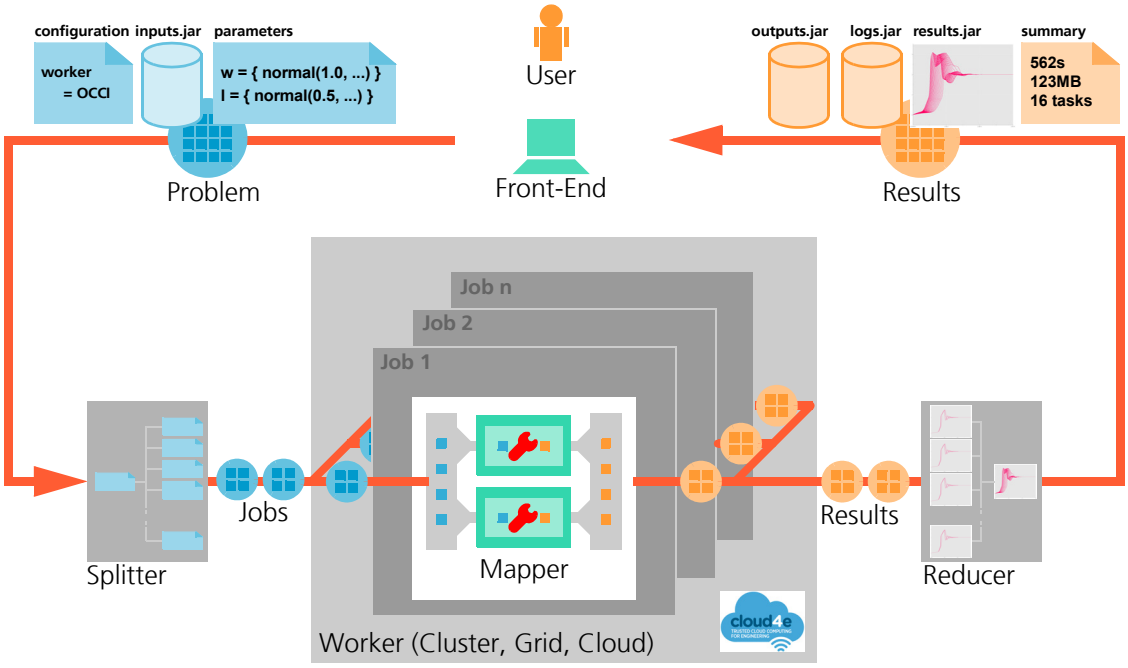


Bild 1. Variantensimulation mit GridWorker

auf den jeweils verfügbaren Compute-Ressourcen ausgeführt werden. *GridWorker* steuert hierfür über Adapter – sogenannte *Worker* – lokal im LAN erreichbare Rechner, Cluster (DRMAA, SGE, PBS, LSF, ...), Grids (Globus Toolkit, GridWay, ...) oder Clouds (OpenStack, OCCI, ...) an und organisiert die parallele Bearbeitung auf Job- und Task-Ebene vollkommen transparent für den Anwender. Die Ergebnisse werden in Containern zusammengefasst und an den Anwender direkt oder als Referenz zurückgegeben.

In Bild 2 ist die Beschreibung für eine Monte-Carlo-Simulation im Rahmen einer Fehlerdiagnose [6] illustriert.

```
# total number of tasks: 10,000
fault = { 0:1:100 }
R1 = [ normal(1.0, 0.01, 100) ]
R2 = [ normal(7.5, 0.05, 100) ]

# total number of tasks: 1,000,000
fault = { 0:1:1000 }
R1 = [ normal(1.0, 0.01, 1000) ]
R2 = [ normal(7.5, 0.05, 1000) ]
```

Bild 2. Beispiele für Parameterspezifikationen für eine kombinierte Fehler-/Monte-Carlo-Simulation

Im Beispiel müssen insgesamt 10.000 Einzelsimulationen – für 100 Fehler je 100 Monte-Carlo-Varianten – durchgeführt werden, wofür beispielsweise auf einem 128-Core-Compute-Cluster etwa 4 Minuten benötigt werden (vgl. Bild 3). Erweitert nun der Anwender sein Experiment und nimmt weitere Fehler und Monte-Carlo-Varianten hinzu, wächst die Zahl der durchzuführenden Einzelsimulationen bereits auf eine Million und der Anwender müsste vermutlich nicht nur 4 Minuten sondern etwa 6 Stunden auf das Ergebnis warten.

Kritisch ist, dass ein Anwender die Experimentkonfiguration im Hinblick auf den erforderlichen Ressourcenbedarf oft nicht hinreichend überschaut. Aufgrund der kompakten Syntax können kleine Änderungen mitunter große Auswirkungen haben, die sich dann bei der Nutzung kommerzieller Cloud-Angebote dramatisch auf die Kosten niederschlagen. Es ist daher zwingend notwendig, Anwendern Hilfswerkzeuge zur Verfügung zu stellen, mit denen sie sich beim Experimentieren jederzeit einen Überblick über die erwartete Ressourcennutzung beschaffen können. Nur wer Aufwand und Kosten im Blick behält, wird letztlich von der Flexibilität, Skalierbarkeit und Elastizität beim Experimentieren unter Nutzung von Cloud-Diensten profitieren.

2 Ansätze für Ressourcenschätzung

In der Literatur werden seit vielen Jahren unterschiedliche Ansätze für die Ressourcenschätzung beim Cloud Computing diskutiert [7]. Oft wird hierbei aus der Perspektive des Ressourcenanbieters argumentiert. Ziele sind entsprechend eine gleichmäßige oder energieeffiziente Auslastung der verfügbaren Ressourcen und die Vermeidung von Engpässen oder Leerlaufzeiten.

Aus der Perspektive des Anwenders ergeben sich häufig andere Anforderungen. Beispielsweise können die folgenden beiden Ziele relevant sein:

- Die benötigte Gesamtzeit für die Ausführung eines Simulationsexperiments sollte minimal sein.
- Die Kosten dürfen – bei Nutzung kommerzieller Ressourcen – gewisse Grenzen nicht überschreiten.

Das erste Ziel ist sehr strikt und gilt vermutlich bei den meisten Anwendern. Das zweite Ziel dagegen berücksichtigt, dass es in kreativen Bereichen wie dem Systementwurf oder generell in Forschung und Entwicklung hinreichend große Freiräume zum Experimentieren geben muss

und zu strikte Kostensenkungsregularien sinnvolles Arbeiten unmöglich machen. Hier wird eher ein Aufwand-Nutzen-Kompromiss angestrebt. Letzlich ergibt sich aus den beiden Zielen, für welche Größen quantitative Schätzungen benötigt werden. Im Kontext Cloud Computing zählen dazu primär der Gesamtzeitbedarf und die Gesamtkosten pro Simulationsexperiment.

Diese beiden Größen werden bei allen Backend-Technologien (Multi-/Many-Core-Computer, Cluster, Grid, Cloud) durch folgende Ressourcenkonfigurationsparameter bestimmt:

- Anzahl der Jobs, wobei pro CPUs, Cluster-Knoten oder VM (virtuelle Maschine) in der Regel *ein* Job läuft;
- Anzahl der parallelen Threads pro Job, die unter anderem von der Anzahl der physisch verfügbaren Cores abhängt.

Die beiden Parameter müssen unter Berücksichtigung konkret verfügbarer Ressourcen und deren Ausstattung gewählt werden. Einfluss haben unter anderem die folgenden Größen:

- Hauptspeicherbedarf pro Job, der sich aus dem Bedarf für eine Einzelsimulation, der Anzahl der parallelen Threads und dem Job-Overhead ergibt;
- Plattenspeicherbedarf pro Job für die temporäre Datenspeicherung während der Jobausführung
- Netzwerkbandbreite für die Übertragung von Eingangs- und Ausgangs- bzw. Ergebnisdaten.

Will man die genannten Größen in quantifizierbare Relationen setzen, sind grundsätzlich die folgende Vorgehensweisen möglich:

- Die komplette Ressourceninfrastruktur wird in einem Modell erfasst, mit dem über Simulation quantitative Aussagen zur Auslastung und zu den Kosten (Gesamtzeitbedarf, Speicherbedarf, etc.) abgeleitet werden können. Ein Beispiel hierfür wird in [2] mit dem Simulator *CloudSim* beschrieben. Der Aufwand ist groß, zumal auch das Verhalten einzelner Simulationsexperimente mit modelliert werden muss. Diese Option ist für Provider geeignet, die ihre Ressourcen energie- und kosteneffizient betreiben wollen.
- Aus Anwenderperspektive ist es eher interessant, grundsätzlich bei der Durchführung von Simulationsexperimenten den jeweils aktuellen Ressourcenverbrauch zu erfassen und zu protokollieren. Die so gesammelten Daten können später ausgewertet und als Grundlage zur Ressourcenbedarfsschätzung für künftige, ähnliche Experimente dienen.
- Eine weitere Möglichkeit bietet das testweise Ausführen kleiner, abgerüsteter Experimente in Verbindung mit der Ressourcenverbrauchserfassung. Hier besteht die Möglichkeit, unmittelbar vor einer Ausführung eines geplanten Experiments auf den Zielressourcen unter den jeweils aktuell vorliegenden Lastbedingungen sogenannte *probes* auszuführen und die dabei für die konkrete Experimentkonfiguration Messdaten zu gewinnen, die wiederum Grundlage für weitere Schätzungen sein können.

Für *GridWorker* wurde der letztgenannte Ansatz implementiert und evaluiert. Die Details werden im Folgenden beschrieben.

3 *GridWorker*-Kommando *probe*

Um *GridWorker*-Anwendern eine einfache Möglichkeit zu bieten, für Simulationsexperimente in der Cloud oder auf anderen Compute-Backends den Ressourcenbedarf vorab zu schätzen, wurde das Kommando *probe* implementiert:

```
gridworker probe
             [ -n <jobs>.<threads> ]
             [ -t <timeout> ]
```

Der Anwender bereitet wie gewohnt sein Simulationsexperiment vor und erstellt eine Konfiguration. Im Anschluss kann mit *probe* mittels verschiedener Messungen in Kombination mit einer Extrapolation eine Übersicht zum erwarteten Ressourcenbedarf ausgegeben werden. Ein Beispiel für ein *probe*-Ergebnis ist in Bild 3 dargestellt. Für eine Variantensimulation müssen ca. 100000 Parameterkombinationen simuliert werden. Mit *probe* kann binnen weniger Minuten ermittelt werden, dass bei einer Verteilung des Problems auf 26 Knoten eines Clusters mit Quadcore-CPUs (26 Jobs zu je 4 Threads) eine reichliche halbe Stunde benötigt wird. Eine genauere Betrachtung der Messer-

```
lina5> gridworker probe -n 26.4

MEASUREMENTS
================================================================================
tasks                1     2     5    10    20    50   100   200   500  1000
jobs.threads
--------------------------------------------------------------------------------
        26.4       32s   27s   34s   32s   31s   28s   31s   32s   46s   51s
================================================================================

ESTIMATIONS
================================================================================
       tasks         1       10      100     1000    10000   100000 1000000
jobs.threads
--------------------------------------------------------------------------------
        26.4       30s      31s      33s      52s       4m      37m    6h
================================================================================
```

Bild 3. Mess- und Schätzergebnisse von *GridWorker probe*

gebnisse zeigt zudem, dass eine Mindestlaufzeit von einer halben Minute gibt, die *GridWorker* für die Organisation der Jobs generell benötigt.

Der Algorithmus, der *probe* zugrunde liegt, ist Folgender: *GridWorker* führt nacheinander Simulationsexperimente aus und erfasst dabei den jeweils benötigten Ressourcenbedarf. Es wird mit einer einzelnen Task, was einer einzelnen Simulation entspricht, begonnen und anschließend wird die Task-Anzahl schrittweise vergrößert. Die Task-Zahlen sind dabei konfigurierbar. Entscheidend ist, dass *GridWorker* nach jedem *probe*-Experiment grob abschätzen kann, wie lange das nächste Experiment dauern wird. Der Anwender kann beim *probe*-Kommando optional eine Maximalzeit (*timeout*) angeben, nach der die Messungen beendet werden sollen. Damit besteht die Möglichkeit, für Experimente, für die keinerlei Erfahrungen über die erforderliche Simulationszeit vorliegen, innerhalb einer definierten Zeitspanne quantitative Aussagen zum erwarteten Ressourcenbedarf zu erhalten. Es ist einzusehen, dass bei langen Simulationszeiten auch für die *probe*-Phase mehr Zeit eingeräumt werden muss oder aber keine quantitativen Aussagen möglich sind, was in diesem Fall ein Indiz für den Anwender ist, dass der Ressourcenbedarf offenbar (extrem) groß ist und eventuell das Simulationsmodell vereinfacht oder der Parameterraum bei Variantensimulationsexperimenten eingeschränkt werden muss.

Auf Basis der Messwerte werden zusätzlich mittels Extrapolation Schätzwerte für weitere Experimentkonfigurationen bestimmt. Hierfür werden intern verschiedene Regressionsverfahren eingesetzt. Es ist geplant, für ein weiteres *GridWorker*-Kommando *estimate* die Schätzverfahren weiter zu verbessern.

```
#jobs = <jobs>
#threads = <threads>

for #tasks in 1 2 5 10 20 50 ...
do
   begin measurement

   solve problem using #jobs with
   #threads per job

   end measurement

   if <timeout> exceeded then break
done

print measurement results
calculate estimations
print estimation results
```

Bild 4. *probe*-Algorithmus zur Messung und Schätzung des Ressourcenbedarfs

4 Evaluation

Um die Funktionsweise des Kommandos *probe* nachzuweisen und die implementierten Mess- und Schätzalgorithmen hinsichtlich Praxistauglichkeit zu bewerten, wurden zahlreiche Anwendungsbeispiele getestet.

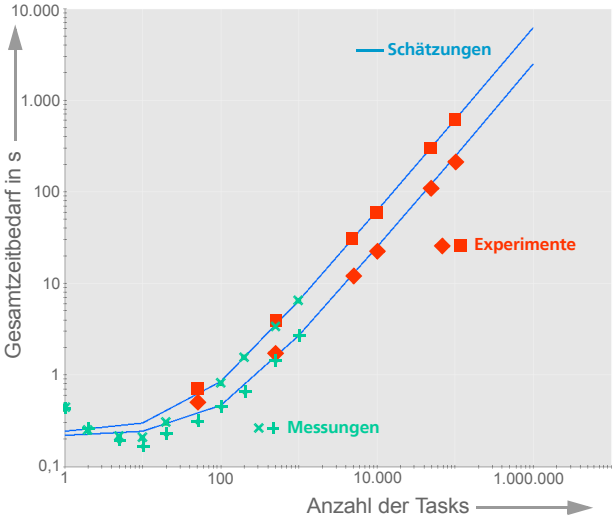


Bild 5. Ergebnisse (Gesamtsimulationszeit) des Kommandos *probe* für zwei Experimentkonfigurationen für die Parameterstudie zur Filterschaltung (grüne Kreuze: *probe*-Messungen, blaue Linie: *probe*-Schätzung, rote Quadrate: reale Experimente)

In Bild 5 sind die Ergebnisse für zwei Konfigurationen für ein Simulationsexperiment (Parameterstudie für eine Filterschaltung) dargestellt. Hierfür wurden Simulationsexperimente auf Multi-Core-Rechnern sowie auf einem 128-Core-Compute-Cluster durchgeführt.

Die Schätzung stimmt sehr gut mit den bei realen Experimenten gemessenen Simulationszeiten überein.

Bei einem zweiten Beispiel wurden der Gesamtzeitbedarf für die in [6] beschriebene Fehlerdiagnose mittels *GridWorker probe* ermittelt. Die Experimente wurden dabei auf unterschiedlichen Compute-Ressourcen durchgeführt: ein Laptop mit Dual-Core-CPU, zwei Compute-Server mit 8 und 16 CPU-Cores, ein 128-Core-Compute-Cluster und eine OpenStack-Test-Cloud mit drei VM-Instanzen. Ziel war es, die Eignung von *probe* in den unterschiedlichen Leistungsklassen der vier Back-ends zu prüfen. Da *GridWorker* über Adapter, die sogenannten *Worker*, die verschiedenen Compute-Ressourcen direkt und für den Nutzer transparent ansprechen kann, wurde jeweils das gleiche Beispielszenario verwendet und lediglich der Konfigurationsparameter *gridworker.worker* angepasst.

Beim Fehlerdiagnose-Beispiel [6] werden für ein SPICE-Modell mit ADC-DAC-Loop-Back-Struktur (ADC – analog digital converter, DAC – digital analog converter) für *1420* Fehlersituationen (*1197* ADC-Fehler + *223* DAC-Fehler) jeweils *1000* Monte-Carlo-Simulationen, bei denen Schaltungsparameter zufällig variiert werden, simuliert. Berücksichtigt man die beiden fehlerfreien Situationen und das jeweilige Nominalverhalten, müssen im konkreten Fall *1422 x 1001 = 1.423.422* Einzelsimulationen durchgeführt werden. Damit die *probe*-Evaluierung in einer akzeptablen Zeit und mit einer Vielzahl von Experimenten durchgeführt werden konnte, wurde die Zahl der Monte-Carlo-Simulationen auf *10* begrenzt. Pro Experi-

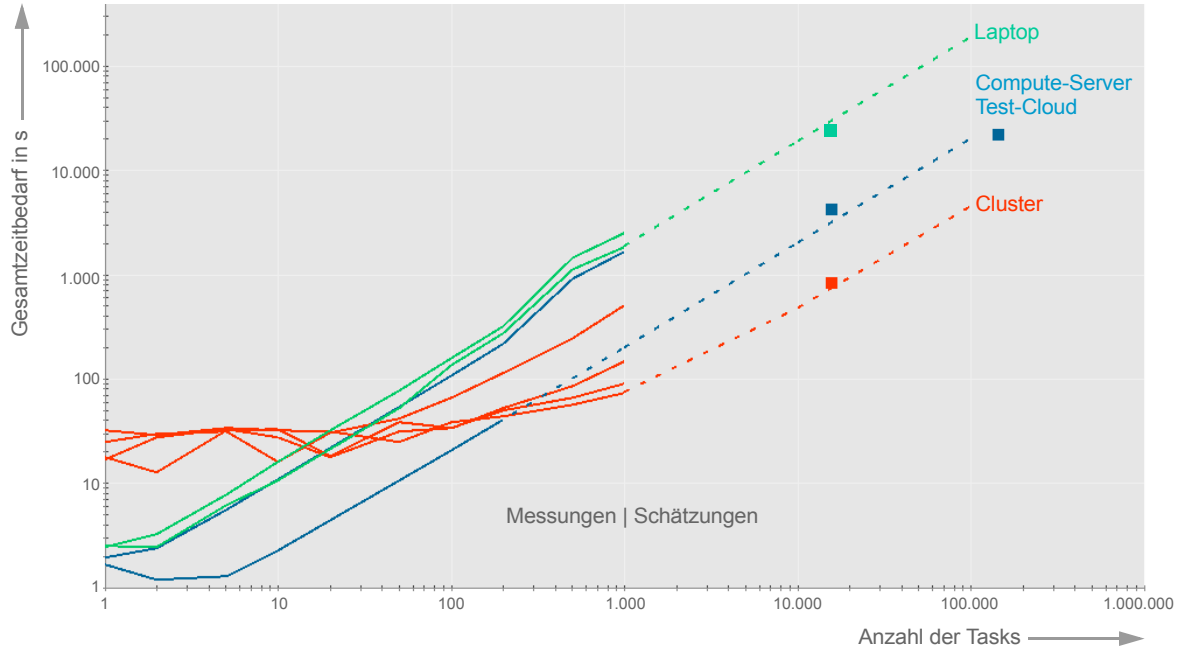


Bild 6. Evaluierungsergebnisse für das Beispiel Fehlerdiagnose. Die Linien im linken Diagrammbereich zeigen *probe*-Messungen für unterschiedliche Parallelisierungsgrade, die Strichlinien im rechten Diagrammbereich zeigen die *probe*-Schätzungen für je einen typischen Parallelisierungsgrad pro Backend. Die Quadrate repräsentieren konkrete Simulationsexperimente.

ment mussten damit nur noch *1422 x 11 = 15642* Simulationen ausgeführt werden.

In Bild 6 sind die Evaluierungsergebnisse zusammengefasst. Für Back-end-Plattformen in drei unterschiedlichen Leistungsklassen wurde mittels *GridWorker probe* zunächst der Gesamtzeitbedarf automatisiert geschätz. Anschließend wurden einzelne konkrete Simulationsexperimente durchgeführt. Teilweise konnten – analog zu den Ergebnissen in Bild 5 – recht gute Übereinstimmungen für die geschätzten und tatsächlich benötigten Zeiten erzielt werden. Teilweise traten noch Abweichungen auf, die im weiteren Projektverlauf näher untersucht werden. Deutlich sichtbar wird in Bild 6 auch der enorme Zeitaufwand für größere Simulationsexperimente mit sehr vielen Einzelsimulationen. Werden die Vorteile von Cluster und Grid Computing mit denen von Cloud Computing verbunden, können die Simulationszeiten teilweise um Größenordnungen reduziert werden, ohne dass lokal eine Hochleistungsinfrastruktur vorgehalten werden muss.

Insgesamt arbeitet das *probe*-Verfahren bereits in der aktuellen Implementierung sehr robust und hinreichend zuverlässig und stößt, nicht zuletzt aufgrund der einfachen Bedienbarkeit, auf sehr gute Resonanz bei den Anwendern.

5 Zusammenfassung

Mit dem Ausbau leistungsfähiger, skalierbarer Compute-Ressourcen auf Grid- und Cloud-Basis ergeben sich vollkommen neue Möglichkeiten für die Bearbeitung komplexer, rechenintensiver Simulationsexperimente. Im Rahmen des Projektes *Cloud4E* [3] werden hierfür prototypische Dienste entwickelt, die eine Erschließung dieser neuen Möglichkeiten für Aufgaben aus dem Ingenieursbereich vereinfachen sollen.

```
ESTIMATIONS
================================================================================
       tasks          1       10      100     1000    10000   100000 1000000
jobs.threads
--------------------------------------------------------------------------------
        20.3        31s       31s       35s       76s        8m       75m   12h
        10.3        26s       26s       33s       96s       12m      117m   19h
         5.3        24s       25s       36s        2m       20m        3h   34h
         2.3        27s       30s       54s        4m       45m        7h    3d
         1.3        20s       25s       68s        8m       79m       13h    5d
================================================================================
```

Bild 7. *probe*-Zeitschätzungen für das Fehlerdiagnose-Experiment [6] mit bis zu einer Million Einzelsimulationen bei einer Aufteilung auf 1 bis 20 parallele Jobs mit je 3 Threads pro Job.

Eine Herausforderung besteht für Anwender darin, bei der Konfiguration von Simulationsexperimenten vorab den Ressourcenbedarf abzuschätzen. Neben modellbasierten Ansätzen erweisen sich vor allem Ansätze auf der Basis von Probemessungen vielversprechend. Der Beitrag stellte hierfür eine für das *GridWorker*-Framework implementierte Methode *probe* vor, bei der zunächst einfache, danach schrittweise komplexere Simulationsexperimente durchgeführt werden und die dabei gewonnenen Messergebnisse für den Ressourcenverbrauch als Grundlage für eine Ressourcenbedarfsschätzung dienen. Das Verfahren wurde erfolgreich an praxisrelevanten Beispielen evaluiert.

Es ist geplant, den derzeit für die Gesamtsimulationszeit realisierten Ansatz auf weitere Ressourcenindikatoren wie Speicherbedarf und Netzwerkbandbreite auszudehnen und schließlich über hinterlegte Preismodelle auch die erwarteten finanziellen Kosten mit in die Schätzung einzubeziehen.

Die vorliegende Arbeit ist im Rahmen des Verbundprojektes „Cloud4E – Trusted Cloud Compting for Engineering“, das vom Bundesministerium für Wirtschaft und Technologie unter dem Kennzeichen 01MD11053 gefördert wird, entstanden.

Literatur

[1] Apache Hadoop: http://hadoop.apache.org

[2] Buyya, R.; Ranjan, R.; Calheiros, R.N.: Modeling and Simulation of Scalable Cloud Computing Environments and the CloudSim Toolkit: Challanges and Opportunities. IEEE Int. Conference on High Performance Computing & Simulation, HPCS '09, Leipzig, Germany, June 21-24, 2009, pp. 1-11

[3] Cloud4E – Trusted Cloud Computing for Engineering. http://www.cloud4e.de

[4] Dean, J.; Ghemawat, S.: MapReduce: Simplified Data Processing on Large Clusters. Sixth Symposium on Operating Systems Design and Implementation (OSDI '04), San Francisco, California, USA, December 6-8, 2004

[5] *GridWorker*: http://www.gridworker.de

[6] Gulbins, M.; Schneider, A.; Rülke, S.: Ein Cloud-basierter Workflow für die effektive Fehlerdiagnose von Loop-Back-Strukturen. Grid4Sys-Workshop „Grid-, Cloud- und Big-Data-Technologien für Systementwurf und -analyse“, Dresden, 27.-28. November 2013

[7] Iosup, A.; Ostermann, S.; Yigitbasi, M.N.; Prodan, R.; Fahringer, T.; Epema, D.H.J.: Performance Analysis of Cloud Computing Services for Many-Tasks Scientific Computing. IEEE Trans. on Parallel and Distributed Systems, Vol. 22, No. 6, June 2011

[8] Limmer, S.; Schneider, A.; Boehme, C.; Fey, D.; Schmitz, S.; Graupner, A.; Sülzle, M.: Services for Numerical Simulations and Optimizations in Grids. In: Dr. Nitin; Prof. Satya Prakash Ghrera (Ed.): Proceedings of 2012 2nd IEEE International Conference on Parallel, Distributed and Grid Computing (PDGC), Waknaghat, India, December 6-8, 2012, pp. 214 -219

[9] Schneider, A.: Variantensimulation mit *GridWorker*. ASIM-Workshop „Simulation technischer Systeme und Grundlagen und Methoden in Modellbildung und Simulation“, Krefeld, 24.-25. Februar 2011

[10] Schneider, A.; Dietrich, M.: Variantensimulation im Grid. Workshop „Numerische Simulationen im D-Grid“, Göttingen, 26. November 2009

[11] Schneider, A.; Schneider, P.; Dietrich, M.: Grid-spezifische Problembeschreibung und -aufbereitung im Systementwurf. Workshop „Grid-Technologie für den Entwurf technischer Systeme“, Dresden, 12. Oktober 2007

[12] Schneider, A.; Schneider, P.; Schwarz, P.; Dietrich, M.: Grid-Computing – Anwendungsszenarien für den Systementwurf. 2. Workshop „Grid-Technologie für den Entwurf technischer Systeme“, Dresden, 6.-7. April 2006

1. Aktuelle und zukünftige Aktivitäten (Bericht des Sprechers)

Die 31. Ausgabe der PARS-Mitteilungen enthält die Beiträge des 11. PASA-Workshops sowie des 5. Grid4Sys-Workshops, die die wesentliche Aktivität der Fachgruppe im Jahr 2014 darstellen.

Der 11. PASA-Workshop fand am 25. Und 26. Februar 2014 in Lübeck im Rahmen der Konferenz ARCS 2014 statt. Der Workshop wurde erneut gemeinsam mit der Fachgruppe ALGO organisiert. Mehr als 30 Teilnehmer fanden sich ein. Am Morgen des ersten Tages präsentierte Herr Keller (FernUni Hagen) ein Tutorial „Energy Challenges in Processors“. Bis zum Mittag des zweiten Tages folgten 11 Vorträge und ein eingeladener Vortrag, die ein umfangreiches Themenspektrum abdeckten. Herrn Professor Wanka (Univ. Erlangen-Nürnberg) sei für die Co-Organisation des Workshops gedankt.

Den zum neunten Mal ausgeschriebenen und mit 500 € dotierten Nachwuchspreis erhielt in diesem Jahr Frau Anna Lührs geb. Westhoff (FZ Jülich). Herr Tobias Fleig (KIT) und Herr Johannes Hofmann (Univ. Erlangen-Nürnberg) erhielten Buchpreise.

v.l.n.r.: J. Keller (FG-Sprecher), J. Hofmann, Preisträgerin A. Westhoff, T. Fleig. Bild: E. Maehle

Während des PASA-Workshops fand auch eine Sitzung des PARS-Leitungsgremiums statt. Die dort besprochenen Regeln für die Aufnahme von PARS-Bänden im GI-Publikationsportal sind mittlerweile umgesetzt, die PARS-Bände der letzten Jahre sind im Portal eingestellt.

Unser nächster Workshop ist der

26. PARS-Workshop am 7. und 8. Mai 2015 in Potsdam.

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 2015.

Hagen, im September 2014
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

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, gemeinsam mit ITG-FA 4.1 und 4.4 und mit GI/ITG FG Rechnernetze, aber aus Kostengründen nicht erschienen. Es wird deshalb stattdessen auf den Tagungsband des Instituts für Mikroelektronik Stuttgart hingewiesen.)

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)

4. Mitteilungen (ISSN 0177-0454)

Bisher sind 31 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.

28TH GI/ITG International Conference on Architecture of Computing Systems
Porto, Portugal 24-27 March 2015

ARCS 2015

THIS YEAR'S FOCUS: Reconciling Parallelism and Predictability in Mixed-Critical Systems

CALL FOR PAPERS

The ARCS series of conferences has over 30 years of tradition reporting high quality results in computer architecture and operating systems research. The focus of the 2015 conference will be on **Reconciling Parallelism and Predictability in Mixed-Critical Systems**. Like the previous conferences in this series, it continues to be an important forum for computer architecture research.

The proceedings of ARCS 2015 will be published in the Springer Lecture Notes on Computer Science (LNCS) series. After the conference, authors of selected papers will be invited to submit an extended version of their contribution for publication in a special issue of the Journal of Systems Architecture. Also, a best paper and best presentation award will be provided at the conference.

Important Dates

Paper submission deadline (extended):	**October 17, 2014**
Workshop/tutorial proposals:	**November 3, 2014**
Notification of acceptance:	**December 1, 2014**
Camera-ready papers:	**December 15, 2014**

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

- Multi-/many-core architectures, memory systems, and interconnection networks.
- Models and tools for multi-/many-core systems including but not limited to programming models, runtime systems, middleware, and verification.
- Design, methods, and hardware and software architectures for mixed-critical systems.
- Architectures and design methods/tools for robust, fault-tolerant, real-time embedded systems.
- Generic and application-specific accelerators in heterogeneous architectures.
- Cyber-physical systems and distributed computing architectures.
- Adaptive system architectures such as reconfigurable systems in hardware and software.
- Organic and Autonomic Computing including both theoretical and practical results on self-organization, self-configuration, self-optimization, self-healing, and self-protection techniques.
- Operating Systems including but not limited to scheduling, memory management, power management, and RTOS.
- 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.
- Grid and cloud computing.

Submissions

Regular papers should be submitted via the link provided on the conference website, formatted according to the Springer LNCS style and not exceeding 12 pages.

Workshop and Tutorial proposals within the technical scope of the conference are solicited. Those should be submitted by email directly to the corresponding chair (address at the website).

http://www.cister.isep.ipp.pt/arcs2015

Organizing Committee

General Co-Chairs
Luís Miguel Pinho, CISTER/INESC-TEC, ISEP, Portugal
Wolfgang Karl, Karlsruhe Institute of Technology, Germany

Program Co-Chairs
Albert Cohen, INRIA, France
Uwe Brinkschulte, Universität Frankfurt, Germany

Workshop and Tutorial Co-Chair
João Cardoso, FEUP/University of Porto, Portugal

Publication Chair
Thilo Pionteck, Hamburg University of Technology, Germany

Industrial Liaison Co-Chairs
Sascha Uhrig, Technische Universität Dortmund, Germany
David Pereira, CISTER/INESC-TEC, ISEP, Portugal

Poster Co-Chairs
Florian Kluge, University of Augsburg, Germany
Patrick Meumeu Yomsi, CISTER/INESC-TEC, ISEP, Portugal

Publicity Chair
Vincent Nelis, CISTER/INESC-TEC, ISEP, Portugal

Local Organization Chair
Luis Ferreira, CISTER/INESC-TEC, ISEP, Portugal

Program Committee

Michael Beigl, Karlsruhe Institute of Technology, Germany
Mladen Berekovic, Technische Universität Braunschweig, Germany
Simon Bliudze, Rigorous System Design Laboratory (RiSD), EPFL, Switzerland
Florian Brandner, École Nationale Supérieure de Techniques Avancées (ENSTA ParisTech), France
Jürgen Brehm, Leibniz Universität Hannover, Germany
David Broman, KTH Royal Institute of Technology, Sweden, and UC Berkeley, USA
João M.P. Cardoso, University of Porto, Portugal
Luigi Carro, Instituto de Informatica, Universidade Federal do Rio Grande do Sul, Brasil
Koen De Bosschere, Ghent University, Belgium
Nikitas Dimopoulos, University of Victoria, Canada
Ahmed El-Mahdy, Egypt-Japan University for Science and Technology (E-JUST), Alexandria, Egypt
Fabrizio Ferrandi, Politecnico di Milano-DEIB, Italy
Dietmar Fey, Friedrich-Alexander-University Erlangen-Nürnberg, Germany
Pierfrancesco Foglia, Università di Pisa, Italy
William Fornaciari, Politecnico di Milano, Italy
Björn Franke, University of Edinburgh, United Kingdom
Roberto Giorgi, Università di Siena, ITALY
Daniel Gracia Perez, Thales Research & Technology, France
Jan Haase, University of the Federal Armed Forces Hamburg, Germany
Jörg Hähner, Universität Augsburg, Germany
Jörg Henkel, Karlsruhe Institute of Technology, Germany
Andreas Herkersdorf, TU München, Germany
Christian Hochberger, Technische Universität Darmstadt, Germany
Michael Hübner, Chair for Embedded Systems in Information Technology (ESIT), Ruhr-University of Bochum, Germany
Gert Jervan, Tallinn University of Technology, Estonia
Ben Juurlink, Technische Universität Berlin, Germany
Christos Kartsaklis, Oak Ridge National Laboratory, USA
Jörg Keller, FernUniversität in Hagen, Germany
Raimund Kirner, University of Hertfordshire, United Kingdom
Andreas Koch, Embedded Systems and Applications Group, TU Darmstadt, Germany
Hana Kubatova, Czech Technical University in Prague, Czech Republic
Olaf Landsiedel, Chalmers University of Technology, Sweden
Paul Lukowicz, Universität Passau, Germany
Erik Maehle, Institut für Technische Informatik, Germany
Christian Müller-Schloer, Leibniz Universität Hannover, Germany
Alex Orailoglu, University of California, San Diego , USA
Carlos Eduardo Pereira, Universidade Federal do Rio Grande do Sul, Brazil
Thilo Pionteck, Universität zu Lübeck, Germany
Pascal Sainrat, Université Toulouse III, France
Toshinori Sato, Fukuoka University, Japan
Martin Schulz, Lawrence Livermore National Laboratory, USA
Karsten Schwan, Georgia Institute of Technology, USA
Leonel Sousa, Instituto Superior Técnico (IST), Universidade de Lisboa, Portugal
Rainer Spallek, Technische Universität Dresden, Germany
Olaf Spinczyk, Technische Universität Dortmund, Germany
Benno Stabernack, Fraunhofer Institut für Nachrichtentechnik, Germany
Walter Stechele, Technical University of Munich (TUM), Germany
Djamshid Tavangarian, Universität Rostock, Germany
Jürgen Teich, University of Erlangen-Nuremberg, Germany
Martin Törngren, KTH Royal Institute of Technology, Sweden
Eduardo Tovar, CISTER/INESC-TEC, ISEP, Portugal
Pedro Trancoso, University of Cyprus, Cyprus
Carsten Trinitis, Technische Universität München, Germany
Sascha Uhrig, Technische Universität Dortmund, Germany
Theo Ungerer, Universität Augsburg, Germany
Hans Vandierendonck, Queen's University Belfast, United Kingdom
Stephane Vialle, SUPELEC & UMI GT-CNRS 2958, France
Lucian Vintan, "Lucian Blaga" University of Sibiu, Romania
Klaus Waldschmidt, Universität Frankfurt am Main, Germany
Wolfgang Karl, Karlsruhe Institute of Technology, Germany
Stephan Wong, Delft University of Technology, The Netherlands

GESELLSCHAFT FÜR INFORMATIK E.V.

PARALLEL-ALGORITHMEN, -RECHNERSTRUKTUREN UND -SYSTEMSOFTWARE

PARS

ITG 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 **1. März 2015** in elektronischer Form unter folgendem Link einzureichen:
http://www.easychair.org/conferences/?conf=pars2015

Benachrichtigung der Autoren bis **1. 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 • 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

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, Univ. Karlsruhe
Prof. Dr. Jörg Keller, Sprecher, FernUniversität in Hagen
Prof. Dr. Christian Lengauer, Univ. Passau
Prof. Dr.-Ing. Erik Maehle, Universität zu Lübeck
Prof. Dr. Ernst W. Mayr, TU München
Prof. Dr. Wolfgang E. Nagel, TU Dresden
Dr. Karl Dieter Reinartz, Ehrenvorsitzender, Univ. Erlangen-Nürnberg
Prof. Dr. Hartmut Schmeck, Univ. Karlsruhe
Prof. Dr. Peter Sobe, HTW Dresden
Prof. Dr. Theo Ungerer, Univ. Augsburg
Prof. Dr. Rolf Wanka, Univ. Erlangen-Nürnberg
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/