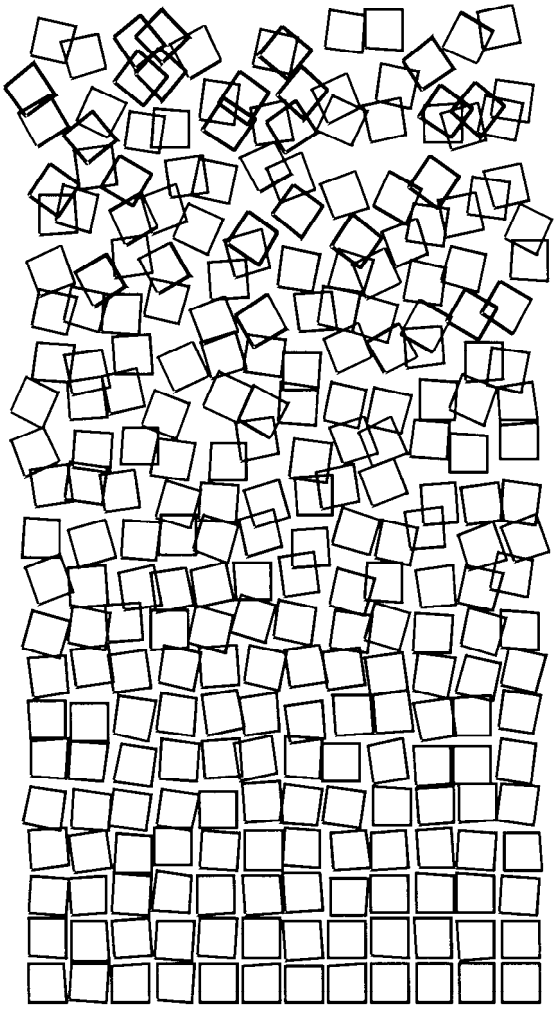
GESELLSCHAFT FÜR INFORMATIK E.V.

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://www.pars.gi-ev.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://www.pars.gi-ev.de/

vermittelt aktuelle Informationen über PARS.

CALL FOR PAPERS

9th Workshop on Parallel Systems and Algorithms
PASA 2008

in Conjunction with

21st International Conference on Architecture of Computing Systems (ARCS)
Dresden, Germany, February 25th to 28th, 2008

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

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*
- *models of parallel computation*
- *data stream-oriented computing*
- *parallel and distributed algorithms*
- *network and grid computing*
- *parallel organic computing*
- *interconnection networks*
- *reconfigurable parallel computing*
- *distributed and parallel multimedia systems*
- *performance evaluation of parallel systems*
- *software engineering for parallel systems*
- *parallel programming languages*
- *new technologies & architectures (SoC, Multicores, PIM, etc.)*
- *alternative technologies (e.g. quantum or DNA computing)*

The workshop will comprise invited papers on current topics of leading experts in the field as well as submitted papers. The accepted papers shall have a length of about 10 pages A4 and will be published in the ARCS Workshop Proceedings as well as in the PARS Newsletter (ISSN 0177-0454). For ARCS 2008 participants the workshop is included in the conference fee, for participation in the workshop alone, only a reduced fee is due. 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.

Important Dates

23rd Nov. 2007: Deadline for electronic submission of full papers of about 10 pages or extended abstracts of about 3 to 4 pages length (in English) under: http://www.ra.informatik.tu-darmstadt.de/pasa/2008/
14th Dec. 2007: Notification of the authors
28th Dec. 2007: Final version for workshop proceedings

Program Committee

H. Burkhart (Basel), A. Döring (Zürich), W. Erhard (Jena), R. Hoffmann (Darmstadt), F. Hoßfeld (Jülich), W. Karl (Karlsruhe), J. Keller (Hagen), Ch. Lengauer (Passau), E. Maehle (Lübeck), E. W. Mayr (München), F. Meyer auf der Heide (Paderborn), W. E. Nagel (Dresden), K. D. Reinartz (Höchstadt), U. Rüde (Erlangen), P. Sanders (Karlsruhe), Ch. Scheideler (München), H. Schmeck (Karlsruhe), U. Schwiegelshohn (Dortmund), P. Sobe (Lübeck), T. Ungerer (Augsburg), R. Wanka (Erlangen), H. Weberpals (Hamburg-Harburg), M. Zitterbart (Karlsruhe)

Organisation

Prof. Dr. Wolfgang E. Nagel, Technische Universität Dresden, Zentrum für Informationsdienste und Hochleistungsrechnen (ZIH), D-01062 Dresden, Tel. +49-351-463-35450, Fax +49-351-463-37773
E-Mail: wolfgang.nagel@tu-dresden.de

Prof. Dr.-Ing. Rolf Hoffmann, FG Rechnerarchitektur, FB Informatik, TU Darmstadt, Hochschulstr. 10, D-64289 Darmstadt, Tel. 06151-16-3611/3606, Fax 06151-165410
E-Mail: hoffmann@informatik.tu-darmstadt.de

9. PASA-Workshop

Seite

Towards PetaFlops Computing with IBM Blue Gene

Norbert Attig and Friedel Hoßfeld

Jülich Supercomputing Centre (JSC)
Forschungszentrum Jülich GmbH
52425 Jülich
n.attig@fz-juelich.de
f.hossfeld@fz-juelich.de

Abstract: Driven by technology, Scientific Computing is rapidly entering the PetaFlops era. The Jülich Supercomputing Centre is focusing on the IBM Blue Gene architecture to provide computer resources of this class to its users. Details of the system will be discussed and applications will be introduced which significantly benefit from this new architecture.

1 Introduction

In many areas, numerical simulations are the essential method for achieving innovative, high-quality results. Driven by the rapid development in computer technology, this insight has dramatically increased the requirements of computational scientists with respect to application performance, memory, data storage and data transfer capabilities [BHL05, Ko06]. Currently, only high-performance supercomputers with a large number of processors are capable to fulfil these needs.

The Jülich Supercomputing Centre, one of three national supercomputing centres, has implemented a dual supercomputer concept to provide computational scientists with adequate computing resources. On one side, a moderately parallel cluster serves about 150 user groups from academia and research institutions. This system allows the development of parallel codes as well as the execution of small to mid-size projects. For applications which scale up to thousands of processors, on the other side, IBM Blue Gene systems are available, serving as Leadership-class systems addressing the PetaFlops scale. Here, a much smaller number of projects are granted to give selected researchers the opportunity to get new insights into complex problems which were out of reach before.

2 IBM Blue Gene systems at JSC

When in 2004/2005 the IBM Blue Gene technology became available, the Jülich Supercomputing Centre recognised the potential of this architecture as a Leadership-class system for capability computing applications. In early summer 2005, Jülich started testing a single Blue Gene/L rack with 2,048 processors [AW05]. It soon became obvious that many more applications than initially expected were ported to efficiently run on the Blue Gene architecture. Therefore, in January 2006 the system was expanded to 8 racks with 16,384 processors, funded by the Helmholtz Association. The 8-rack system has successfully been in operation for almost two years now. Today, about 30 research groups, which were carefully selected with respect to their scientific quality, run their applications on the system using job sizes between 1,024 and 16,384 processors.

In early 2007, Research Centre Jülich decided to order a powerful next-generation Blue Gene system. In October 2007, a 16-rack Blue Gene/P system with 65,536 processors was installed [SW07], mainly financed by the Helmholtz Association and the State of North Rhine Westphalia. With its peak performance of 222.8 TFlop/s, Jülich's Blue Gene/P – alias JUGENE – is currently the biggest supercomputer in Europe and ranked No 2 worldwide.

The important differences between Blue-Gene/P and Blue Gene/L largely concern the processor and the networks while the principal build-up of Blue Gene/L was kept unchanged. Key features of Blue Gene/P are:

- 4 PowerPC® 450 processors are combined in a fully 4-way SMP (node) chip which allows a hybrid programming model with MPI and OpenMP (up to 4 threads per node).

- The network interface is fully DMA (Direct Memory Access) capable which increases the performance while reducing the processor load during message handling.

- The available memory per processor has been doubled.

- The external I/O network has been upgraded from 1 to 10 Gigabit Ethernet.

A key feature of this architecture is its scalability towards PetaFlops computing based on low power consumption, small footprint and an outstanding price performance ratio.

3 Running Applications on Blue Gene

Due to the fact that the Blue Gene systems are well balanced in terms of processor speed, memory latency, and network performance, many applications scale reasonably up to large numbers of processors. More surprising was that so many applications could be ported to and run efficiently on this new architecture which in a forerunner version was mainly designed to perform lattice quantum chromo dynamics (LQCD) codes. Blue Gene applications at JSC cover a broad spectrum ranging from LQCD to MD codes like CPMD and VASP, materials science, protein folding codes, fluid flow research, quantum computing and many, many others.

The performance and the scaling behaviour of the applications are continuously being improved in close collaboration between the user support team at JSC and the corresponding computational scientists. For example, a code from theoretical elementary particle physics runs now on Blue Gene/P at nearly 40 % of the peak performance compared to about 25 % on Blue Gene/L. In this context Blue Gene Scaling Workshops, where experts from Argonne National Laboratory, IBM and Jülich help to further optimise some important applications are highly welcome. Computational scientists from many research areas take the chance to improve their codes during these events and then later apply for significant shares of Blue Gene computer time to tackle unresolved questions which were out of reach before.

References

[AW05] Attig, N., Wolkersdorfer, K.: IBM Blue Gene/L in Jülich: A First Step to Petascale Computing. In inSiDE, 3, 2005, 2, S. 18-19

[BHL05] Bode, A., Hillebrandt, W., Lippert, T: Petaflop-Computing mit Standort Deutschland im europäischen Forschungsraum, Bedarf und Perspektiven aus Sicht der computergestützten Natur- und Ingenieurwissenschaft. "Scientific Case" im Auftrag des BMBF, Bonn, 2005

[Ko06] Koski, K. et al.: "European Scientific Case for high-end computing". HPC in Europe Task Force, http://www.hpcineuropetaskforce.eu/draftdeliverables

[SW07] Stephan, M., Wolkersdorfer, K.: IBM BlueGene/P in Jülich: The Next Step towards Petascale Computing. In inSiDE, 5, 2007, 2, S. 46-47

Specifying and Processing Co-Reservations in the Grid

Thomas Röblitz
Zuse Institute Berlin, Takustr. 7, D-14195 Berlin, Germany
roeblitz@zib.de

Abstract: Executing complex applications on Grid infrastructures necessitates the guaranteed allocation of multiple resources. Such guarantees are often implemented by means of advance reservations. Reserving resources in advance requires multiple steps – beginning with their description to their actual allocation. In a Grid, a client possesses little knowledge about the future status of resources. Thus, manually specifying successful parameters of a co-reservation is a tedious task. Instead, we propose to parametrize certain reservation characteristics (e.g., the start time) and to let a client define criteria for selecting appropriate values. Then, a Grid reservation service processes such requests by determining the future status of resources and calculating a co-reservation candidate which satisfies the criteria. In this paper, we present the *Simple Reservation Language (SRL)* for describing the requests, demonstrate the transformation of an example request into an integer program using the *Zuse Institute Mathematical Programming Language (ZIMPL)* and experimentally evaluate the time needed to find the optimal co-reservation using *CPLEX*.

1 Introduction

In many scientific disciplines, large scale simulations and workflows for analyzing petascale data sets necessitate the adoption of Grid technologies to meet their demanding resource requirements. The use of Grid resources, however, poses new challenges, because of their heterogeneity, geographic distribution and autonomous management. Especially, the efficient execution of complex applications, e.g., large simulations of black holes with Cactus or distributed observations of astronomic objects with robotic telescope as envisioned in the AstroGrid-D project [GAC07], requires the *co-allocation* of multiple Grid resources. Because Grid resources are autonomously managed, the allocation of them at the same time or in some sequence cannot be *guaranteed* by standard Grid job management schemes, i.e., a broker decides where to submit a job, but has no control on when the job is actually executed.

This problem can be circumvented by acquiring *reservations* for the required resources. A reservation guarantees that a specific resource can be allocated to a request at the desired time. Typically, reservations have a fixed duration and are acquired some time in advance. Considering the vast number of resources eligible for reservation, a user may pose constraints on which resources are selected. In addition, the provider may want to optimize the utilization of their resources. Hence, a broker has to solve a complex optimization problem to find a mapping of application components to resources at specific times.

In this paper, we model the mapping problem as an integer programming problem and use a standard solver to find its best solution. We use the *Zuse Institute Mathematical Programming Language* (ZIMPL) [Koc04] to describe the integer programming problem and the solver *CPLEX* for finding the solution. To simplify the specification of co-reservation requests, we

propose a simple language for defining the requirements of each part along with the objective to be optimized. Throughout the paper, we reuse an example presented in our previous work [RR05]. The example describes a co-reservation request including temporal and spatial relationships between atomic request parts.

> *Reserve 16 CPUs of an IBM p690, 32 CPUs of a PC cluster and a one Gbit/s-network connection between them, each for six hours between 2007/12/12 06:00pm and 2007/12/15 06:00pm. All reservations must start at the same time. Reserve a visualization pipeline for two hours starting four hours after the reservation on the IBM p690 begins and a 100 Mbit/s-network connection between the p690 and the visualization pipeline for the same time.*

Outline. We summarize related work in Section 2. The general procedure for processing co-reservation requests is presented in Section 3. Section 4 describes the *Simple Reservation Language (SRL)* for specifying requests in detail. Thereafter, we demonstrate – using the scenario described above – the transformation of SRL requests into an optimization problem in Section 5. Then, we present an experimental evaluation of the times to find an optimal co-reservation candidate in Section 6 and conclude in Section 7.

2 Related Work

In [FKL+99], Foster et al. present GARA, a resource management architecture for reserving multiple resources in advance. The focus of their work is on the general scheme to guarantee that an application can co-allocate multiple resources at specific times. They also discuss needed extensions of existing local resource management such that they support advance reservations. Our work focuses on the description and processing of co-reservation requests. The needed components for processing requests are similar to those in GARA. Hence, our mechanism could be easily integrated in their framework.

Condor ClassAds, proposed by Raman et al. [RLS98], provides a mechanism for matching atomic requests to resources. Condor ClassAds is a very versatile language to specify both a user's request and the offers of the resource providers. The language we propose for describing reservation requests is conceptually similar to ClassAds, because it is also based on $(attribute, value)$-pairs and allows references between different matching parties. The main reason for not using pure ClassAds was that the processing of them differs from the processing of multi-resource reservation requests. Liu and Foster extend Condor ClassAds by applying constraint programming techniques to the problem of selecting resources for execution of immediate requests in a Grid environment [LF03].

In [NLYW05], Naik et al. present an integer programming based approach for assigning requests to heterogeneous resources. In contrast to our work, they consider immediate requests only. That is the resource manager tries to match as many requests to resources at the current time. Scheduling requests at future times is not considered. The VIOLA metascheduler [WWZ05] schedules parallel compute jobs to multiple resources by incrementally increasing the advance booking period until all jobs may be allocated. In contrast to our work, it only supports one criteria – earliest completion time.

3 The Reservation Framework

The framework consists of the three main components: the *Grid Reservation Service (GRS)*, the *Local Reservation Service (LRS)*, and the *Resource Information Service (RIS)*. The general procedure for processing co-reservation requests is as follows.

① A user sends a request described by the *Simple Reservation Language* to the GRS.

② The GRS queries the RIS to determine resources which support advance reservation and match the core characteristics such as type and operating system.

③ The GRS sends *probe* requests to the LRSes of these resources to let them provide detailed information about their future status and reservation fees.

④ The GRS transforms the user request including the probed information into the *ZIMPL* format and thereby instantiates the optimization problem.

⑤ The GRS determines the best co-reservation candidate. A co-reservation candidate maps each request part to a $(resource, starttime)$-pair.

⑥ The GRS tries to acquire the reservations given by the best co-reservation candidate. Thus, for each $(resource, starttime)$-pair the GRS sends a *reserve* request to the LRS of the resource *resource*.

⑦ If some *reserve* requests are denied, the GRS refines the optimization problem and proceeds with step ⑤.

⑧ If all *reserve* requests are admitted or no solution could be found eventually, the GRS sends a response message to the client.

Note, in this paper we focus on the steps ①, ④ and ⑤. Step ② is implemented in todays Grid resource brokers (cf. Section 2). Methods for determining the future status of resources are described in our previous work [RR06, RSR06]. The actual acquiring of the reservations (step ⑥) and the refinement in case of failures (step ⑦) are subject to future work.

4 The Simple Reservation Language

The purpose of the *Simple Reservation Language (SRL)* is twofold. First, it enables clients to easily describe reservation requests without requiring them to know the details of a mathematical programming language. Second, the Grid Reservation Service (GRS) can efficiently pre-process reservation requests, because the SRL uses a small set of attributes and limited value domains only. Despite these restrictions, the language is powerful enough to describe a large variety of requests as will be seen in the following subsections. As outlined in the introduction, we regard the co-reservation problem as an optimization problem. Formally, a co-reservation request must define a set of variables including their domains, a set of constraints on these variables' domains and an objective function which is to be optimized. The SRL follows a less formal approach by letting a user define certain attributes, i.e., the SRL vocabulary, which are transformed into the corresponding mathematical terms.

Table 1: Attribute scopes of the Simple Reservation Language.

Used in	Scope	Description
key & value	TS	Temporal specification of a request
key & value	QOS	QoS specification of a request
key & value	MISC	Miscellaneous attributes of a request
key	CON	Constraints of a request
key	OBJ	Objectives of a request
value	RVC	Attributes of a reservation candidate

4.1 Structure of a Co-Reservation Request

A co-reservation request consists of multiple atomic requests as well as constraints and objectives defining the relationships between any two atomic requests. Each SRL request is a set of $(attribute, value)$-pairs. The domain of a value depends on the attribute and will be discussed in the following subsections. The syntax of an attribute is defined as

```
<attribute> := <id>'.'<scope>'.'<key>.
```

The *key* of an attribute is an alpha-numeric string. For each scope there exist several keys with a pre-defined meaning which we will present along with the discussion of the scopes. In addition, any other string may be used, but its meaning is only defined by the request itself. The *scope* denotes a specific group of attributes. We distinguish three kinds of scopes, based on its appearance in an $(attribute, value)$-pair (cf. Table 1). The component `<id>` associates attributes with a specific part of a co-reservation request. Thus, it is possible to reference attributes of a specific part from within any other part.

4.2 Description of Atomic Requests

The vocabulary of the SRL must cope with the following issues to describe a (meaningful) atomic request:

- When should the resource be reserved?
- Type and quantity of the resource to be reserved.
- Which constraints must be met?
- Which criteria should be optimized?

Temporal specification. The scope `TS` defines three attributes `est` (earliest start time), `let` (latest end time) and `duration` for specifying, when the resource should be reserved. The values can be given as epoch seconds or UTC[1] string.

[1] UTC stands for Coordinated Universal Time.

Quality-of-Service Specification. The scope `QOS` defines attributes for specifying what type of resource and what quantity or quality of this resource should be reserved. These attributes are `QOS.type`, `QOS.cpus`, `QOS.netbw`, `QOS.diskspace`. The GRS does not handle the differences among the corresponding resource types internally. It must just ensure to contact the right local reservation service for each type of resource. The values for `QOS.type` are `compute`, `network`, `storage` and `visualize`. The values for the corresponding quantity attributes are within the usual domains.

Miscellaneous Attributes. The scope `MISC` may contain any attribute which does not fit into the previous categories. For example, it may be used to associate a user id or certificate with a request which could be necessary for authentication and authorization.

Constraint Specification. The scope `CON` is used to define constraints on any attributes of all scopes except `CON` and `OBJ`. The key of a `CON`-attribute can be any alpha-numeric string. Two different attributes must not use the same key. The syntax of the value of a `CON`-attribute is defined as follows.

```
<con value> := <expr><op><expr>
<expr>      := <attribute>|<quantity>
<op>        := '<'|'<='|'=='|'!='|'>='|'>'
```

The term `<attribute>` refers to any attribute (except for the scopes `CON` and `OBJ`). For example, the cost `IBM.RVC.cost` for reserving the requested resources could be limited. The term `<quantity>` specifies a quantity such as 100 Mbit/s.

Objectives Specification. The scope `OBJ` defines the objective of the request. The objective is the weighted sum of all sub-objectives. A client must assign a weight to each sub-objective such that their sum equals one. Because the range of values of the sub-objective attributes may be very different and not known a-priori, each attribute must be normalized. The syntax of the actual sub-objective specification is as follows.

```
<obj> := {'min'|'max'},'<attribute>','<weight>
```

The term `<attribute>` refers to any attribute (except for the scopes `CON` and `OBJ`). For example, to execute the application on the IBM p690 at the earliest moment, the sub-objective is `min,IBM.RVC.begin,0.6`.

Reservation Candidates. The scope `RVC` contains attributes of a **R**eser**V**ation **C**andidate, which are determined in the probing phase (cf. step ③).

4.3 Description of Co-Reservation Requests

A *co-reservation request* consists of multiple atomic requests, additional constraints and objectives defining the relationships among the atomic parts. Relationships among different parts are enabled by the identifier component `<id>` of an attribute's key. Thus, attributes of the

part <id> can be referenced in the constraints and objectives of any other part. For example, the *same time*-constraint in the introduction's scenario can be specified as `IBM.RVC.begin == PCC.RVC.begin`. This scheme is very flexible. Also, workflow-like applications may be modeled, i.e., that one part should start only after another has finished. In that case the constraint would be `VIS.RVC.begin >= IBM.RVC.end`. Additionally, objectives may reference attributes in different parts. For example, the above workflow application may desire that its whole execution time is minimized. Such goal can be specified by the sub-objective `min,VIS.RVC.end-PCC.RVC.begin,10`. In Fig. 1, the main parts of the scenario of the introduction are described using the Simple Reservation Language.

```
IBM.TS.est   = Dec 12 18:00:00 2007
IBM.TS.duration = 21600
IBM.TS.let   = Dec 15 18:00:00 2007
IBM.QOS.type = compute
IBM.QOS.cpus = 16
IBM.QOS.arch = "IBM p690"
IBM.CON.cost = IBM.RVC.cost <= 10000
IBM.OBJ.cost = min,IBM.RVC.cost,8
IBM.OBJ.start = min,IBM.RVC.begin,10
PCC.TS.duration = 21600
PCC.QOS.type = compute
PCC.QOS.cpus = 32
PCC.QOS.arch = "PC cluster"
PCC.CON.time = PCC.RVC.begin
               == IBM.RVC.begin
VIS.TS.duration = 7200
VIS.TS.let   = Dec 15 18:00:00 2007
VIS.QOS.type = visualize
VIS.QOS.cpus = 4
VIS.QOS.arch = "SGI Onyx"
VIS.CON.time = PCC.RVC.begin+14400
               == VIS.RVC.begin
```

Figure 1: A co-reservation request described in SRL (in extracts).

5 Transforming SRL Requests into ZIMPL

In the following, we exemplify how SRL requests are transformed into the *Zuse Institute Mathematical Programming Language* (ZIMPL) [Koc04]. The result can then be used by standard integer programming solvers to find the best reservation candidate. Listing 1 shows the complete implementation of the scenario described in the introduction. It consists of seven parts, which we will now explain in detail.

Request Structure. The number of atomic request parts is read from a configuration file in line 2. The example co-reservation request contains 5 parts (one IBM p690, one PC cluster, one network link IBM–PC, one SGI visualization resource and one network link SGI–IBM).

Reservation Candidates. The set of reservation candidates is initialized in line 5. This set is transformed into a multi-dimensional array (lines 6–8) to make individual metrics of a candidate accessible. Note, by separating the reservation candidates for each part (first index in array rvc), we ensure that a request is matched to a resource with the required type.

Model Variables. Each atomic request $r \in R$ must be assigned to a resource $s \in S$ at a certain start time $t \in T$. Thus, we model the problem with $R \times S \times T$ *binary* variables (line 11). The variable xb[r, s, t] is set to 1 iff the atomic request r is assigned to resource s at start time t.

Matching and Cost Constraints. We constrain the number of binary variables set to 1 such that each request part is only once assigned to a resource (line 14). The total cost for all parts of the co-reservation are limited in line 15.

Listing 1: Implementation of the example scenario as *integer program* in ZIMPL.

```
1  # request parts
2  set P := { read "req.dat" as "<1n>" comment "#" };
3
4  # reservation candidates
5  set PRTMV  := { read "rvc.dat" as "<1n,2n,3n,4n,5n>" comment "#"};
6  set RVC    := proj(PRTMV, <1,2,3>);
7  set METRIC := proj(PRTMV, <4>);
8  param rvc[RVC*METRIC] := read "rvc.dat" as "<1n,2n,3n,4n> 5n" comment "#" default 100;
9
10 # model variables
11 var xb[RVC] binary;
12
13 # matching, type and cost constraints
14 subto once: forall <p> in P: (sum <p,i,j> in RVC: xb[p,i,j]) == 1;
15 subto cost: (sum <p,i,j> in RVC: xb[p,i,j]*rvc[p,i,j,1]) <= 10000;
16
17 # temporal relationships
18 subto temp1: sum <1,i,j> in RVC: xb[1,i,j]*j == sum <2,m,n> in RVC: xb[2,m,n]*n;
19
20 subto temp2: sum <1,i,j> in RVC: xb[1,i,j]*j == sum <3,m,n> in RVC: xb[3,m,n]*n;
21
22 subto temp3: sum <1,i,j> in RVC: xb[1,i,j]*(j+14400) == sum <4,m,n> in RVC: xb[4,m,n]*n;
23
24 subto temp4: sum <4,i,j> in RVC: xb[4,i,j]*j == sum <5,m,n> in RVC: xb[5,m,n]*n;
25
26 # spatial relationships
27 set NET  := proj({ read "net.dat" as "<1n,2n>" comment "#" }, <1>);
28 set ENDS := { <1>, <2> };
29 param net[NET*ENDS] := read "net.dat" as "<1n,2n> 3n" comment "#";
30
31 subto spat1: sum <1,i,j> in RVC: xb[1,i,j]*i ==
32              sum <m> in NET: sum <3,m,n> in RVC: xb[3,m,n]*net[m,1];
33
34 subto spat2: sum <2,i,j> in RVC: xb[2,i,j]*i ==
35              sum <m> in NET: sum <3,m,n> in RVC: xb[3,m,n]*net[m,2];
36
37 subto spat3: sum <1,i,j> in RVC: xb[1,i,j]*i ==
38              sum <m> in NET: sum <5,m,n> in RVC: xb[5,m,n]*net[m,1];
39
40 subto spat4: sum <4,i,j> in RVC: xb[4,i,j]*i ==
41              sum <m> in NET: sum <5,m,n> in RVC: xb[5,m,n]*net[m,2];
42
43 # objective function
44 set OBJS := { <1>, <2>, <3> }; # 1-cost, 2-time & 3-bandwidth
45 param CF[P*OBJS] := read "coeff.dat" as "<1n,2n> 3n" comment "#";
46
47 minimize objective:  sum <p> in P:
48   ( sum <p,i,j> in RVC: xb[p,i,j]*rvc[p,i,j,1]*CF[p,1]
49   + sum <p,i,j> in RVC: xb[p,i,j]*rvc[p,i,j,2]*CF[p,2]
50   - sum <p,i,j> in RVC: xb[p,i,j]*rvc[p,i,j,3]*CF[p,3] );
```

Temporal Relationships. We require that the start times of part one (IBM p690) and part two (PC cluster) must be equal (line 18). Also, the start times of part one and part three (network IBM–PC) must be equal (line 20). The next constraint (line 22) requires that the SGI visualization part begins exactly four hours after the computational parts begin (e.g., the IBM part). The last temporal constraint (line 24) ensures that the network between the IBM and the SGI is reserved from the same start time as the SGI part. Note, due to the fixed durations of all parts we do not need to put constraints on the end times of the reservations.

Table 2: Parameters of the experimental evaluation.

Parameter	Values
no. of resources	`1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 12, 14, 16, 18, 20`
no. of candidates	`7, 12, 23, 34, 67, 133, 265, 397`
corresponding time gap	`11h, 6h, 3h, 2h, 1h, 30m, 15m, 10m`

Spatial Relationships. We assume a fully connected network with bi-directional links. Thus, n resources (compute and visualization) are connected by n^2 links. The network configuration is initialized from line 27 to line 29. The actual spatial relationships are defined for the network link between the IBM p690 part and the PC cluster (lines 31/32 and 34/35) and for the network link between the IBM p690 part and the SGI visualization pipeline (lines 37/38 and 40/41). For the sake of simplicity, we use the number of a resource as its location identifier (cf. multiplication with i on the left side of each comparison).

Objective Function. We use the weighted sum as global objective function (lines 47–50). It aggregates three sub-objectives – minimum cost, minimum start time and maximum available bandwidth. Because the value ranges of the metrics differ significantly (cost: 0-20000, start time: 0-237600, bandwidth: 0-1000), we scale them by appropriate factors (the maximum of each value range). These factors are the coefficients initialized in line 45.

6 Experimental Evaluation of the Scalability

Many optimization problems, in particular integer problems, suffer from a large search space. We studied the impact of the number of eligible resources and the number of reservation candidates on the time needed to find the optimal co-reservation candidate. In the absence of workload traces for co-reservations we randomly generated the reservation candidates (step ③). For each parameter pair (no. of resources, no. of candidates), we generated 10 experiments and calculated the average time for finding the optimal co-reservation candidate. Table 2 lists the parameters of all experiments, which were sequentially executed on a SUN Galaxy 4600 16 core system with 64 Gbytes of RAM. Each experiment used a single processor core only.

Fig. 2 shows the solving time against the number of reservation candidates. Each curve represents the experiments with a specific number of resources. Additionally, the graph shows two approximations of the solving time. For the experiments with one resource, the solving time increases exponentially with the number of candidates. For the experiments with 20 resources, the solving time increases quadratically with the number of candidates.

Whether the solving time is acceptable in real world scenarios depends on several parameters. First, a client may want a response as soon as possible. Second, the calculated future status (cf. step ③) may only be valid for a certain time. Thereafter, the "best" co-reservation candidate is sub-optimal or reservation attempts (cf. step ⑥) simply fail. Third, the longer the book-ahead time (earliest start time) of the co-reservation is, the longer solving times may be acceptable.

The experimental results provide two means for limiting the solving time – (1) the GRS may ask the resource providers for a limited number of reservation candidates and (2) use less eligible resource than found through the resource information service query (cf. step ②).

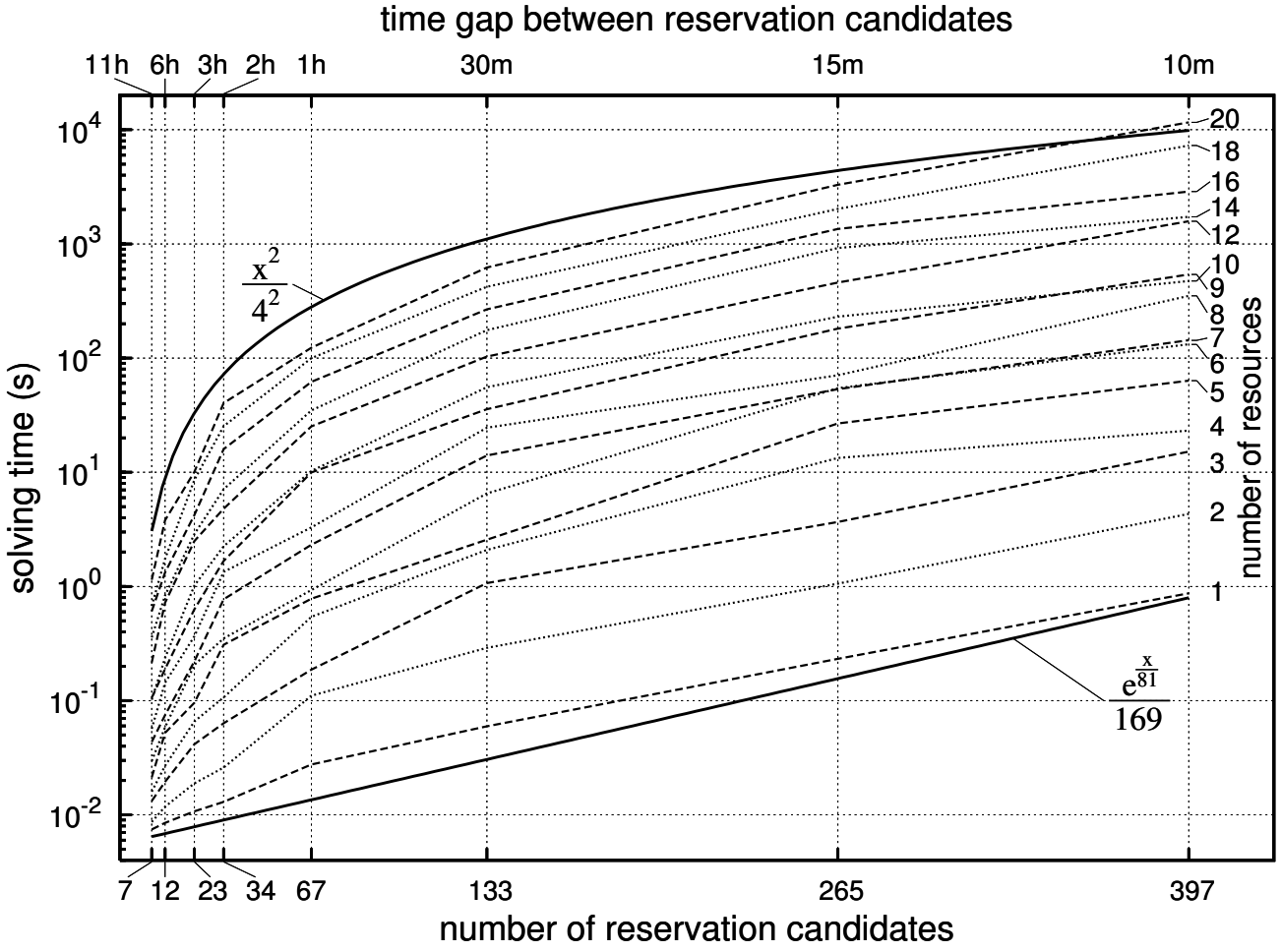


Figure 2: Solving time for several numbers of resources against the number of reservation candidates.

7 Conclusion

Resource allocation for complex applications requires guarantees to ensure desired quality-of-service levels. Such guarantees are typically implemented by reserving the requested resources in advance. We demonstrated an approach for specifying and processing co-reservation requests. The requests are specified in a simple yet powerful language and transformed into an integer program. Our approach is motivated by the observation, that finding the best co-reservation is an optimization problem. The use of a mathematical programming language makes extensions and refinements easy to implement. Moreover, there already exist well-known tools for solving optimization problems described in such languages. Of course, that flexibility comes at some cost namely the time needed to find a solution. The performance experiments revealed means to limit the solving time, i.e., by limiting the number of reservation candidates and/or by using less eligible resources. Nevertheless, the flexibility of the approach facilitates many extensions such as specifying data dependencies and optimizing data transfers. Also, moldable requests may be supported to optimize the utilization of the resources.

Acknowledgment

This work was supported by the German Federal Ministry of Education and Research within the D-Grid initiative under contract 01AK804C and the European Network of Excellence CoreGRID, Institute on Resource Management and Scheduling, under contract IST-2002-004265.

References

[FKL+99] I. Foster, C. Kesselman, C. Lee, R. Lindell, K. Nahrstedt, and A. Roy. A Distributed Resource Management Architecture that Supports Advance Reservations and Co-Allocation. In *Proceedings of the International Workshop on Quality of Service*, pages 27–36. IEEE Press: Piscataway, NJ, 1999.

[GAC07] AstroGrid-D project homepage. http://www.gac-grid.org/, November 2007.

[Koc04] Thorsten Koch. *Rapid Mathematical Programming*. PhD thesis, Technische Universität Berlin, 2004. ZIB-Report 04-58.

[LF03] Chuang Liu and Ian Foster. A Constraint Language Approach to Grid Resource Selection. Technical Report TR-2003-07, Department of Computer Science, University of Chicago, March 2003.

[NLYW05] Vijay K. Naik, Chuang Liu, Lingyun Yang, and Jonathan Wagner. On-line Resource Matching in a Heterogeneous Grid Environment. In *Proceedings of the IEEE International Symposium on Cluster computing and Grid 2005 (CCGrid05), Cardiff, Wales, UK*, volume 2, pages 607–614, May 2005.

[RLS98] Rajesh Raman, Miron Livny, and Marvin Solomon. Matchmaking: Distributed Resource Management for High Throughput Computing. In *Proceedings of the 7th IEEE International Symposium on High Performance Distributed Computing, Chicago, Illinois, USA*, pages 140–146. IEEE Computer Society Press, July 1998.

[RR05] Thomas Röblitz and Alexander Reinefeld. Co-Reservation with the Concept of Virtual Resources. In *Proceedings of the IEEE International Symposium on Cluster computing and Grid 2005 (CCGrid05), Cardiff, Wales, UK*, volume 1, pages 398–406, May 2005.

[RR06] Thomas Röblitz and Krzysztof Rzadca. On the Placement of Reservations into Job Schedules. In *Proceedings of the 12th International Euro-Par Conference 2006, Dresden, Germany*, pages 198–210, 2006.

[RSR06] Thomas Röblitz, Florian Schintke, and Alexander Reinefeld. Resource Reservations with Fuzzy Requests. *Concurrency and Computation: Practice and Experience*, 18(13):1681–1703, November 2006.

[WWZ05] Oliver Wäldrich, Philipp Wieder, and Wolfgang Ziegler. A Meta-scheduling Service for Co-allocating Arbitrary Types of Resources. In *Proceedings of the 6th International Conference on Parallel Processing (PPAM 2005), Poznan, Poland*, volume 1, pages 782–791, September 2005.

Grid Virtualization Engine: Providing Virtual Resources for Grid Infrastructure

Emeric Kwemou[1], Lizhe Wang[2], Jie Tao[2], Marcel Kunze[2],
David Kramer[1], Wolfgang Karl[1]

[1]Institut für Technische Informatik, Universität Karlsruhe (TH),
76128 Karlsruhe, Germany
Emeric.Kwemou@web.de, {karl, Kramer}@ira.uka.de
[2]Institut für Wissenschaftliches Rechnen, Forschungszentrum Karlsruhe,
76344 Eggenstein-Leopholdshafen, Germany
{Lizhe.Wang, Jie.Tao, Marcel.Kunze}@iwr.fzk.de

Abstract. Virtual machines offer a lot of advantage such as easy configuration and management and can simplify the development and the deployment of Grid infrastructures. Various virtualization implementations despite have similar functions often provide different management and access interfaces. The heterogeneous virtualization technologies bring challenges for employing virtual machine as computing resources to build Grid infrastructures. The work proposed in this paper focus on a Web service based virtual machine provider for Grid infrastructures. The Grid Virtualization Engine (GVE) creates an abstraction layer between users and underlying virtualization technologies. The GVE implements a scalable distributed architecture, where an GVE Agent represents a computing center. The Agent talks with different virtualization product inside the computing center and provides virtual machine resources to GVE Site Service. Users could require computing resources through GVE Site Services. The system is designed and implemented in the state of the arts of distributed computing: Web service and Grid standards.

Keywords: Virtual machine, Grid Virtualization Engine, Grid Computing, Web Service

1 Introduction

Grid computing architecture [3] allows secure and scalable resource sharing in a large distributed environment. In most of the cases, shared resources are computing and storage resources, as well as software artifacts and scientific experiments [4] [5]. One of the challenges of a computational Grid is the configuration of computing resources for new jobs. Different software configurations might be required; new software needs to be installed before computations; the reconfiguration may occur every time when a new job needs the computing resource. In this paper, a software layer, Grid Virtualization Engine (GVE), is developed to hide the complexity of configuration in a Grid infrastructure and provide a customized computing environment for Grid users.

1.1 Virtual Machine

A Virtual Machine (VM) can support individual processes or a complete system depending on the abstraction level where virtualization occurs. It executes software in the same manner as the machine where they are developed [7]. Since several years the Grid computing research community shows interest for virtual machines and virtual environments. The typical Virtual Machine Monitor (VMM) or hypervisor setup includes Xen VMM [8], VMware server/ESX server [9], and User Mode Linux [10]. In general, users can benefit from the virtualization techniques in the following aspects:

(1) On demand creation and customization
Users can create a customized virtual machine, which can provide customized resource allocation for users, e.g., OS, memory, storage. Dynamic cloning instantiation and configuration of virtual machines offer the capability to easily and quickly reproduce virtual machines with a preconfigured profile.

(2) Performance isolation
Virtual machines can guarantee the performance for users and applications. Users of virtual machine could expect a dedicated computing environment, which is hard to find in multiple-user computing servers.

(3) Easy management
On the contrary of traditional computing resources, *superuser* privileges are granted for virtual machines users, allowing more flexibility. Another interesting feature of virtual machine is that VM can be easily suspended and migrated to other hosts.

1.2 Grid Virtualization Engine

The virtual machine based Grid system includes heterogeneous hosting resources, virtual machine technologies as well as programming interfaces. To ease the development of such infrastructures, it is important to:

(1) represent virtual machine profile in a standardized format,
(2) simplify the access to virtual machines provided by computer centers with standard and uniformed access interfaces, and
(3) build an infrastructure that can provide virtual machines from many computer centers.

We design and implement the Grid Virtualization Engine to fulfill the foregoing objectives. GVE implements a Web service based abstraction layer and can manage a set of virtual machines provided by heterogeneous VMMs located in distributed computing centers. Information provided for a virtual machine uses the standardized GLUE Schema [18]. The rest of the paper is organized as follows: related work is investigated in Section 2; Section 3 describes the architecture of the Grid Virtualization Engine; In Section 4, GVE use cases are discussed; this is followed by a description of implementation in Section 5; the paper concludes in Section 6 with a brief summary and future work.

2 Related Work

The VM-Shop virtual machine management system [13], used for example in the In-Vigo System [14], is an abstraction level for virtual machine monitors. The architecture is similar with our work: VM-Shop is correspondent to the GVE Site Service while VMPlants could be compared with the GVE Agents. The instantiation process is based on cloning like in the GVE. However, a graph based model is used to represent VMs and it makes the interoperability with existing Grid systems more difficult. We claim that our GVE implementation is more scalable and flexible.

SODA [16] project from University of Illinois is a Service-On-Demand architecture for application service hosting utility platforms. The difference between the GVE approach and the SODA architecture is that SODA provides long lived virtual machines that are preconfigured and used to handle services, rather than providing flexibly reconfigurable virtual machines for computational tasks.

Cluster on Demand (COD) project [15] of Duke University is an automated resource management framework that enables rapid, on-the-fly partitioning of a physical clusters into multiple independent virtual clusters. The provided virtual clusters are configured partitions of a physical cluster. So users will not have the benefits provided by a real virtual machines based virtual cluster.

GVE reaches the interoperability by using Web service to provide virtual machines and describing VMs with GLUE Schema. The GLUE (Grid Laboratory Uniform Environment) schema [18] represents an abstract model for Grid resources and mappings to concrete schemas that can be used by information services within Grids. The GLUE schema is widely used in production Grids. The underlying idea of the schema therefore is to provide an information model that can be used to exchange pieces of information among different knowledge domains and virtual organizations.

3 The Architecture of Grid Virtualization Engine

The Grid Virtualization Engine is a software layer between various virtualization implementations, computing centers and Grid users. Users can require and employ virtual machines via access interface of GVE. GVE talks with underlying computing centers and VMMs for virtual machine operations. Computer centers could also provide virtual machines to form Grid infrastructure. The used virtualization technology may differ from one computing center to another (i.e VMWare and XEN technologies). The GVE thus provides a standard and uniform access to virtual computing resources.

3.1 Virtual Machine Based Grid Infrastructure

The example scenario of virtual machine based Grid computing system is shown in Fig. 1. The system contains multiple computing centers (computing sites). Each center includes several hosting resources, where VMMs are installed, provide virtual machine resources.

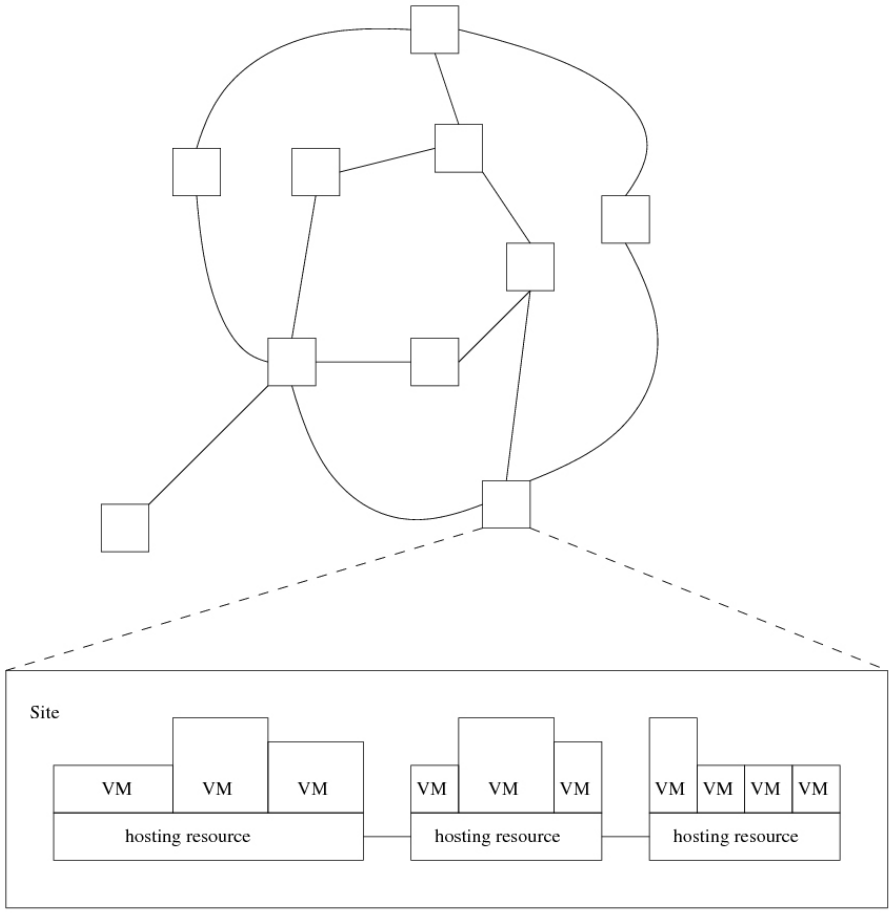


Fig. 1. Virtual machine based Grid infrastructure

3.2 GVE Architecture

The GVE is a hierarchical system that includes several components (Fig. 2) correspondent to the VM based Grid infrastructure (Fig. 1.):

GVE Site Service

The GVE Site Service resides on the access point of the computing center. It manages hosting resources inside the center via communicate with GVE agents that run on the hosting resources. Users who want to build Grid computing system with distributed virtual machines can access multiple GVE Site Services for virtual machine manipulations.

GVE Agent

The GVE Agent is a Web service which runs on hosting resources. It receives operation commands from GVE Site Service and talks with the specific VMM that is installed on the hosting resource. Thus GVE agent is virtualization technology dependent. In other word, for each type of VMM we need to prepare a correspondent GVE Agent.

User Information Database

The User Information Database records management policy for user access to the virtual machines inside the computing center.

Virtual Machine Disk Database

Virtual Machine Disk Database stores typical virtual disk images and software packages, which can facilitate the creation of virtual machines and customization of user execution environment.

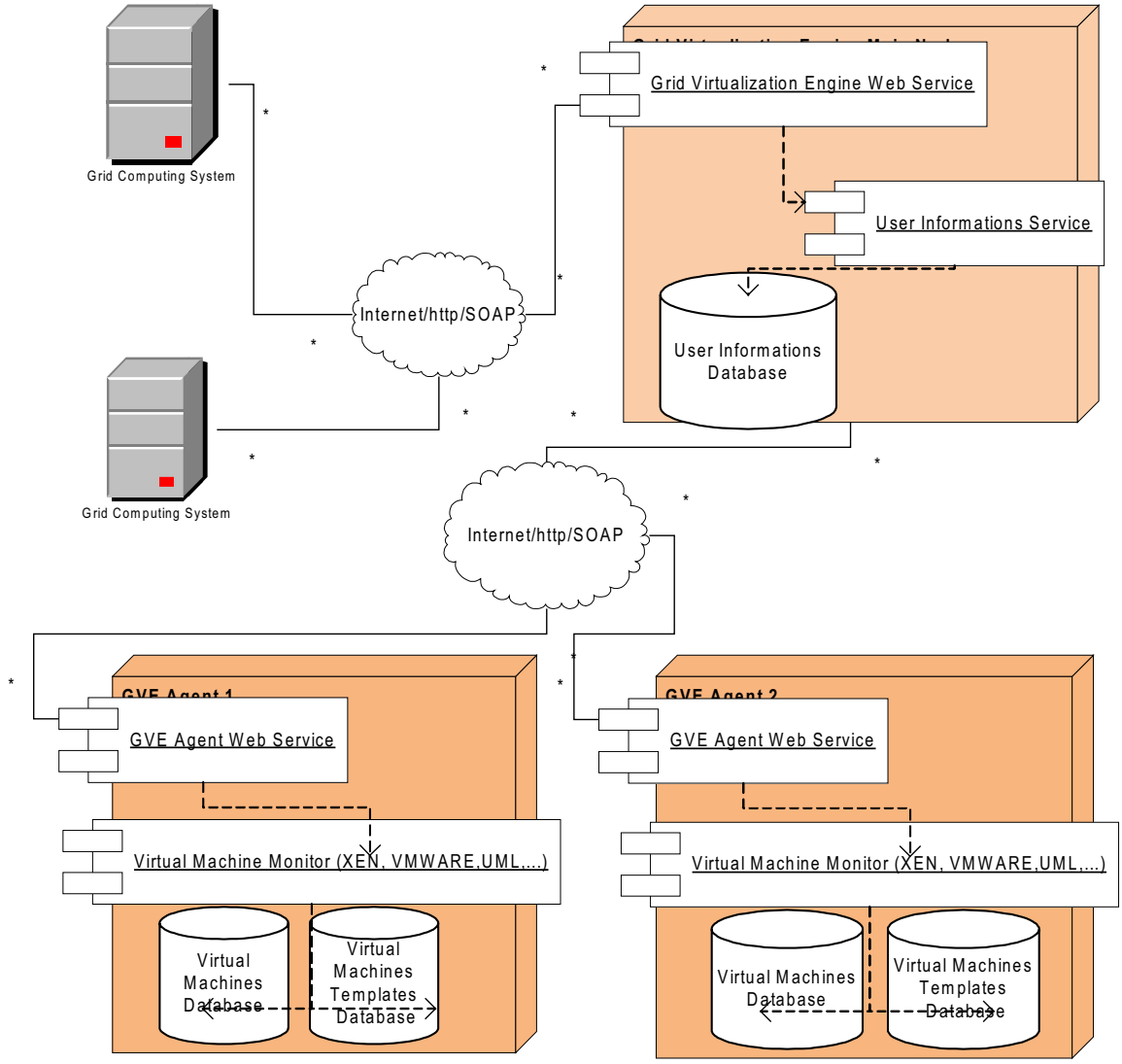


Fig. 2. Distribution diagram of the GVE with Agents plugged in.

The Distributed paradigm used to design the Grid Virtualization Engine makes the System scalable and extensible. Agents can be plugged in at runtime and can serve multiple GVE Site Services simultaneously. GVE Site Service runs as a client of the GVE Agent.

Another important key feature of the architecture is the use of Web service [1] to provide the functionalities needed. Web service is an established standard supported by many technology vendors. The use of Web service simplifies the integration of the GVE in distributed infrastructures.

4 Use Cases of Grid Virtualization Engine

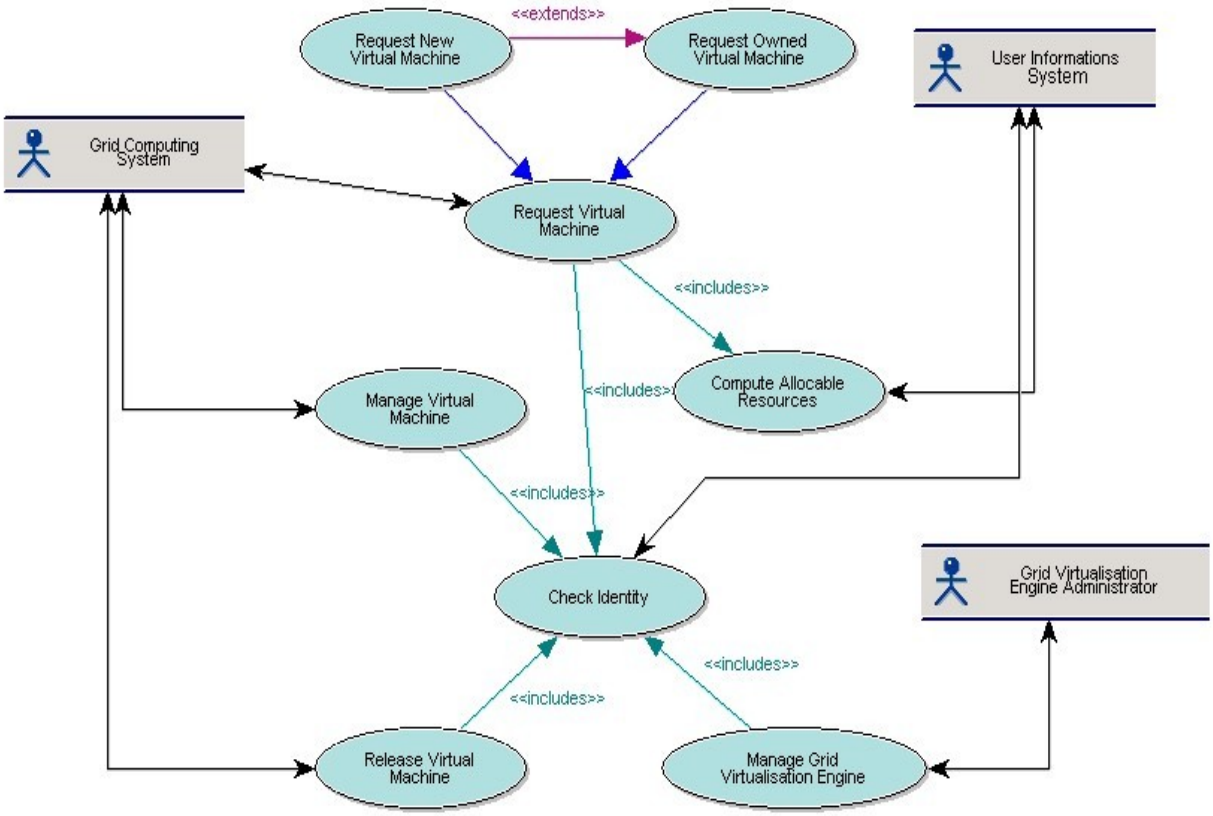


Fig. 3. Use Case Diagram of the Grid Virtualization Engine.

The detailed use cases of GVE are shown in Fig. 3. We have, for example:

the actors:

- The Grid Computing System (GCS) or the user who want to build Grid infrastructures is the main Actor in this use case. It is the one which requests virtual machine at runtime.
- The User Information Database is the actor which performs identity checks and resource allocation control.
- The administrator is another actor of the GVE use case. The administrator may changes GVE internal settings.

the Use Cases

- Request VM and Request VM are two generalizations of the Request Virtual Machine use case.
- Manage VM is triggered if the GCS needs to perform some management tasks on a VM.
- Release VM is triggered if a computational task on a VM is terminated or if the time limit allocated for the use of a VM is elapsed.
- Manage GVE is triggered by an administrator who wants to change security settings or plug in new GVE Components.
- Compute Allocable Resources is triggered when GVE wants to control if a new VM can be allocated to the user.

- Check Identity is triggered to authenticate users.

5 The Implementation of GVE

The GVE is implemented in the project and a Beta version of the GVE now is operational. The following technologies are used for the implementation:

Java API for XML-Web Services technology (JAX-WS): provide tools and libraries used to generate clients and server artifacts for Web service.

Java XML Binding technology: Used to bind specific XML Schema to java data type.

Hibernate 3.0: Provide libraries for object relational mapping and transaction security.

Apache Tomcat as Java Servlet Container.

Many other librairies of the Apache Software foundation.

5.1 GVE Agent Service

The GVE Agent Web service is described in Web Service Description Language (WSDL) and generated into Java Class skeleton. Fig.4 shows the UML Diagram of the GVE Agent interface generated from WSDL. Now GVE Agent has been implemented for the VMWare ESX Server. The VMWare ESX Server provides a Web service interface and software artifacts for VM management. One important task in the Agent implementation is to organize VM information of the ESX Server in GLUE Schema [18]. In the UML Diagram represented in Fig. 4, the *HostType* class is correspondent to the GLUE Schema *Host* element.

5.2 Virtual Machine Database

It is no need for the Agent to store all information collected about virtual machines in the database since the VM data are dynamic, i.e. IP Address, available memory. However the Agent has to identify each virtual machine to control its GVE related lifecycle: a VM could be free; it could be reserved or be acquired by a user. It is therefore necessary to store the state of every VM. Fig.5 shows the interface to the database. Every request sent to the agent is called a *job* and the state of a *job* is stored in the database as a *JobEntity*.

A *HostEntity* object is used to save the state of a VM. It can also be used to retrieve the user which currently owns the VM. A *UserEntity* object describes a GVE Site Service. It contains for each GVE Site Service the total amount of resource that has been granted to it. The data are useful for resource allocation policy. The implementation does not make any resource restriction for requesting GVE Site Service. The data provider is implemented in SQL by using *Hibernate 3.0* object relational mapping libraries.

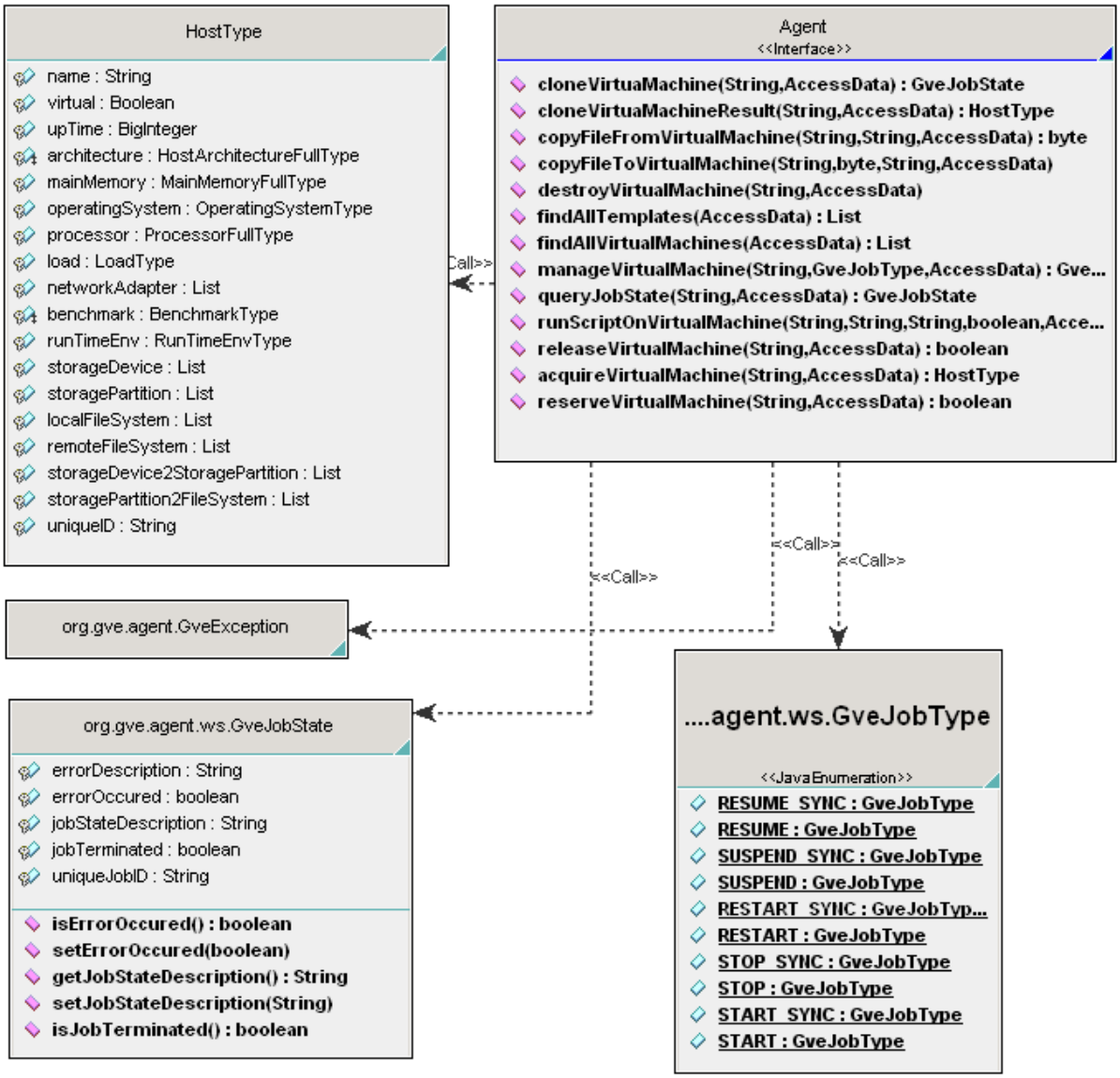


Fig. 4. The GVE Agent interface

5.3 GVE Site Service

Every VM request sent to the GVE Site Service is handled by newly created Java thread. The state of the job is also stored in a database using a data provider implemented with hibernate object relational mapping (Fig. 5). This implementation has a security policy based on *username*/*password* authentication. The total amount of VM, memory and hard drive resource allocated to a user is stored in the database. The system controls every request if the maximal limit is reached. The total amount of VM is fixed to be *infinite*. The maximal amount of main memory to *10 GB* and the Hard Disk is fixed to *infinite*.

The GVE administrator needs to explicitly register new agents to the GVE. For evaluation purpose, a Web interface for the administration of the GVE has been implemented. The Web administration interface (Fig. 6) is built to test VM acquisition and management functionality.

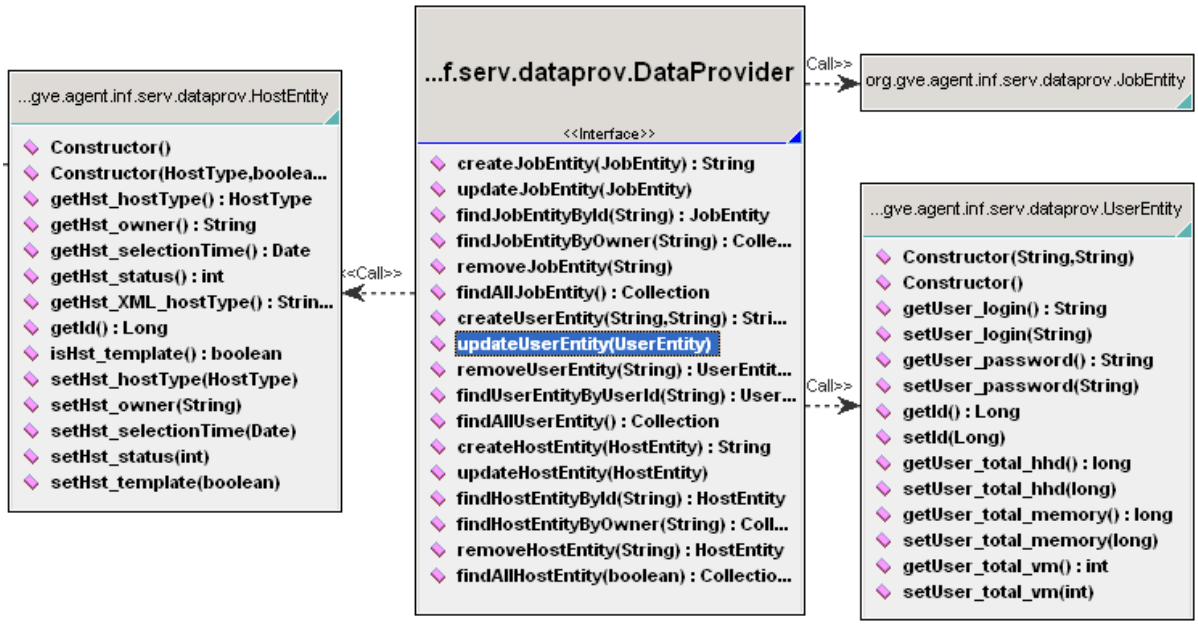


Fig. 5. Agent data provider

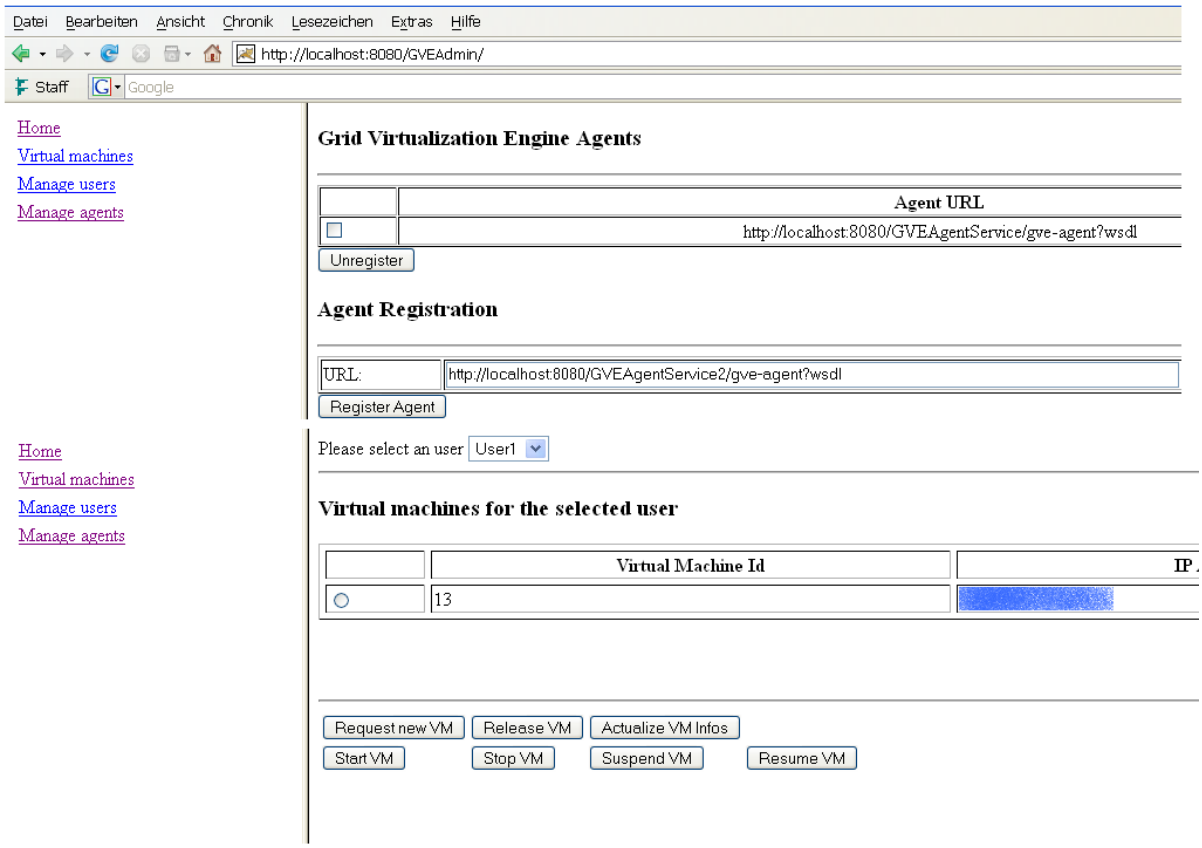


Fig. 6. Web interface of GVE administration

6 Conclusion and future work

In this paper we present the work of Grid Virtual Engine (GVE). GVE is a software layer which resides between users and various VM implementations. GVE provides a standard Web service interface for users to manipulate virtual machine resources, thereafter to form a Grid computing system. The GVE adopts a scalable distributed architecture which can help deploy scalable, manageable and efficient Grid infrastructures. Users therefore can find an easy and customized computing environment.

GVE is an ongoing work and will be developed and deployed in D-Grid [19] test bed and be further evaluated with some typical Grid workflow applications, e.g., CMS benchmark and protein alignment application.

References

1 I. Melzer etc. Service-orientierte Architecturen mit Web Service.

2 Open Grid Forum. GLUE Schema Specification version 1.2 Final Specification.

3 I. Foster, What is the Grid? A Three Point Checklist. Technical Report. Argonne National Laboratory & University of Chicago.

4 I. Foster and C. Kesselman. The Grid: Blueprint for a new computing infrastructure, Morgan Kaufmann, 1998.

5 I. Foster and C. Kesselman. The Grid: Blueprint for a new computing infrastructure, 2nd edition, Morgan Kaufmann, 2003.

6 L. Wang, M. Kunze. Information service of virtual machine pool and the interface to VDS workflow system. LNCS Vol. 4854, 2007, Rennes, France.

7 J. E. Smith, R. Nair. Virtual Machines: versatile platforms for systems and processes. Morgan Kaufman Publisher.

8 P. Barham, etc. Xen and the Art of Virtualization. In Proc. of the 19th ACM Symp. on Operating Systems Principles, pp. 164–177, New York, USA, Oct2003. ACM Press.

9 VMware virtualization products. http://www.vmware.com.

10 User Mode Linux. http://user-modelinux.sourceforge.net.

11 R. Figueiredo, P. Dinda, and J. Fortes. A Case for Grid Computing on Virtual Machines. in Proc. of the 23rd Int'l Conf. on Distributed Computing Systems. 2003.

12 K. Keahey, K. Doering, and I. Foster. From Sandbox to Playground: Dynamic Virtual Environments in the Grid. in Proc. of the 5th Int'l Workshop in Grid Computing. 2004.

13 VM-Shop Project. http://www.acis.ufl.edu/~aganguly/vmshop/

14 S. Adabala, etc. From Virtualized Resources to Virtual Computing Grids: the In-VIGO System. Future Generation Computer Systems, 21(6):896–909, June 2005.

15 J. Chase, D. E.Irwin, and L. E.Grit, J. D.Moore, and S. E.Sprenkle. Dynamic Virtual Clusters in a Grid Site Manger, in Proc. of the 12th IEEE Int'l Symp. High Performance Distributed Computing, June 2003.

16 X. Jiang and D. Xu. SODA: a Service-on-Demand Architecture for Application Service Hosting Utility Platforms, in Proc. of the 12th IEEE Int'l Symp. High Performance Distributed Computing, Seattle, WA, June 2003.

17 C. Sapuntzakis, etc. Virtual Appliances for Deploying and Maintaining Software, in Proc. of the 17th Conf. on Large Installation Systems Administration. pp. 181-194, 2003.

18 Open Grid Forum, GLUE schema Working Group. GLUE Schema v1.3, Jan. 2006.

19 D-Grid Inittiative. http://www.d-grid.de/

High Performance Multigrid on Current Large Scale Parallel Computers

Tobias Gradl, Ulrich Rüde

Lehrstuhl für Systemsimulation
Friedrich-Alexander-Universität Erlangen-Nürnberg
Cauerstr. 6
D-91058 Erlangen
tobias.gradl@informatik.uni-erlangen.de
ulrich.ruede@informatik.uni-erlangen.de

Abstract: Making multigrid algorithms run efficiently on large parallel computers is a challenge. Without clever data structures the communication overhead will lead to an unacceptable performance drop when using thousands of processors. We show that with a good implementation it is possible to solve a linear system with 10^{11} unknowns in about 1.5 minutes on almost 10,000 processors. The data structures also allow for efficient adaptive mesh refinement, opening a wide range of applications to our solver.

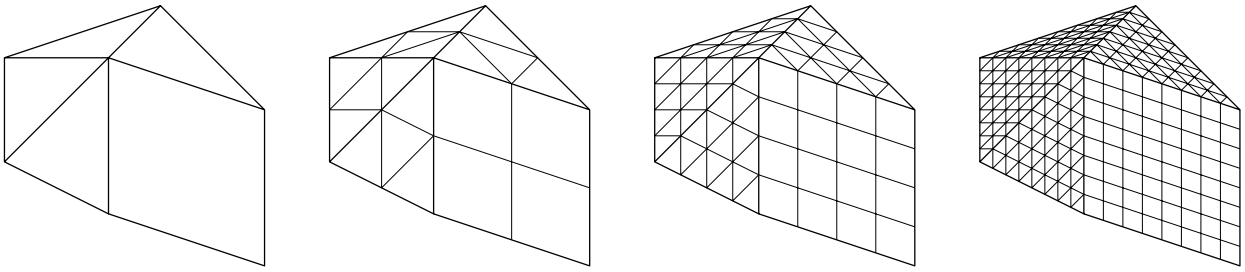
1 Introduction

In the most recent TOP-500 list, published in November 2007, HLRB II at the Leibniz Computing Center of the Bavarian Academy of Sciences is ranked at position 15 for solving a linear system with 1.58 million unknowns at a rate of 56.5 Teraflops in the Linpack benchmark. However, this impressive result is of little direct value for scientific applications. There are few real life problems that could profit from the solution of a general dense system of equations of such a size. The test problem reported in this article is a finite element discretization on tetrahedral 3D finite elements for a linear, scalar, elliptic partial differential equation (PDE) in 3D, as it could be used as a building block in numerous more advanced applications. We have selected this problem, since it has a wide range of applications, and also, because it is an excellent test example for any high performance computer architecture. Our tests on HLRB II show that this computer is well suited and yields high performance also for this type of application.

HLRB II, an SGI-Altix system, went into operation in September 2006 with 4096 processors and an aggregate main memory of 17.5 Terabytes ("phase 1"). In April 2007, the system was upgraded to 9728 cores and 39 Terabytes of main memory ("phase 2"). In particular in terms of available main memory, it is currently one of the largest computers in the world. Though HLRB II is a general purpose supercomputer, it is especially well suited for finite element problems, since it has a large main memory and a high bandwidth.

With our article we would like to demonstrate the extraordinary power of today's comput-

Figure 1: Regular refinement example for a two-dimensional input grid. Beginning with the input grid on the left, each successive level of refinement creates a new grid that has a larger number of interior points with structured couplings.



ers for solving finite element problems, but also which algorithmic choices and implementation techniques are necessary to exploit these systems to their full potential.

2 Hierarchical Hybrid Grids

In this article we focus on multigrid algorithms [BHM00, TOS01], since these provide mathematically the most efficient solvers for systems originating from elliptic PDEs. Since multigrid algorithms rely on using a hierarchy of coarser grids, clever data structures must be used and the parallel implementation must be designed carefully so that the communication overhead remains minimal. This is not easy, but our results below will demonstrate excellent performance on solving linear systems with up to 3×10^{11} unknowns and for up to almost 10,000 processors.

HHG ("Hierarchical Hybrid Grids") [BGHR06, BHR05] is a framework for the multigrid solution for finite element (FE) problems. FE methods are often preferred for solving elliptic PDEs, since they permit flexible, unstructured meshes. Among the multigrid methods, algebraic multigrid [Mei06] also supports unstructured grids automatically. Geometric multigrid, in contrast, relies on a given hierarchy of nested grids. On the other hand, geometric multigrid achieves a significantly higher performance in terms of unknowns computed per second than algebraic multigrid.

HHG is designed to close this gap between FE flexibility and geometric multigrid performance by using a compromise between structured and unstructured grids: a coarse input FE mesh is organized into the grid primitives vertices, edges, faces, and volumes that are then refined in a structured way, as indicated in fig 1. This approach preserves the flexibility of unstructured meshes, while the regular internal structure allows for an efficient implementation on current computer architectures, especially on parallel computers.

The grid decomposition into the primitives allows each group of primitive to be treated separately during the discretization and solver phases of the simulation, so that the structure of the grid can be exploited. For example, instead of explicitly assembling a global stiffness matrix for the finite element discretization element by element, we can define it implicitly using stencils. If the material parameters are constant within an element, the stencil for each element primitive is constant for all unknowns interior to it for a given

level of refinement. Then, of course, only one stencil has to be stored in memory for each level of that element, which is the main reason for HHG's memory efficiency and high execution speed.

3 Parallelization

To exploit high end computers, the programs must be parallelized using message passing. For an overview of parallel multigrid algorithms see [HKMR06] The HHG framework is an ideal starting point for this, since the mesh partitioning can be essentially accomplished on the level of the coarse input grid, that is, with a grid size that can still be handled efficiently by standard mesh partitioning software like Metis[1]. In order to avoid excessive latency, the algorithmic details and the communication must be designed carefully. The multigrid solver uses a Gauß-Seidel smoother that traverses the grid points in the order of the primitives of the coarse input mesh: first, all vertices are smoothed, then all edges, and so on. During the update of any such group, no parallel communication is performed. Instead, data needed in the same iteration by neighbors of higher dimension is sent after the update of a group in one large message per communication partner; data needed by neighbors of lower dimension in the next iteration can even be gathered from all groups and sent altogether at the end of the iteration (see fig 2).

This procedure minimizes the number of messages that have to be sent, and thus greatly reduces communication latency. At the same time, it guarantees an important prerequisite of the Gauß-Seidel algorithm: because primitives within a group are never connected to each other directly, but only to primitives of other groups, all neighbors' most recent values are already available when a grid point is updated. For example, faces are only connected to other faces via vertices, edges or volumes, no communication is necessary during the smoothing of the faces. This strategy only goes wrong near the corners of triangles, where edges directly depend on each other (see fig 3). Here the values from the previous iteration are used, giving the smoother Jacobi characteristics at the affected points. Numerically, this leads to only a slight deterioration of the convergence rates, but the gain in execution time more than outweighs this effect.

4 World record in linear system solving

In our largest computation to date, we have used 9170 cores of HLRB II and HHG to solve a finite element problem with 307 billion unknowns in 93 seconds run time. We believe that this is the largest finite element system that has been solved to date. Additionally, we point out that the absolute times to solution are still fast enough to leave room for using this solver as a building block in e. g. a time stepping scheme.

The results in Table 1 show the results of a scaling experiment from 4 to 9170 compute

[1] `http://glaros.dtc.umn.edu/gkhome/views/metis`

Figure 2: HHG grouping of communication

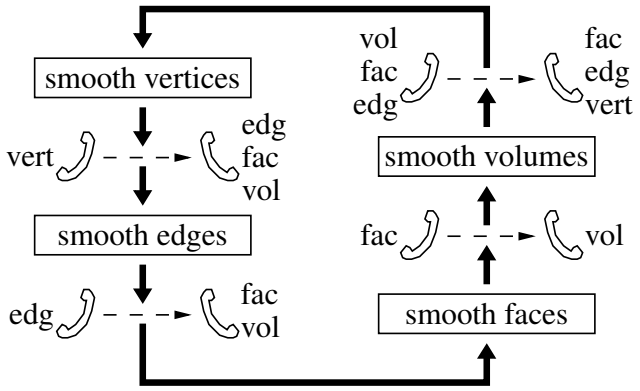
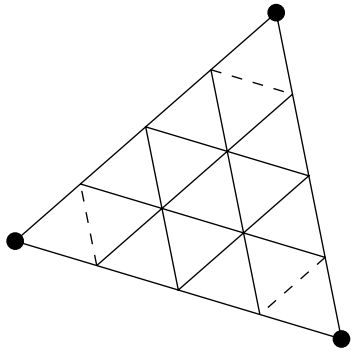


Figure 3: HHG communication and ignored dependencies



cores. The amount of memory per core is kept constant and the problem size is chosen to fill as much of the available memory as possible, which is commonly referred to as *weak scaling experiment*. If the program were perfectly scalable, the time per V cycle would stay constant throughout the table, because the ratio of problem size (i. e. workload) versus number of cores (i. e. compute power) stays constant. Near perfect scaling is seen as measure of the quality of an algorithm and its implementation. For HLRB II in installation phase 2, the computation time increases by only 20% when scaling from 4 to 9170 cores. This is still not perfect but in our view acceptable, especially when compared to other algorithms and especially in terms of the absolute compute time. Note that perfect scalability is the more difficult to achieve the faster a code is. The shorter the time spent in actual calculations is, the larger is the fraction of the time spent in communication, and the more pronounced are the communication overheads introduced by the scaling.

Phase 1 of HLRB II used single-core processors, providing every core with its own memory and network interface. The dual-core configuration of phase 2 provides less bandwidth per core, since two cores must now share an interface. Additionally, a part of the installation is now configured as so-called “high density partitions” where two dual-core processors share one interface, which means there is even less bandwidth available per core. Benchmark results including these high density partitions are marked with an asterisk in table 1. HHG is highly sensitive to the available memory bandwidth. The timings for 64, 504, and 2040 cores show that the dual-core processors of phase 2 account for approximately 39% deterioration in runtime compared to phase 1; compare this to the 20% of efficiency lost through scaling over the whole computer. The same effect is observed when switching between the regular and the high density partitions of phase 2. While one V cycle takes only 6.33 s on 6120 cores of the regular partitions, on the high density partitions the runtime is already 7.06 s on just 128 cores but then increases only slightly further to 7.75 s for our largest runs.

Table 1: Scaleup results for HHG. With a convergence rate of 0.3, 12 V cycles are necessary to reduce the starting residual by a factor of 10^{-6}. The entries marked with * correspond to runs on (or including) high density partitions with reduced memory bandwidth per core.

# Processors	# Unknowns	Time per V cycle (s)		Time to solution (s)	
		Phase 1	Phase 2	Phase 1	Phase 2
4	134.2	3.16	6.38 *	37.9	76.6 *
8	268.4	3.27	6.67 *	39.3	80.0 *
16	536.9	3.35	6.75 *	40.3	81.0 *
32	1 073.7	3.38	6.80 *	40.6	81.6 *
64	2 147.5	3.53	4.93	42.3	59.2
128	4 295.0	3.60	7.06 *	43.2	84.7 *
252	8 455.7	3.87	7.39 *	46.4	88.7 *
504	16 911.4	3.96	5.44	47.6	65.3
2040	68 451.0	4.92	5.60	59.0	67.2
3825	128 345.7	6.90		82.8	
4080	136 902.1		5.68		68.2
6120	205 353.1		6.33		76.0
8152	273 535.7		7.43 *		89.2 *
9170	307 694.1		7.75 *		93.0 *

5 Parallel Adaptive Grid Refinement

The paradigm of splitting the grid into its primitives (vertices, edges, faces, and volumes) and the technique of regular refinement also prove valuable when implementing adaptivity. Our remarks on this topic divide the refinement methods into two groups, those which create conforming grids, and those which do not. For an exact definition of the term *conforming grid* see [Rüd93b]; in short it means that all grid nodes lie only at the ends of edges and at the boundaries of faces. Techniques like red-green refinement [BSW83] create conforming grids and are widely used, because they are numerically unproblematic. The other group of techniques creates non-conforming grids with hanging nodes which are often considered numerically unpleasant. Yet the implementation of such a technique is especially straightforward in HHG, that is why we show how to treat the hanging nodes correctly so they do not pose a problem.

For more details we refer to an introductory article about multigrid on adaptively refined grids, with many links to related work, by Bastian and Wieners [BW06]. A paper by Lang and Wittum describes the building blocks of a parallel adaptive multigrid solver in detail [LW05].

Figure 4: Grid originating from two triangles, the lower one refined once, the upper one refined twice. The hanging nodes are encircled.

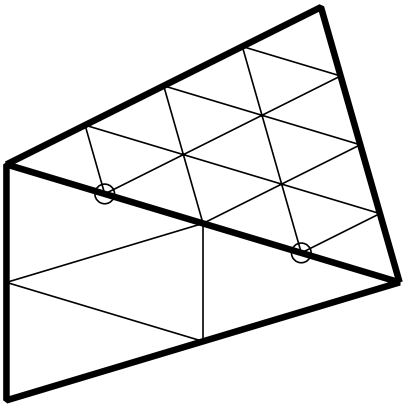
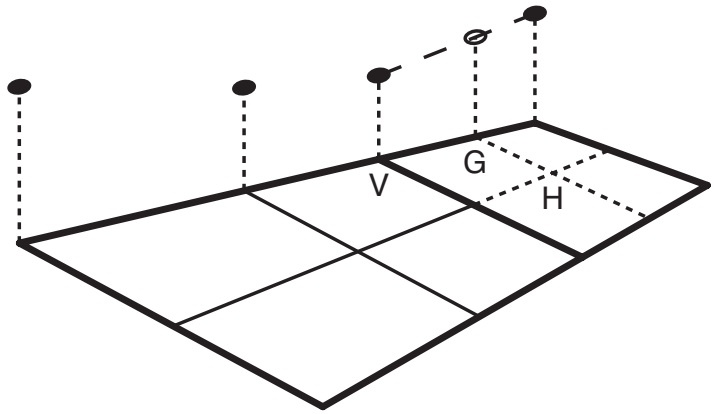


Figure 5: Two quadrilaterals, the left one with one level of refinement. For evaluating the stencil at vertex V, the values at the grid points G and H have to be interpolated.



5.1 Refinement with Hanging Nodes

In HHG, adaptive refinement can simply be achieved by allowing the coarse grid elements to be refined to different levels, with the effect of having a non-conforming grid with hanging nodes at the interfaces between elements. Figure 4 shows a non-conforming grid consisting of two triangles refined to different levels.

In HHG, a grid primitive is always refined to the finest level of all adjacent primitives of higher dimension. This ensures that the primitives of highest dimension (faces in 2D, volumes in 3D), comprising the largest part of the unknowns, are all surrounded by primitives with at least the same refinement level. Thus, they do not need any special care and can be treated without performance penalties.

An interface primitive sitting between two primitives with different levels of refinement (vertex V in fig 5) sets up its stencils just as if all adjacent primitives were refined to the finest level, with the effect that some grid points referred to by the stencils do not exist (points G and H in fig 5). These points are interpolated from points on the finest available level on their primitives. The interpolation rules—crucial for the numerical stability of the algorithm—can be derived easily by interpreting the problem from the viewpoint of *hierarchical bases* (see e. g. [Rüd93a]), with the hierarchical surplus defined to be zero on the non-existent levels.

5.2 Red-Green Refinement

The two basic rules used in red-green refinement are shown in fig 6 and explained in detail in [BSW83]. The *red* rule is identical to the one we use in our regular refinements (cf. fig 1): a triangle, for example, is refined into four new triangles by connecting its edge midpoints with new edges.. The resulting hanging nodes are taken care of by applying

Figure 6: The middle triangle is red-refined and induces green refinement in the other elements.

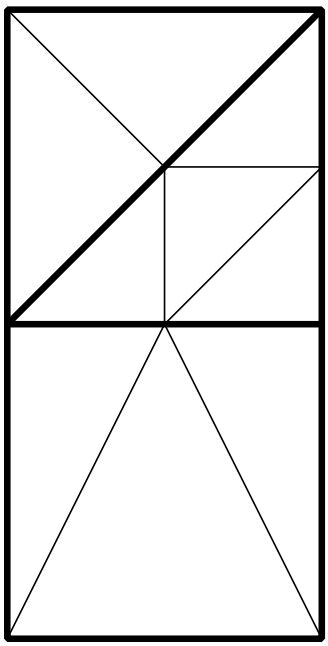
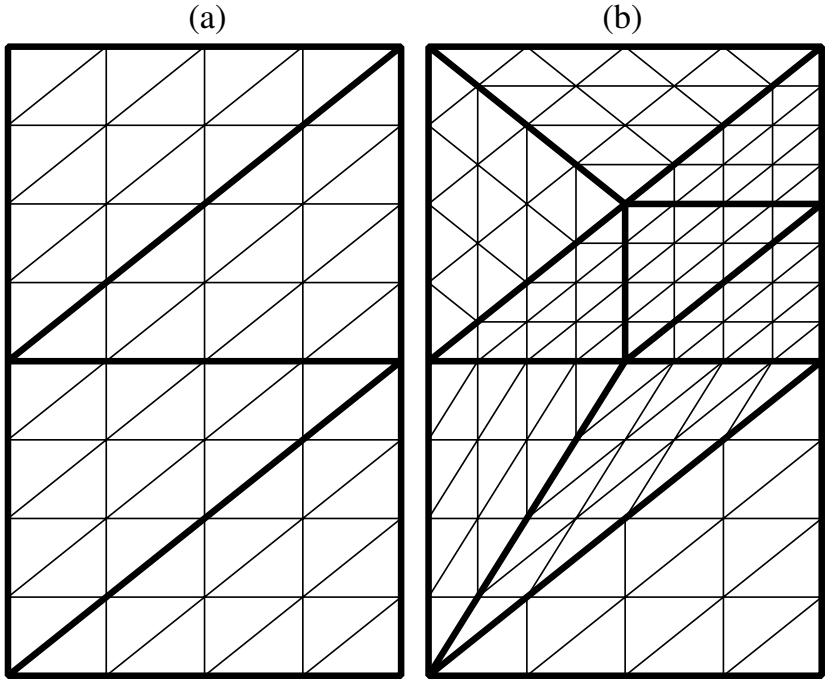


Figure 7: Red-green refinement of an already regularly refined mesh. (a) The initial mesh consists of four triangles, each refined regularly two times. (b) Red refinement in the upper right triangle induces green refinement in the upper and lower left triangles.



the *green* rule to each affected neighboring element: the element is split into two or more new elements by connecting the hanging node with one or more of the element's already existing corner nodes.

Red-green refinement can also be applied to already regularly refined elements as they occur in HHG, which is illustrated in fig 7. The refinement can—and should—be applied at the level of the coarse input grid. Then, thanks to the small number of elements in the input grid, its application is very cheap, and all the tools developed for these methods can be used. Furthermore, the regular internal grid structure of the input elements, responsible for HHG's high performance, is not harmed.

The upper right triangle in fig 7 with three interior points is split into four new triangles with three interior points each, doubling the grid resolution in this region. The structured interiors after refinement can be initialized in a natural and efficient way from the structured interiors before refinement. Some of the new grid points have the same location as old grid points, values at these points are simply inherited from the old grid. The values at the grid points in between are obtained by linear interpolation. The same applies to the neighboring triangles that have to be green-refined.

5.3 Combining Both Approaches

While each of the methods can alone be used to implement adaptive refinement, we pose that combining both results in additional advantages. Being able to do red-green refinement as well as structured refinement with varying levels, we can trade the advantages and disadvantages of both methods to obtain an optimal compromise between adaptivity and performance.

If there are not many geometrical features that have to be resolved in the domain, the initial HHG mesh can be very coarse. If it turns out during the solution process that the mesh has to be refined in some area, one or more of the very large coarse grid elements have to be refined regularly, leading to a fine mesh resolution also in areas where it is not needed. The solution to this problem is to red-green-refine the initial coarse mesh.

A disadvantage of red-green refinement is that it does not necessarily preserve the element type. A green refinement step turns a quadrilateral element into triangles (cf. fig 6). So, if purely quadrilateral/hexahedral meshes are desired, red-green refinement cannot be used.

One of the goals when setting up a simulation for HHG is to have as few coarse grid elements as possible, because then the structured areas are large and can be treated with high performance. Refinement with hanging nodes creates less coarse grid elements than red-green refinement and should thus be used whenever possible.

6 Conclusions and Outlook

The HHG framework and HLRB II have been used to solve a finite element problem of world-record size. HHG draws its power from using a multigrid solver that is especially designed and carefully optimized for current, massively parallel high performance architectures. The SGI Altix architecture is found to be well-suited for large scale iterative FE solvers. While the parallel scalability is already good, there is still room for improvement which we will exploit by further optimizing the communication patterns. The future will also bring comparative studies on other architectures, for example the IBM BlueGene. Adaptive refinement will enable us to conquer a wider range of applications than before.

Acknowledgments

We would like to point out that it was Benjamin K. Bergen who brought the idea of HHG to life within his Ph.D. thesis [Ber06]. The initial phase of the project was funded by the KONWIHR supercomputing research consortium[2]. The ongoing research on HHG is funded by the international doctorate program “Identification, Optimization and Control with Applications in Modern Technologies” within the Elite Network of Bavaria[3].

[2] http://konwihr.in.tum.de/

[3] http://www2.am.uni-erlangen.de/elitenetzwerk-optimierung

References

[Ber06] B. Bergen. *Hierarchical Hybrid Grids: Data Structures and Core Algorithms for Efficient Finite Element Simulations on Supercomputers*, volume 14 of *Advances in Simulation*. SCS Europe, July 2006.

[BGHR06] B. Bergen, T. Gradl, F. Hülsemann, and U. Rüde. A Massively Parallel Multigrid Method for Finite Elements. *Computing in Science & Engineering*, 8:56–62, November 2006.

[BHM00] W.L. Briggs, V.E. Henson, and S.F. McCormick. *A Multigrid Tutorial*. SIAM, 2. edition, 2000.

[BHR05] B. Bergen, F. Hülsemann, and U. Rüde. Is 1.7×10^{10} Unknowns the Largest Finite Element System that Can Be Solved Today? In *SC '05: Proceedings of the 2005 ACM/IEEE conference on Supercomputing*, page 5. IEEE Computer Society, 2005.

[BSW83] R.E. Bank, A.H. Sherman, and A. Weiser. Some refinement algorithms and data structures for regular local mesh refinement. In R. Stepleman et al., editors, *Scientific Computing, Applications of Mathematics and Computing to the Physical Sciences, Volume I*. IMACS, North-Holland, 1983.

[BW06] P. Bastian and C. Wieners. Multigrid Methods on Adaptively Refined Grids. *Computing in Science & Engineering*, 8:44–54, November 2006.

[HKMR06] F. Hülsemann, M. Kowarschik, M. Mohr, and U. Rüde. Parallel geometric Multigrid. In A.M. Bruaset and A. Tveito, editors, *Numerical Solution of Partial Differential Equations on Parallel Computers*, volume 51 of *Lecture Notes in Computational Science and Engineering*, pages 165–208. Springer, 2006.

[LW05] S. Lang and G. Wittum. Large-scale density-driven flow simulations using parallel unstructured Grid adaptation and local multigrid methods. *Concurrency and Computation: Practice and Experience*, 17:1415–1440, September 2005.

[Mei06] U. Meier Yang. Parallel Algebraic Multigrid Methods — High Performance Preconditioners. In A.M. Bruaset and A. Tveito, editors, *Numerical Solution of Partial Differential Equations on Parallel Computers*, volume 51 of *Lecture Notes in Computational Science and Engineering*, pages 209–236. Springer, 2006.

[Rüd93a] U. Rüde. Fully Adaptive Multigrid Methods. *SIAM Journal on Numerical Analysis*, 30(1):230–248, February 1993.

[Rüd93b] U. Rüde. *Mathematical and Computational Techniques for Multilevel Adaptive Methods*, volume 13 of *Frontiers in Applied Mathematics*. SIAM, 1993.

[TOS01] U. Trottenberg, C. Oosterlee, and A. Schüller. *Multigrid*. Academic Press, 2001.

SDVM^R: A Scalable Firmware for FPGA-based Multi-Core Systems-on-Chip

Andreas Hofmann and Klaus Waldschmidt

J. W. Goethe-University, Technical Computer Sc. Dep.,
Box 11 19 32, D-60054 Frankfurt, Germany
E-mail: {ahofmann,waldsch}@ti.informatik.uni-frankfurt.de

Abstract: As the main scope of mobile embedded systems shifts from control to data processing tasks high performance demand and limited energy budgets are often seen conflicting design goals. Heterogeneous, adaptive multicore systems are one approach to meet these challenges. Thus, the importance of multicore FPGAs as an implementation platform steadily grows. However, efficient exploitation of parallelism and dynamic runtime reconfiguration poses new challenges for application software developement. In this paper the implementation of a virtualization layer between applications and the multicore FPGA is described. This virtualization allows a transparent runtime-reconfiguration of the underlying system for adaption to changing system environments. The parallel application does not see the underlying, even heterogeneous multicore system. Many of the requirements for an adaptive FPGA-realization are met by the SDVM, the scalable dataflow-driven virtual machine. This paper describes the concept of the FPGA firmware based on a reimplementation and adaptation of the SDVM.

1 Introduction

Multicore systems are no longer restricted to the area of high performance computing. Today, as the main scope of embedded systems shifts from control oriented applications to data processing tasks even mobile phones contain chipsets which provide multiple, heterogeneous cores. To satisfy market needs those mobile devices must provide audio and video processing capabilites and therefore have high performance demands despite living on a tight energy budget.

Multicore systems consisting of heterogeneous components are one way to tackle these conflictive design goals. Further freedom of design space exploration is provided if these systems can be reconfigured at runtime and therefore adapt to changing needs of the application. For example, to optimize the power management the number of active, or even existing, cores can be adapted dynamically to the current workload. Therefore, the importance of multicore FPGAs as an implementation platform steadily grows as they allow to design parallel systems with most of the components placed on-chip. With multiple processor cores, both as fixed hardware (hardcore) or implemented using the configurable logic (softcores), and their ability to reconfigure they provide a good basis for parallel, scalable and adaptive systems.

To efficiently exploit parallel and adaptive systems the application software must cope with the changing number of existing cores and manage the hardware reconfiguration. As these features are shared by most applications running on such a system it is beneficial to provide a virtualization layer which hides the – due to runtime reconfiguration – changing hardware system from the application software. The Scalable Dataflow-driven Virtual Machine (SDVM) is such a virtualization of a parallel, adaptive and heterogeneous cluster of processing elements [Aut05, Aut04]. Thus, it is well suited to serve as a managing firmware for multicore FPGAs and to fulfill the above mentioned requirements. This paper covers the description of the main features and the general concept of the firmware.

This paper is structured as follows: Section 2 describes the fundamentals of FPGAs and gives an outline of the firmware concept. Section 3 discusses the realization of the firmware using the SDVM and the developement system architecture it is tested on. Section 4 gives an overview of related work. The paper closes with a brief conclusion in Section 5.

2 A Firmware concept for FPGAs

Besides the primary functions that a System-on-Chip (SoC) should accomplish, e.g. speech encoding in a cell phone or packet filtering in a network router, their design has to address a multitude of secondary requirements. These requirements are important for most systems, merely the weighting differs. The introduction of FPGAs as a target platform for SoCs adds an other important requirement: The runtime reconfiguration ability of some FPGAs provide additional flexibility to the system. To make optimal use of these reconfigurable systems an efficient management of the reconfiguration process is necessary.

The list of secondary requirements can be summarized as follows:

- performance and scalability
- support for parallelism
- adaptivity
- robustness and reliablity
- energy efficiency
- support for runtime reconfiguration
- incorporation of heterogeneous components

As these requirements and therefore the techniques to achieve them are common to a vast number of SoCs it is beneficial to supply a generic module which manages these supporting features. This lightens the burden of the designer who can concentrate on the primary functions of the SoC.

To avoid an increase in complexity, provide flexibility, and improve portability and code reusability through different hardware the division of the functionality into several layers is a possible solution. The aforementioned generic module should therefore be implemented

System	FPGA	Slices	RAMB16
1 MicroBlaze	Virtex4 FX20	2,025 (23 %)	36 (52 %)
2 MicroBlaze	Virtex4 FX20	3,958 (46 %)	68 (100 %)
1 PPC	Virtex4 FX20	1,158 (13 %)	36 (52 %)
1 PPC + 1 MB	Virtex4 FX20	3,067 (35 %)	68 (100 %)
2 PPC	Virtex-II Pro 30	2,096 (15 %)	38 (50 %)
4 MicroBlaze	Virtex-II Pro 30	7,513 (54 %)	132 (97 %)
2 PPC + 2 MB	Virtex-II Pro 30	5,768 (42 %)	132 (97 %)

Table 1: Resource requirements of multi-processor systems implemented on different Xilinx FPGAs

as a functional layer between the system hardware and the application software thus acting as a middleware.

The middleware should provide a complete virtualization of the underlying hardware. The application has no longer to be tailored to the hardware, instead it is sufficient to tailor it to the virtual layer. This virtual layer not only provides hardware independence, it can also hide changes in the underlying hardware due to reconfiguration. Thus, such a middleware is specifically well suited to be used as a firmware for FPGAs.

2.1 Dynamically reconfigurable platform FPGAs

A generic FPGA architecture consists of three major components: Configurable logic blocks (CLBs), input/output blocks (IOBs) and a configurable routing network that connects all IOBs and CLBs. The CLBs can be used to implement arbitrary logic functions while the IOBs provide the physical interface to the off-chip environment. Today, FPGAs are no longer limited to these basic components. They incorporate additional specialized function blocks like embedded memory or processor cores. Thus, modern FPGA can implement complete parallel system architectures on-chip, so-called microgrids.

Any function block which is implemented on an FPGA using dedicated silicon area and therefore uses no CLBs is called a hard macro. If the function block realizes a processor core it is more accurately called a *hardcore*. In contrast, processor cores which are implemented solely using the CLBs are called *softcores*. However, hardcores may require additional support logic which has to be implemented in CLBs to be fully usable.

The vast amount of CLBs enables the designer to add several softcores. As seen in Table 1 even the second smallest device of the Virtex-4 FX family can host two MicroBlaze softcores including an FPU and dedicated RAM each and still has more than 50 % free logic resources that can be used to implement application specific functions. Larger FPGAs can support systems with four cores and still have 42 % of their logic elements unsused.

2.2 The Middleware concept

To efficiently use a dynamic reconfigurable FPGA as an implementation architecture for multi-core systems the middleware has to support a number of different features.

Today, even small FPGAs can host multiple cores (See Table 1). Besides vendor-supplied softcores or embedded hardcores system designers often add application specific function blocks like digital signal processors (DSP) or specialized datapath units. Unless a distinction is necessary these cores and function blocks will be referred to as processing elements (PE) in the following.

The logic resources and therefore the computing power of the FPGA and the internal memory blocks can be distributed evenly among all PEs, but there are resources which cannot be efficiently split. The most important one being the external memory. As FPGAs typically have only up to some hundred kilobytes of internal memory – the smaller ones actually provide less than one hundred kilobytes – a lot of applications require external memory. Therefore, the middleware should support a multi-level memory architecture that is transparent to the application software.

Besides external memory every interface of the FPGA system to the outside world like ethernet or PCIe cannot be allocated to every PE. The middleware must manage these resources on the cluster level.

The middleware should provide a complete virtualization of runtime reconfigurable platform FPGAs. Therefore it has to support the following primary features:

- Combine all PEs on the FPGA to create a parallel system.
- Provide task mobility between all PEs even if they are heterogenous. It should be possible to execute a task on general purpose processors of different architectures and on custom function units if applicable.
- Virtualize the I/O-system to enable the execution of a task on an arbitrary PE.
- Combine the distributed memory of each PE to form a virtually shared memory. To avoid bottlenecks each PE should have its own memory both for program and data. On the other hand, applications for shared memory are much easier to design than applications for message passing systems. Thus, the distributed memory should be transparent to the applications.
- Manage the reconfiguration of the FPGA, i.e. keep track of the current usage of the FPGA resources and available alternative partial configurations. Furthermore, an adequate replacement policy has to be defined.
- Monitor a number of system parameters to gather information the configuration replacement policy depends on.
- Adjust the number of active PEs at runtime. For example, this can be used to meet power dissipation or reliability targets.
- The previous feature requires the firmware to hide the actual number of PEs from the application to ease programming.
- As the user software does not know the number and architecture of the PEs the firmware has to provide dynamic scheduling as well as code and data distribution.

One of the fundamental decisions in the design process of the firmware is whether each PE forms an independent building block of the parallel cluster or multiple PEs are merged in a higher-order cluster element. The latter may impose less overhead but the former eases

the implementation of adaptive features like coping with errors in the fabric or reducing hotspots.

If each PE is augmented with a complete set of the virtualization functions and therefore no PE is the sole provider of any function, the system is much more flexible. If an error is detected in some part of the FPGA the affected PE can be disabled or reconfigured to avoid the erroneous location without hampering the functionality of the cluster. Furthermore, as each augmented PE provides its share of the cluster management functionality the number of bottlenecks is reduced. The distribution of functionality can lead to a better distribution of workload thus reducing the number of hotspots on the FPGA.

In addition, the firmware depends to a much lesser extent on the number and type of cores in a cluster if it runs on each core independently and communicates using hardware-independant messages.

Therefore, the middleware will be implemented as a firmware running on each core of the FPGA based system. The parallel cluster is created by the communication of each firmware instance with the other instances.

3 Realization

In this section the realization of the presented virtualization concept is described. Due to its features which match the requirements specified in Section 2, the SDVM was chosen as a basis. Thus, the firmware for FPGA-based reconfigurable systems is called SDVM^R.

3.1 The Scalable Dataflow-driven Virtual Machine (SDVM)

The Scalable Dataflow-driven Virtual Machine (SDVM) [Aut05, Aut04] is a dataflow-driven parallel computing middleware (see Fig. 1). It was designed to feature undisturbed parallel computation flow while adding and removing processing units from computing clusters. Applications for the SDVM must be cut to convenient *code fragments* (of any size). The code fragments and the *frames* (a data container for parameters needed to execute them) will be spread automatically throughout the cluster depending on the data distribution.

Each processing unit which is encapsulated by the SDVM virtual layer and thus acts as an autonomous member of the cluster is called a *site*. The sites consist of a number of modules with distinct tasks and communicate by message passing. The SDVM has been implemented as a prototypical UNIX-daemon to be run on each participating machine or processor, creating a site each.

The SDVM is a convenient choice as a middleware (virtual layer) for FPGAs due to several distinguishing features. The two most important features are that the SDVM cluster can be resized at runtime without disturbing the parallel program execution, and each site in a cluster can use a different internal hardware architecture. These two features are the basis for the runtime reconfiguration ability of the system.

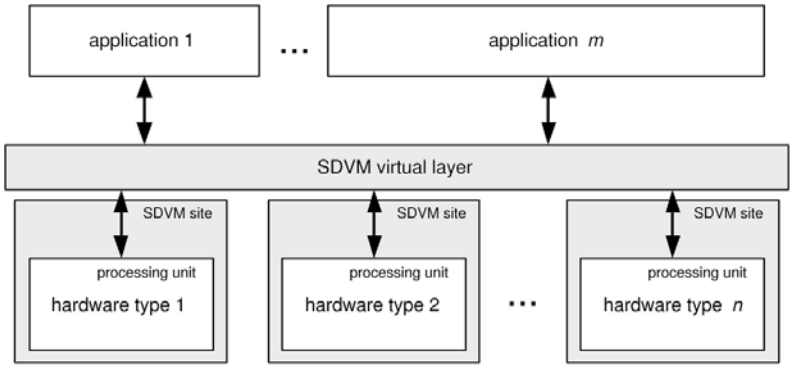


Figure 1: The processing units are encapsulated by an SDVM site each. The sites form the SDVM virtual layer. The applications don't see the underlying (possibly heterogeneous) hardware.

During a reconfiguration cycle a site drops out of the cluster, its logic estate gets reconfigured and the newly created core joins the cluster. The displacement of data and code objects is managed by the SDVM middleware. Besides changes of the sites' architectures the cluster resize mechanism can be used to adapt the cluster to different grades of parallelism of the currently running applications. On an excess of computing power sites can drop out of the cluster shutting down the affected area of the FPGA to minimize power consuption. When the demand for computing power rises free areas of the FPGA can be used to deploy new sites thus increasing the capacity of the cluster.

The integration of adaptive features into the middleware which can be used to adjust the behaviour of the system to a variety of targets requires knowledge gathered at runtime. For example, a power management policy using the SDVM was developed which can be used to improve the reliability of a multicore chip [Aut06]. Independent selection of the core's power states in conjuction with dynamic parallelism is used to minimize temperature changes of the chip thereby improving the reliability significantly while sacrificing less performance than simpler power management policies. The policy requires the collection of serveral runtime parameters like the amount of workload and current core temperature which are gathered and distributed by the SDVM.

All in all the SDVM offers the following features that are beneficial for the implementation of an FPGA middleware:

- undisturbed parallel computation while resizing the cluster
- dynamic scheduling and thereby automatic load balancing
- distributed scheduling: no specific site has to decide a global scheduling and therefore any site can be shut down at any time
- participating computing resources may have different processing speeds
- participating computing resources may use different hardcores and softcores
- applications may be run on any SDVM-driven system, as the number and types of the processing units do not matter
- no common clock needed: the clock is locally synchronous but globally asynchronous.
- support for distribution of knowledge gathered at runtime

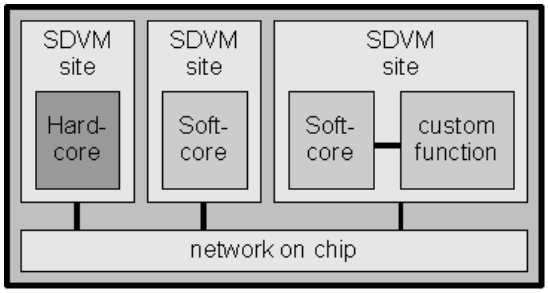


Figure 2: Each core is encapsulated by a SDVM site. The sites are implemented as software running on the cores.

3.2 Different realization schemes

One important question when implementing a parallel system on an FPGA is, how the reconfigurable area provided by the FPGA is used. There are two primal possibilities:

1. The available resources on the FPGA are used up by configuring additional processing units. Thus, the SDVM^R cluster consists of more sites and a higher parallelism can be achieved. The number of sites can be changed by reconfiguration to adopt to the available parallelism of the application as far as FPGA resources permit.

2. The FPGA fabric is used to implement custom function units, each attached to and therefore controlled by one of the cores. The function units conform to specific code fragments which are to be executed often. The supported functions of the custom function units can be changed at runtime by reconfiguration to adapt the system to the needs of the application.

The different approaches can be combined (see Fig. 2). This is especially aided by the fact that both the MicroBlaze and the PowerPC hardcore provide a fast low-level interface to the FPGA logic fabric. In this way the middleware still runs as software on the core while the some of the data processing is shifted to specialized hardware (i.e. the logic fabric). The realization of the middleware functions as hardware modules is an option for the future.

3.3 Developement system architecture

For the developement of the SDVM^R firmware a scalable system has been created using the Xilinx EDK 8.1 software. The system is based on IP blocks supplied by Xilinx which are included in EDK. As the processing element the MicroBlaze softcore is used. It is supported by a timer and an interrupt module connected to a local On-Chip Peripheral Bus (OPB) to allow for the execution of the Xilinx XMK realtime operating system. The MicroBlaze core has 64 KB of embedded memory blocks connected to its Local Memory Bus (LMB) to store and execute the firmware. These four IP blocks build the basic processing element of the system.

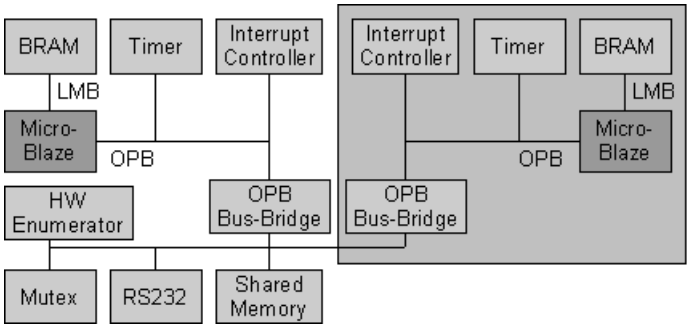


Figure 3: The system architecture of a dual core MicroBlaze system. The grey-shaded box is omitted for a single core system; for multicore systems it is replicated for each additional core.

The communication between the processing elements is done using a shared memory connected to the system OPB. To allow for mutual exclusion of the access to this memory and to the RS232 interface a hardware mutex modul is attached to the system OPB. The multiple cores are attached to the system using OPB-to-OPB bridges (See Fig. 3).

The PowerPC based system is basically the same. The MicroBlaze and its memory is replaced by the PowerPC core and a PLB-connected memory block. The resource requirements of different system configurations are listed in Table 1. The MicroBlaze is implemented with all features and FPU enabled but without caches and debugging support.

As the hardware basis a Virtex4 FX20 populated evaluation board was chosen due to its fine-grained reconfiguration features and embedded PowerPC core. Resources, especially embedded memory blocks, are quite limited on the V4FX20 to implement systems with more than two cores, so a Virtex-II Pro 30 based system is used for tests with 2 PowerPCs and up to 4 MicroBlaze cores. The busses and MicroBlazes cores of all systems run at 100 MHz clock frequency; the PowerPC cores are clocked at 300 MHz.

3.4 Preliminary performance results

The current developement version of the SDVM^R has been tested with a parallel application calculating a mandelbrot set. The application is mapped to the execution model of the SDVM^R in such a way that the calculation of a row of the mandelbrot set is done in one code fragment. Thus, the parallelism of the application increases with the number of rows to calculate. The disadvantage of this straight-forward approach is that the runtime of each instance of this code fragment differs. This is due to the mandelbrot algorithm which requires a different amount of iterations depending on the current point it is calculating.

The application has been tested on the one, two and four core systems including combinations of PowerPC and MicroBlaze cores described in Section 3.3. The results are shown in Table 2.

Despite the lower clock frequency of the MicroBlaze (100 MHz) compared to the PowerPC cores (300 MHz) the latter ones are much slower due to a missing FPU. The dualcore PowerPC system scales noticeably better that the dualcore MicroBlaze system with an

System	FPGA	CPU-Clock	Runtime	Efficiency
1 MicroBlaze	V4FX20	100 MHz	7181	—
2 MicroBlaze	V4FX20	100 MHz	3826	0.94
4 MicroBlaze	V2Pro30	100 MHz	2290	0.78
1 PPC	V4FX20	300 MHz	65574	—
2 PPC	V2Pro30	300 MHz	33045	0.99
1 PPC + 1 MB	V4FX20	300/100 MHz	6800	—

Table 2: Runtimes of a single-precision floating point mandelbrot set

efficiency of 0.99 compared to 0.94. For the four core MicroBlaze system the efficiency drops to 0.78. As the MicroBlaze system is more than 8 times faster it is more sensitive to the way the workload is distributed. As long as a site has nothing to do it keeps asking other sites for work every once in a while. Currently, this request rate is fixed. Therefore some sites may sit idle for some time even if new work is available.

4 Related work

The middleware presented in this paper uses hardware abstraction based on a virtual layer to exploit runtime reconfiguration and parallelism of currently available FPGAs on system level. Most of related work in this field either focuses on the virtualization of an FPGA on the level of the logic fabric, or on techniques for runtime reconfiguration. Furthermore the discussion of adaptive features is most often detached from concrete implementation technologies. An overview of related work in a broader sense confirm this.

Roman Lysecky et al. developed techniques for dynamic hardware/software partitioning based on online profiling of software loops and just-in-time (JIT) synthesis of hardware components called WARP [LV05]. They also present a dynamic FPGA routing approach which can be used to solve the routing and placement problem of reconfigurable components at runtime [LVT04]. However, their approach relies on a special, to our knowledge not yet implemented, FPGA architecture called Configurable Logic Architecture [LV04].

To overcome the need for this special FPGA architecture Roman Lysecky et al. developed a virtualization layer for FPGAs [LMVV05] which is placed on top of the logic fabric of a real FGPA. This virtualization layer emulates the Configurable Logic Architecture and enables the use of JIT techniques on existing FPGAs but leads to a 6X decrease in performance and a 100X increase in hardware requirements. In contrast the concept presented in our paper uses softcores and functional units specifically designed to the target FPGA family thus the overhead should be much lower.

The usage of adaptive features to tackle the complexity of modern SoCs is extensively covered by Gabriel Lipsa et al. [LHR+05]. Their paper proposes a concept that applies autonomic or organic computing principles to hardware designs. The paper does not present any kind of implementation, neither as software nor as hardware. The SDVM^R as a firmware for FPGAs is a software-realization of these autonomous principles.

5 Conclusion

In this paper a middleware providing a virtualization layer for FPGAs is presented. The virtualization layer hides the changing hardware of a reconfigurable FPGA-based system from the application software. Thereby, it allows to exploit runtime reconfiguration and parallelism of currently available FPGAs on system level. It is based on the SDVM, a middleware for computer clusters and multicore chips. Due to its features, the FPGA may reconfigure itself at runtime to adapt to changing conditions and requirements. The work is currently under developement and some preliminary performance results are presented.

References

[HDHW06] Jan Haase, Markus Damm, Dennis Hauser, and Klaus Waldschmidt. Reliability-aware power management of Multi-Core Processors, 2006. DIPES 2006, Braga, Portugal.

[HEKW04] Jan Haase, Frank Eschmann, Bernd Klauer, and Klaus Waldschmidt. The SDVM: A Self Distributing Virtual Machine. In *Organic and Pervasive Computing – ARCS 2004: International Conference on Architecture of Computing Systems*, volume 2981 of *Lecture Notes in Computer Science*, Heidelberg, 2004. Springer Verlag.

[HEW05] Jan Haase, Frank Eschmann, and Klaus Waldschmidt. The SDVM - an Approach for Future Adaptive Computer Clusters. In *10th IEEE Workshop on Dependable Parallel, Distributed and Network-Centric Systems (DPDNS 05)*, Denver, Colorado, USA, April 2005.

[LHR+05] Gabriel Lipsa, Andreas Herkersdorf, Wolfgang Rosenstiel, Oliver Bringmann, and Walter Stechele. Towards a Framework and a Design Methodology for Autonomous SoC. In Uwe Brinkschulte, Jürgen Becker, Dietmar Fey, Christian Hochberger, Thomas Martinetz, Christian Müller-Schloer, Hartmut Schmeck, Theo Ungerer, and Rolf P. Würtz, editors, *ARCS Workshops*, pages 101–108. VDE Verlag, 2005.

[LMVV05] Roman L. Lysecky, Kris Miller, Frank Vahid, and Kees A. Vissers. Firm-core Virtual FPGA for Just-in-Time FPGA Compilation (abstract only). In Herman Schmit and Steven J. E. Wilton, editors, *FPGA*, page 271. ACM, 2005.

[LV04] Roman Lysecky and Frank Vahid. A Configurable Logic Architecture for Dynamic Hardware/Software Partitioning. In *DATE '04: Proceedings of the conference on Design, automation and test in Europe*, page 10480, Washington, DC, USA, 2004. IEEE Computer Society.

[LV05] Roman Lysecky and Frank Vahid. A Study of the Speedups and Competitiveness of FPGA Soft Processor Cores using Dynamic Hardware/Software Partitioning. In *DATE '05: Proceedings of the conference on Design, Automation and Test in Europe*, pages 18–23, Washington, DC, USA, 2005. IEEE Computer Society.

[LVT04] Roman Lysecky, Frank Vahid, and Sheldon X.-D. Tan. Dynamic FPGA routing for just-in-time FPGA compilation. In *DAC '04: Proceedings of the 41st annual conference on Design automation*, pages 954–959, New York, NY, USA, 2004. ACM Press.

Adaptive Cache Infrastructure: Supporting dynamic Program Changes following dynamic Program Behavior

Fabian Nowak Rainer Buchty
Wolfgang Karl
Universität Karlsruhe (TH), Institut für Technische Informatik (ITEC)
Zirkel 2, 76131 Karlsruhe, Germany
{nowak|buchty|karl}@ira.uka.de

Abstract: Recent examinations of program behavior at run-time revealed distinct phases. Thus, it is evident that a framework for supporting hardware adaptation to phase behavior is needed. With the memory access behavior being most important and cache accesses being a very big subset of them, we herein propose an infrastructure for fitting cache accesses to a program's requirements for a distinct phase.

1 Introduction

When regarding a program's long-time behavior, it is evident that each program consists of at least three different phases: the first one can be called the *initialization phase*, the second one the *main* or *computational phase*, and the last one the *final* or *termination phase* [LY91]. It can even be shown that after millions of instructions in the so-called *main phase*, new phases of program execution commence [SPH+03]. Metrics changing from phase to phase include, but are not limited to, branch prediction, cache performance, value prediction, and address prediction [SC99, BABD03].

By providing an architecture tailored at only one phase, as is done usually, this very phase is executed with best results concerning the aspired enhancements, i.e. performance or energy efficiency. By means of reconfiguration, however, we are able to support a program during its whole run-time when adapting the hardware in every distinct phase.

In this paper, we address the basic concept, present our latest implementation results, and show in detail how much cache reconfiguration can possibly speed up program execution by giving benchmark [GRE+01] results. Furthermore, we show a simple means to handle phase changes more dynamically.

This is especially interesting for scientific super-computing where slight enhancements of the whole system can result in some fewer days of computation or less energy consumption, and therefore, cooling and money savings. Another interesting aspect is to further speed up execution of parallel computing nodes sharing some memory levels, like L2 cache and main memory, by dynamically partitioning a cache's area to the different processors' needs.

The rest of this paper is organized as follows: as a start, an overview of the finished and ongoing work is given. Upon that base, in Section 3 we present a novel architecture for supporting cache reconfiguration at reunite. The current state of our implementation is then summed up in Section 4 giving a first impression of how much speed-up can be achieved. This is explained in detail in Section 5. In order to round off the whole work, an application supporting the tool-chain is illustrated. The paper finishes with the conclusion.

2 Related work

Much work was done already in the vast field of cache partitioning and cache adaptation. As far as cache partitioning is concerned, dividing into instruction and data caches is a very well-known method for increasing cache hit-rates. Other methods are partitioning into scalar and array data caches [NKOF07] or separating by temporal and spatial locality [GAV95]. The last approach was extended by Johnson and Hwu by means of memory address tables yielding a speed-up of up to 15% [JmWH97]. They request a framework for intelligent run-time management of the cache hierarchy. Partitioning can also be used for offering instruction reuse buffers based on cache technology as is done by Ranganathan et al. in [RAJ00]. Unfortunately, some software changes have to be made for the system to work. Suh et al. examined dynamic partitioning of shared cache memory [SRD04] and come to the conclusion that re-partitioning only needs to take place when a context switch occurs.

Cache adaptation, which may benefit of cache partitioning, requires the micro-system to monitor its memory system performance, detect changing demands, and initiate reconfiguration of at least a subpart of the memory system. Benitez et al. evaluate performance and energy consumption of a cache system, which is very limited concerning possible parameter changes. With their micro-architecture of the Field-Programmable Cache Array (FPCA), they also introduce basic phase detection where branches are counted as a simple means to recognize possible changes related to the memory system [BMRL06]. When reconfiguring, any cache content is lost, thus reconfiguration has to be controlled with the processor initiating a cache flush in advance.

More sophisticated phase detection is achieved by Sherwood et al. based on basic block vectors and clustering [SPHC02]. Similarly, Balasubramonian and Albonesi managed phase detection by measuring branch frequencies and cache misses. Upon this information, cache parameter adjustment is carried out, such as sizes, associativities and access latencies [BABD03]. In their latest work, Huffmire and Sherwood applied wavelet-based phase detection onto L1 accesses and are able to accurately predict L2 miss rates, and thus, phases [HS06].

With their work towards reconfigurable cache memory, Ogasawara et al. proved that varying certain cache parameters indeed makes sense [OTW+06]. Despite delivering first results, their system is only tailored at simulation and very limited with respect to variety of dynamic parameters.

In the embedded field, Vahid and Gordon-Ross among others are developing configurable cache controllers with a strong focus on embedded applications and energy consumption [ZVN03, GRVD04, GRV07, GRZVD04].

A concept of the required tool-chain for a reconfigurable system was worked out by Mei et al. [MVML01]. It consists of a two-level approach that is based on profiling and mapping and should be adaptable to the hardware infrastructure under development at our chair.

3 Reconfigurable Cache Architecture

The first goal of our architecture was the creation of a cache capable of acting both as Level-1 and Level-2 cache. Secondly, it has to be reconfigurable. As a last aspect, we want the cache controller to be synthesizable for hardware usage.

While most reconfigurable caches depend on the reconfiguration capabilities of the underlying FPGA chip, our implementation handles the reconfiguration logically, only by hardware logic in the controller itself. This decision offers a great degree of freedom in choosing different sizes for both the cache memory and its control and replacement information, in running with different associativities and in changing replacement strategies.

The cache only caches accesses to the ordinary main memory, which consists of DDR-SDRAM on the Xilinx ML310 and ML403 evaluation boards available at our institute.

3.1 Cache Structure

As already mentioned, we decided to split the whole cache into three distinct areas: *cache memory*, *control memory* and *replacement memory*. The *control memory* indicates whether the according cache line is valid, modified, partly valid and additionally stores the tag. The *replacement memory* is only needed, when an associativity of more than one and a more sophisticated replacement strategy like LRU is used. In Figure 1, the layout of a 2-way set-associative cache with eight lines and a replacement strategy like FIFO, Pseudo-LRU, or LRU is illustrated.

3.2 Cache Controller

The cache controller handles several things: not only is it responsible for serving read/write requests to the cache memories explained above, but also for initiating the reconfiguration process, which in return is executed by the *reconfiguration controller*, and providing a *monitoring unit* for access data aggregation. The controller employs distinct buses for its individual tasks: it interfaces to the external DDR-SDRAM, to which memory read/write requests from the PLB are forwarded if no entry is found in the cache. Furthermore,

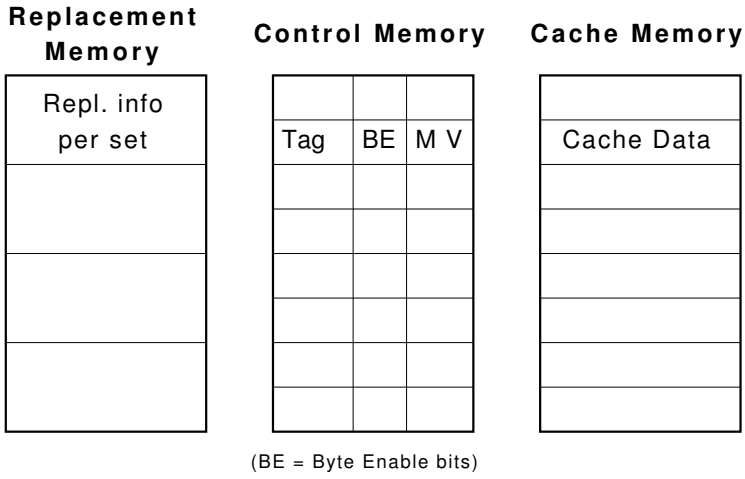


Figure 1: Exemplary cache layout

the controller features a dedicated DCR bus connection for reconfiguration purpose and monitoring control, and a dedicated monitoring interface to output monitor data.

The controller's *monitoring unit* provides therefore a 64 bit wide output register containing information about cache hits and misses, the conflict-causing line number, and additionally the complete memory address. Hence, it is possible to both collect information for on-line and off-line analysis, and achieve phase prediction such that the processor can initiate cache reconfiguration itself or have the system automatically adopt to the detected phase changes. The monitor output register is depicted in Figure 3.

As a reaction to program phases, reconfiguration might take place. Cache reconfiguration is achieved by requesting the controller to enter the reconfiguration state, then writing new parameterization values, explicitly indicating the end of the reconfiguration request, and waiting for the reconfiguration process to finish. Meanwhile, the processor could do different tasks not involving the cached main memory. But for ease of implementation and in order to avoid additional code for busy waiting, we decided to simply halt the processor. In [NBK07], we already proved that several traditionally fixed cache parameters are reconfigurable and presented some implementation approaches using heuristics where necessary. The complete process is illustrated in Figure 2.

4 Implementation Results

Figure 4 illustrates the complete extended cache/memory controller, including all interfaces and sub-modules controlling these interfaces or accessing them.

We are currently in process of synthesizing the whole design for the Virtex-II Pro and the Virtex-4 FX12 in order to benchmark the system in real hardware. First results show that some modifications still have to be made to the design; especially, the maximum clock rate is only at 9.779 MHz. This is due to the fact that the processor clock is also used for lots of comparisons for memory accesses dependent on the chosen associativity and number

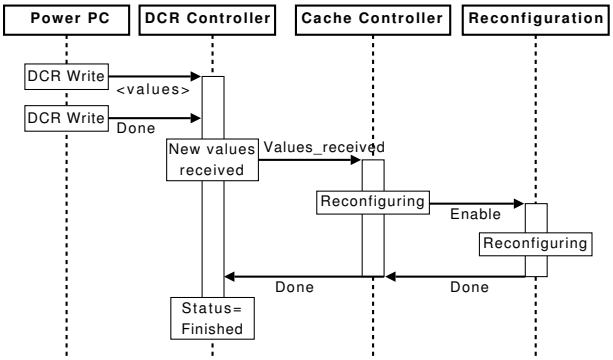


Figure 2: Reconfiguration flow

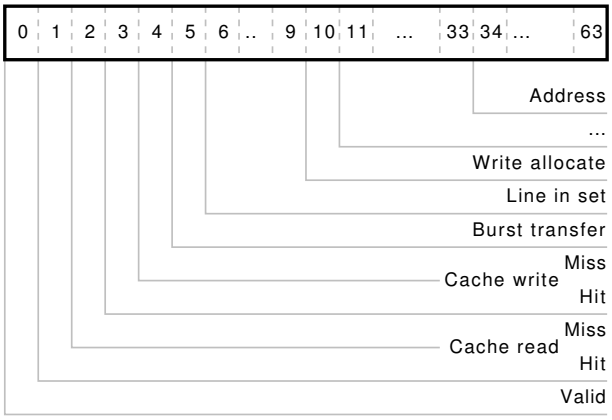


Figure 3: Monitor output register: the unused bits are intentionally reserved for further extensions to the current monitoring interface.

of sets. Hence, the core is currently undergoing re-design for a Virtex-4 FX100 to better match current FPGA's hardware resources, achieving much improved clock rates.

In Table 1, we present the hardware resource usage of parts of the current implementation targeting the Virtex-II Pro, that is to say of the cache controller, the reconfiguration module, the DCR controller, the monitoring component and the basic PLB DDR Components. We again want to emphasize that these numbers reflect the very first concept study and are not to be taken as architecture-optimized numbers for production use in computing systems.

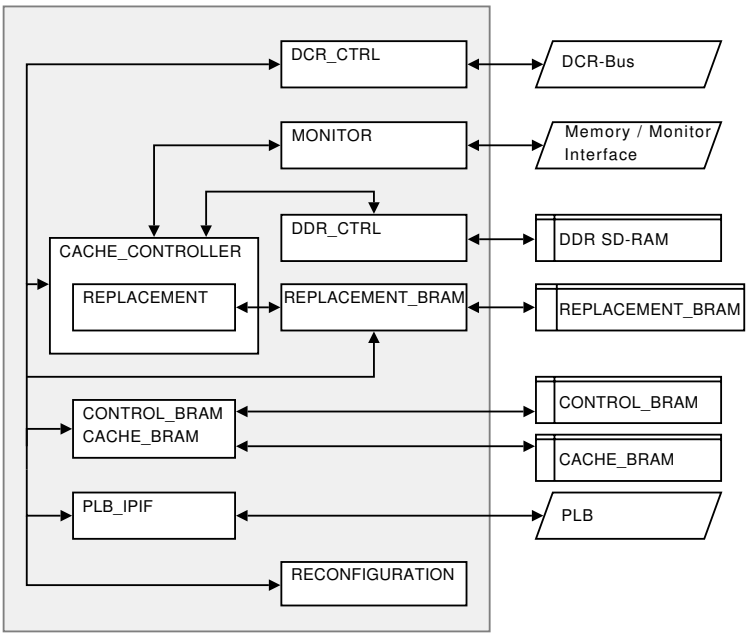


Figure 4: Reconfigurable Cache Controller Architecture

Logic Utilization	Used	Available	Utilization
Total Number Slice Registers	2089	27392	7.6%
Number used as Flip Flops	1826		
Number used as Latches	263		
Number of 4 input LUTs	37980	27392	138.7%
Logic Distribution			
Number of occupied Slices	20212	13696	147.7%
Total Number of 4 input LUTs	38660	27392	141.1%
Number used as logic	37980		
Number used as a route-thru	680		
Number of MULT18X18s	2	136	1.471%
Number of GCLKs	10	16	62.5%
Total equivalent gate count for design	268036		
Additional JTAG gate count for IOBs	79488		

Table 1: Hardware resource usage of synthesized components.

5 Application Speed-Up

Given the benchmark results from [GRE+01], we will show how much speed-up can be obtained. First, we write down the rather simple formula for calculating whether reconfiguration is appropriate and for comparing the overall memory access time with reconfiguration to the overall memory access time without adaptation to program phases.

$$\begin{aligned} & n_{wm} * t_{wm} + n_{wh} * t_{wh} + n_{rm} * t_{rm} + n_{rh} * t_{rh} && > \\ & n'_{wm} * t_{wm} + n'_{wh} * t_{wh} + n'_{rm} * t_{rm} + n'_{rh} * t_{rh} && + t_{reconfiguration} \end{aligned} \quad (1)$$

where n_{wm} indicates the number of write misses, t_{hr} the time required for serving a read request when a hit in the cache occurs, and n' the numbers achieved after reconfiguration. As can be seen easily, the overall memory access time is the sum of the cycles needed in all program phases plus their respective reconfiguration times.

Then we want to outline the time needed for both cache accesses and reconfiguration time measured in clock cycles. Table 2 gives a comparison of the implemented memory access times with and without our cache. Obviously, most speed-up can be achieved by increasing the read hit rate, while the write accesses do not contribute too much.

Access type	Duration w/o cache	Duration w/ cache
Read Hit	15	$8 + \lfloor n/2 \rfloor$
Read Miss	15	$15 + \lfloor (a-1)/2 \rfloor$
Write Hit	9	$8 + \lfloor n/2 \rfloor$
Write Miss	9	$10 + \lfloor (a-1)/2 \rfloor$

n denotes the line in the set where the hit occurs; a denotes the level of associativity.

Table 2: Access times to main memory without and with cache (using write-through)

Already upon that basis, we are able to extend Equation 1 to the following for a one- and two-way set-associative cache:

$$\begin{aligned} c * (p_{wm} * 10 + p_{wh} * 8 + p_{rm} * 15 + p_{rh} * 8) \quad & > \\ c' * (p'_{wm} * 10 + p'_{wh} * 9 + p'_{rm} * 15 + p'_{rh} * 9) \quad & + t_{reconfiguration} \end{aligned} \tag{2}$$

where $n_\alpha = p_\alpha * c$, c the number of memory access cycles "per phase", and $\sum_i p_i = 1$. Of course, we have to pay attention to use the best case access times for the first part, while in contrast the worst case has to be regarded for the second part.

Regarding reconfiguration time, it must be noted that for area and memory concerns, the reconfiguration of associativity only saves half of the cache's content. This decision makes reconfiguration up to 37% faster. The process is thus split into two phases with the first one being responsible for assuring cache consistency of the "rear half" by executing write-back where necessary and the second one moving cache lines to their new locations. The first step takes either 2 or 12 cycles per cache line, depending on whether write-back is needed. If write-through is used for write accesses, this step is only responsible for invalidating the "rear" cache lines. The second phase then takes 3 cycles per cache line for rearranging cache line data (this is indeed where reconfiguration would need double the time if the whole cache content was kept).

Hence, for doubling associativity, a one-way set-associative cache with 1024 lines and write-through strategy requires $512 * 2 + 512 * 3 = 2560$ cycles (ignoring the few setup cycles of each reconfiguration step).

Now, the memory access count stays the same when executing a disinct phase with a different configuration, thus, $c = c'$. In addition, we assume an enhancement of cache hit rate of 20% as stated in [ZVN03] when going from one-way to two-way associativity. We further assume $p_{wm} = p_{wh} = p'_{wm} = p'_{wh} = 0.25$, $p_{rh} = p_{rm} = 0.25$. With this enhancement, we get $p'_{rh} = 0.3$ and accordingly $p'_{rm} = 0.2$. Equation 2 therefore becomes

$$\begin{aligned} c * (0.25 * 10 + 0.25 * 8 + 0.25 * 15 + 0.25 * 8) \quad & > \\ c * (0.25 * 10 + 0.25 * 9 + 0.2 * 15 + 0.3 * 9) \quad & + 2560 \end{aligned} \tag{3}$$

$$\begin{aligned} & \Leftrightarrow \\ c * (0.25 * 7 + 0.5 * 15 - 0.3 * 9) \quad & > 2560 \\ & \Leftrightarrow \\ c > 2560/(1.75 + 7.5 - 2.7) \quad & = 390.84 \end{aligned} \tag{4}$$

Hence, after only 391 memory accesses, the reconfiguration effort proved sensible.

6 Toolchain Integration

In order to simplify the correct and consistent parameterization of the rather complex-to-configure cache-controller, we developed a program with a graphical user interface, which automatically adjusts all parameters with respect to the user's demand.

The *configurator* program has to be invoked manually after having designed the system-on-chip in *Xilinx Platform Studio (XPS)* [Xi07]. It then enables the user to e.g. specify a maximum associativity of 4, while in the beginning of the execution, an associativity of 2 is chosen. Furthermore, it ensures that the data widths of the replacement and control information are wide enough to store all information required. Having written the changes, the user can return to *XPS* and undertake remaining changes, synthesize the description, and download the bitstream to the device. The configuration task flow is depicted in detail in Figure 6.

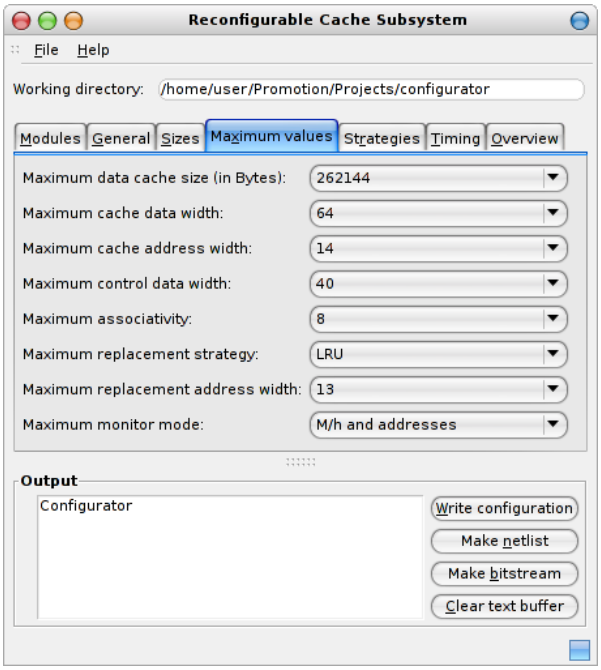


Figure 5: Graphical User Interface for a-priori configuration of the cache/memory system

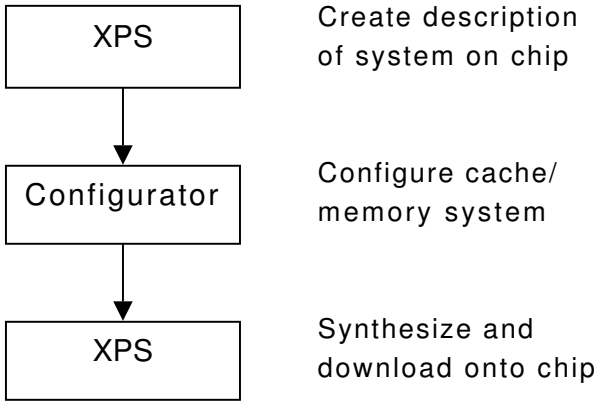


Figure 6: Task flow for initial configuration of the cache/memory system

7 Conclusions and Future Work

We have presented an architecture concept for supporting exploitation of program phase behavior by adapting a subset of the memory system – the cache system – to a program's needs. With such an architecture, it is possible to not only statically reconfigure the system every n instructions, but to have the operating system decide itself whether it seems advantageous to use another configuration. This is where the particular strength of our approach becomes obvious: the time needed for reconfiguration can be clearly estimated between

lower and upper bounds, which do not differ too much. In contrast, when using dynamic FPGA reconfiguration, parts of the system have to be halted and due to side effects, the reconfiguration time may vary unpredictably.

Future work includes the development of a monitoring component, which is capable of indicating new phases, and operating system functions for phase and system evaluation by use of the gathered monitoring information and application-embedded hints. These hints can easily be determined manually by visualization tools like in [TK05] and inserted into binary code.

We also intend to work on faster main memory access by a more direct connection of the memory to the processor and on using wider cache-lines. Additionally, offering hardware acceleration units in the unused cache area seems interesting [KST01].

References

[BABD03] Balasubramonian, R., Albonesi, D. H., Buyuktosunoglu, A., und Dwarkadas, S.: A dynamically tunable memory hierarchy. *IEEE Trans. Computers*. 52(10):1243–1258. 2003.

[BMRL06] Benitez, D., Moure, J. C., Rexachs, D. I., und Luque, E.: Evaluation of the field-programmable cache: performance and energy consumption. In: *CF '06: Proceedings of the 3rd conference on Computing frontiers*. S. 361–372. New York, NY, USA. 2006. ACM Press.

[GAV95] Gonzàlez, A., Aliagas, C., und Valero, M.: A data cache with multiple caching strategies tuned to different types of locality. In: *ICS '95: Proceedings of the 9th international conference on Supercomputing*. S. 338–347. New York, NY, USA. 1995. ACM Press.

[GRE+01] Guthaus, M. R., Ringenberg, J. S., Ernst, D., Austin, T. M., Mudge, T., und Brown, R. B.: Mibench: A free, commercially representative embedded benchmark suite. In: *WWC '01: Proceedings of the Workload Characterization, 2001. WWC-4. 2001 IEEE International Workshop on*. S. 3–14. Washington, DC, USA. 2001. IEEE Computer Society.

[GRV07] Gordon-Ross, A. und Vahid, F.: Dynamic optimization of highly configurable caches for reduced energy consumption. *Riverside ECE Faculty Candidate Colloquium*. March 2007. Invited Talk.

[GRVD04] Gordon-Ross, A., Vahid, F., und Dutt, N.: Automatic tuning of two-level caches to embedded applications. In: *DATE '04: Proceedings of the conference on Design, automation and test in Europe*. S. 10208. Washington, DC, USA. 2004. IEEE Computer Society.

[GRZVD04] Gordon-Ross, A., Zhang, C., Vahid, F., und Dutt, N.: Tuning caches to applications for low-energy embedded systems. In: Macii, E. (Hrsg.), *Ultra Low-Power Electronics and Design*. Kluwer Academic Publishing. June 2004.

[HS06] Huffmire, T. und Sherwood, T.: Wavelet-based phase classification. In: *PACT '06: Proceedings of the 15th international conference on Parallel architectures and compilation techniques*. S. 95–104. New York, NY, USA. 2006. ACM Press.

[JmWH97] Johnson, T. L. und mei W. Hwu, W.: Run-time adaptive cache hierarchy management via reference analysis. In: *ISCA '97: Proceedings of the 24th annual international symposium on Computer architecture*. S. 315–326. New York, NY, USA. 1997. ACM Press.

[KST01] Kim, H., Somani, A. K., und Tyagi, A.: A reconfigurable multifunction computing cache architecture. *IEEE Trans. Very Large Scale Integr. Syst.* 9(4):509–523. 2001.

[LY91] Lim, H.-B. und Yew, P.-C.: Parallel program behavioral study on a shared-memory multiprocessor. In: *ICS '91: Proceedings of the 5th international conference on Supercomputing*. S. 386–395. New York, NY, USA. 1991. ACM Press.

[MVML01] Mei, B., Vernalde, S., Man, H. D., und Lauwereins, R. Design and optimization of dynamically reconfigurable embedded systems. 2001. `http://citeseer.ist.psu.edu/mei01design.html`.

[NBK07] Nowak, F., Buchty, R., und Karl, W.: A run-time reconfigurable cache architecture. In: *Proceedings of the 2007 Parallel Computing Conference, Aachen/Jülich*. September 2007.

[NKOF07] Naz, A., Kavi, K., Oh, J., und Foglia, P.: Reconfigurable split data caches: a novel scheme for embedded systems. In: *SAC '07: Proceedings of the 2007 ACM symposium on Applied computing*. S. 707–712. New York, NY, USA. 2007. ACM Press.

[OTW+06] Ogasawara, Y., Tate, I., Watanabe, S., Sato, M., Sasada, K., Uchikura, K., Asano, K., Namiki, M., und Nakajo, H.: Towards reconfigurable cache memory for a multithreaded processor. In: Arabnia, H. R. (Hrsg.), *PDPTA*. S. 916–924. CSREA Press. 2006.

[RAJ00] Ranganathan, P., Adve, S., und Jouppi, N. P.: Reconfigurable caches and their application to media processing. In: *ISCA '00: Proceedings of the 27th annual international symposium on Computer architecture*. S. 214–224. New York, NY, USA. 2000. ACM Press.

[SC99] Sherwood, T. und Calder, B. The time varying behavior of programs. August 1999. Technical Report UCSD-CS99-630, University of California, San Diego.

[SPH+03] Sherwood, T., Perelman, E., Hamerly, G., Sair, S., und Calder, B.: Discovering and exploiting program phases. *IEEE Micro*. 23(6):84–93. 2003.

[SPHC02] Sherwood, T., Perelman, E., Hamerly, G., und Calder, B.: Automatically characterizing large scale program behavior. In: *ASPLOS-X: Proceedings of the 10th international conference on Architectural support for programming languages and operating systems*. S. 45–57. New York, NY, USA. 2002. ACM Press.

[SRD04] Suh, G. E., Rudolph, L., und Devadas, S.: Dynamic partitioning of shared cache memory. *J. Supercomput.* 28(1):7–26. 2004.

[TK05] Tao, J. und Karl, W.: Optimization-oriented visualization of cache access behavior. In: *Proceedings of the 2005 International Conference on Computational Behavior (Lecture Notes in Computer Science 3515)*. S. 174–181. Springer. May 2005.

[Xi07] Xilinx, Inc. Platform Studio and EDK Details. 2007. Web site: `http://www.xilinx.com/ise/embedded/edk_pstudio.htm`.

[ZVN03] Zhang, C., Vahid, F., und Najjar, W.: A highly configurable cache architecture for embedded systems. In: *ISCA '03: Proceedings of the 30th annual international symposium on Computer architecture*. S. 136–146. New York, NY, USA. 2003. ACM Press.

A Generic Tool Supporting Cache Design and Optimisation on Shared Memory Systems

Martin Schindewolf[1], Jie Tao[2]*, Wolfgang Karl[3] and Marcelo Cintra[4]

[1]Universität Karlsruhe (TH), Zirkel 2, 76131 Karlsruhe, Germany
schindew@ira.uka.de
[2]Universität Karlsruhe (TH), Zirkel 2, 76131 Karlsruhe, Germany
jie.tao@iwr.fzk.de
[3]Universität Karlsruhe (TH), Zirkel 2, 76131 Karlsruhe, Germany
karl@ira.uka.de
[4]University of Edinburgh, Mayfield Road, EH9 3JZ Edinburgh, United Kingdom
mc@inf.ed.ac.uk

Abstract: For multi-core architectures, improving the cache performance is crucial for the overall system performance. In contrast to the common approach to design caches with the best trade-off between performance and costs, this work favours an application specific cache design. Therefore, an analysis tool capable of exhibiting the reason of cache misses has been developed. The results of the analysis can be used by system developers to improve cache architectures or can help programmers to improve the data locality behaviour of their programs. The SPLASH-2 benchmark suite is used to demonstrate the abilities of the analysis model.

1 Motivation

As Moore's Law — the number of transistors per die doubles every 18 months — still holds, higher clock rates for the cores are feasible due to shorter signal distances. Higher processor speed demands faster access to the requested data. A computer system can not exploit its computing capacity if the processor spends time waiting for the data to arrive. Subsequently, the performance increasingly relies on the efficient use of the caches. Therefore, a high cache hit rate is indispensable. The common approach is to design caches whose performance is acceptable for a wide range of applications, as this concept yields the best trade-off between performance and costs. Anyways, if only a few applications have to be considered, an application specific cache design allows for better performance and improved energy efficiency. The idea is to perform a cache miss analysis and use the results to guide the user through the optimisation process. This paper presents a tool, that helps system developers to discover application specific cache parameters (such as cache

*Dr. Jie Tao is now at the Institute for Scientific Computing, Forschungszentrum Karlsruhe, Hermann-von-Helmholtz-Platz 1, 76344 Eggenstein-Leopoldshafen, Germany

size, line size). Further, programmers are supplied with the cause of the cache miss for performing source code optimisation (see section 4.2). A concise analysis model is designed and implemented. The analysis is based on the cache event trace acquired from the SIMICS simulation environment. Cache misses are classified according to their cause. Analysis results help the designers to deduce the best cache configuration for individual applications. Furthermore, an interface to an existing visualisation tool is implemented, enabling a graphical representation of the analysis results. Information in this form allows the human user to easily detect the optimisation target and identify bottlenecks.
The remainder of this paper is organised as follows. Section 2 first gives a brief introduction to related work. This is followed by a detailed description of the proposed analysis model and its implementation in section 3. Evaluation results are presented in section 4. The paper concludes in section 5 with a brief summary.

2 Related Work

Any cache optimisation, either targeting the cache architecture or the source code, relies on the knowledge about the cache miss reasons. During the last decades, cache miss analysis is in the focus of computer architecture research. Dubois et al. [DSR+93] present an approach towards cache miss estimation on multi-processor systems. Their work defines cache miss metrics assuming infinite caches. Consequently, the caches need no replacement strategy and reveal no conflict or capacity misses. Under these circumstances, their *Pure True Sharing Misses* are the same as the *true sharing misses* and the *true sharing invalidation misses* (definitions are given in section 3.1.1). Beyls and D'Hollander [BD01] introduce the *reuse distance* as a concept for cache miss estimation. This work implements the reuse distance concept and uses it to distinguish between cold, conflict and capacity misses. Jeremiassen and Eggers [JE95] demonstrate how compiler techniques avoid false sharing misses. The techniques identify shared data and use padding to extend it to the full cache line size.
Our approach combines the methodologies of the first and the second work. Similar to the first work, we consider multi-processor systems and, hence, also calculate coherence misses. Furthermore, we incorporate the cache metrics of the second work to target realistic cache architectures. An accurate algorithm was designed for this computation.

3 The Cache Model

The base of the analysis model is a cache event trace that records every cache event. The *g-cache* module of SIMICS generates this event trace. The *parse simics conf* tool captures the cache configuration and delivers it to the analysis tool. The analysis tool processes every cache event and classifies the misses.

For accuracy and flexibility we used SIMICS to provide the cache event trace. SIMICS is an efficient and instrumented system level instruction set simulator [Rev06]. SIMICS

simulates the hardware running the operating system and the application. Caches are configurable modules called *g-cache*.

3.1 Tool Chain

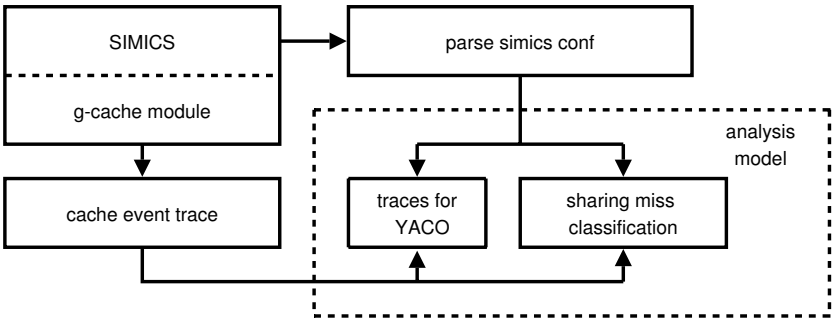


Figure 1: Tool chain interfacing with SIMICS.

Figure 1 depicts the interface between SIMICS and the developed tool, where the left side represents the simulation environment and the right side shows our own work. The *g-cache* module of SIMICS was slightly modified in order to capture the cache events, which are written to a trace file. Each cache event is processed by the analysis model. For this task the analysis model also needs the cache configuration parameters which have been used by SIMICS to generate the trace. This information is delivered by the *parse simics conf*-tool. The main work of the analysis model is to find out the reason of each cache miss. A statistical output is delivered to the user for comparing the cache behaviour of different configurations. Analysis results are also recorded in traces required by the existing visualisation tool YACO. The visualisation presents the analysis results in a user-understandable way. This helps the programmer to detect access bottlenecks and optimisation strategies.

3.1.1 Cache Miss Categories

Traditionally, cache misses are classified as cold, conflict and capacity misses [HS89]. Cold misses are caused by the first reference. Capacity misses occur when the cache is smaller than the working set size, while conflict misses occur due to mapping conflicts. For multi-processor machines the coherence problem has to be solved - this results in coherence misses. When many processors execute an application, data are shared. A write invalidate protocol invalidates the modified, shared data, thus, causing cache misses. To see whether the invalidation is necessary, the sharings are differentiated in *true sharing* (at least two processors access the same data) and *false sharing*.

A sharing miss is defined straightforwardly as a miss occurring on an address having a block address which has been shared. *True sharing misses* are an inevitable effect of parallel execution. However, *false sharing misses* shall be eliminated. Sharing misses are typically identified by examining whether the miss is caused by an earlier invalidation.

Actually, this calculation is not exact, because the replacement strategy might replace the line before the miss. Then the miss must be attributed to the replacement strategy and not classified as a coherence miss. This leads to our refined definition of coherence miss:

true sharing invalidation miss (tsim): *true sharing miss* that would not have been replaced by the local replacement strategy before the miss.

false sharing invalidation miss (fsim): analogue to the *true sharing invalidation miss*.

3.1.2 Miss Classification Implementation

In order to classify each cache miss, the cache event trace is processed. The following subsections give a short description on the implementation of the algorithms.

Recognization of Cold Miss
Each processor records every access to a memory address in a linked list. Subsequently, the first access to a block address is not found in this list. Therefore, this block address has not been accessed before by the corresponding processor. Hence, a *cold miss* is detected.

Detecting Conflict and Capacity Miss
Introduced in [BD01], the *reuse distance* is the concept to distinguish *conflict* and *capacity misses*. It is defined as the number of unique block addresses between two references to the same block. We implemented the *reuse distance* as follows. Every block address is associated with a reuse distance counter and a time stamp with the date of the last reference. Every time a miss occurs, the linked list is traversed and the time stamps of the last reference of the entries and the time stamp of the last reference of the miss are compared. If the time stamp of the entry is greater than the time stamp of the miss, the entry's reuse distance counter is increased by one. This is done because the last reference to the miss occurred before the last reference to the entry. Therefore, the block address of the miss is distinct from the other block addresses accessed since the last reference to the block address of the list entry.

The classification of *conflict* and *capacity miss* compares the reuse distance counter of the miss. If the reuse distance counter is smaller than the number of cache lines, the miss is a *conflict miss*. Otherwise, the miss is a *capacity miss*.

Sharing Invalidation Miss
For recognising the *sharing invalidation miss*, we apply another definition: *set reuse distance*. Based on the *reuse distance*, the *set reuse distance* is also the number of unique block addresses between two accesses to the same block, but only blocks mapped to the same set with the observed address are counted. This value is used to exclude misses that are caused by the replacement strategy of the cache.

If a miss is perceived on a sharing, the *set reuse distance* of that block address is compared to the number of lines in the set (associativity). If the associativity is equal or less than the set reuse distance, then this block address would already have been replaced by the LRU

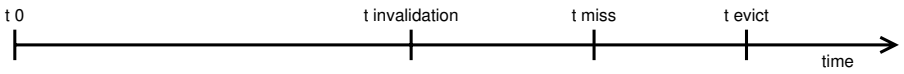


Figure 2: Timeline illustrating the sharing invalidation miss.

strategy, resulting in a *conflict* or *capacity miss*. Otherwise, a *sharing miss* is detected because the replacement strategy would not have evicted this block address. Figure 2 illustrates the *sharing invalidation miss*. The following terms are applied:

- t_0 represents the last reference to this block address which resets the *reuse distance* and the *set reuse distance* instances of this block address. Afterwards, for every different block address referenced, the *reuse distance* is increased by one and if the block belongs to the same set the *set reuse distance* is accumulated as well.
- $t_{invalidation}$ represents the time of the invalidation. The sharing of the block address is classified and saved.
- t_{miss} represents the time at which a miss occurs on that block address.
- t_{evict} is the point in time, where the LRU strategy would have replaced this block address.

If $t_{miss} >= t_{evict}$, a replacement strategy miss is detected, because the associativity is equal or less than the *set reuse distance*. On the contrary, if $t_{miss} < t_{evict}$, which correlates with the *set reuse distance* being less than the associativity, a *sharing invalidation miss* is detected (Figure 2).

4 Evaluation

Parameter	Value
L1 Line Size	32 Bytes
L1 Associativity	2-way
L1 Replacement Policy	Least Recently Used
L1 Write Policy	Write Through
L1 Allocate Policy	Write Allocate
L2 Line Size	32 Bytes
L2 Associativity	4-way
L2 Replacement Policy	Least Recently Used
L2 Write Policy	Write Back
L2 Allocate Policy	Write Allocate

Parameter	private L2	shared L2
Number of Processors	8	8
L1 Number of Caches	8	8
L1 Size (each)	4 KBytes	4 KBytes
L1 Number of Lines	128	128
Coherency Protocol	MESI	MESI
L2 Number of Caches	8	1
L2 Size (each)	128 KBytes	1024 KBytes
L2 Number of Lines	4096	32768
Coherency Protocol	MESI	None

Table 1: Common cache parameters (left) and case specific parameters for 8 processors (right).

In order to verify the functionality, the cache analysis model has been evaluated using the SPLASH-2 benchmark suite. This section first briefly describes the benchmarks and the results obtained with the multi-processor configuration. Then we show a sample view of the visualisation.

4.1 Results with SMP-architectures

For evaluating the analysis model and achieving valuable conclusions for cache optimisation, several experiments have been conducted. Throughout these experiments, the caches are configured as shown in Table 1. This setup allows for examining the SPLASH-2 Benchmark Suite [spl] towards scaling performance on SMP-architectures. SPLASH-2 aims at evaluating cache-coherent shared memory architectures. In order to adapt the SPLASH-2 programs to the x86 architecture and SIMICS the hints given by [Bar, Hei] are performed. The m4 macros [Sto], developed by Bastian Stougie [Sto03], are used to parallelise the benchmark.

Corresponding to the existing and emerging multi-processor cache designs, this experiment examines different structures of the level 2 cache(s), **private** (each processor has an exclusive level 2 cache) and **shared** (one level 2 cache for all processors). Table 1 depicts the applied configuration, with the left side for the common parameters and the right side for the L2 specific one (a sample configuration with 8 processors). This work, as well as many other studies concerning cache miss estimation, uses normalised miss rates: $miss_{rate} = \frac{miss_{sum}}{access_{sum} * number\ of\ processors}$.

4.1.1 Overall Performance

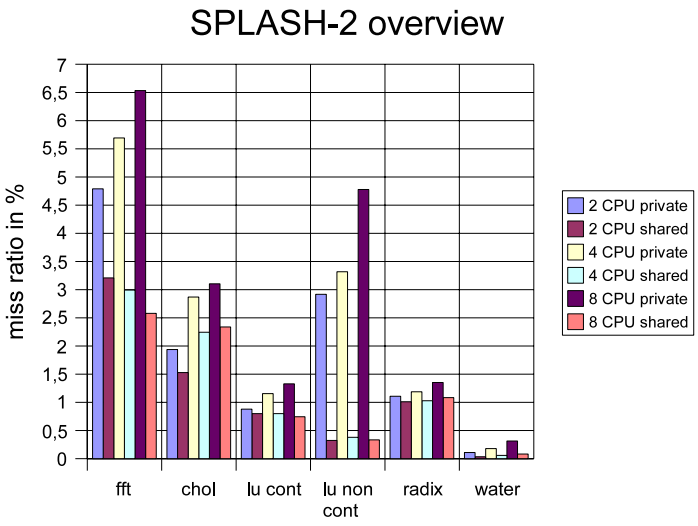


Figure 3: Miss rates grouped by SPLASH-2 programs.

Figure 3 shows the overall miss rate of the benchmarks simulated with 2, 4 and 8 processors. Two adjacent bars belong to the same number of CPUs, the left bar gives the miss ratio for privately owned caches, whereas the right bar refers to the shared case. Surprisingly, for all applications with all processor numbers the shared level 2 cache yields better performance. The improvements of a shared level 2 cache over private level 2 caches, calculated by, $\frac{miss_{private} - miss_{shared}}{miss_{private}}$, range from 9% (LU with continuous blocks measured using 2 CPUs) up to 89% (LU with non continous blocks using 2 CPUs).

For a better understanding of the reasons for the observed results, the misses are further classified (according to section 3.1.1).

4.1.2 Miss Characteristics

The analysis model computes accurately the number of misses in each miss category. This allows us to observe the cause of every cache miss. Figure 4-6 show sample results with the *cholesky*, the *water-n*^{2}, and the *lu with non continuous blocks* programs.

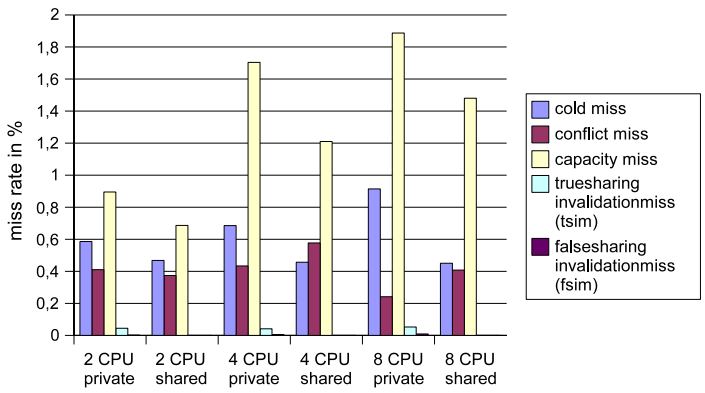


Figure 4: *cholesky* program for shared and private level 2 caches.

Cholesky
The *cholesky* program performs a matrix decomposition using the numerical cholesky method [WOT+95]. As shown in Figure 4, the reduced cold and capacity miss rates mainly contribute to the better performance with shared level 2 caches. The conflict misses, on the other hand, decrease the performance of the shared level 2 caches on 4 and 8 processors.

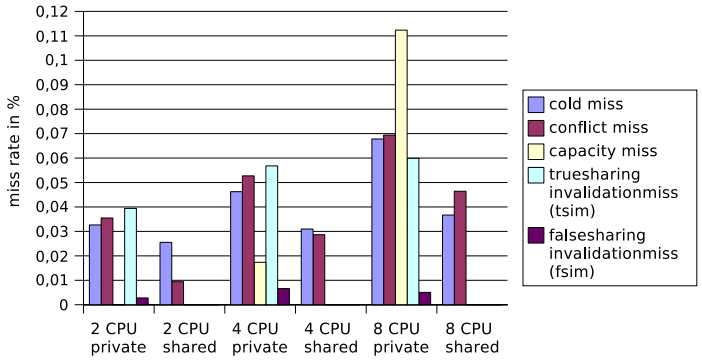


Figure 5: Water n^2 benchmark for shared and private level 2 caches.

Water-n^2
The *water-n*^{2} benchmark simulates a multi dimensional body [WOT+95]. As shown in

Figure 5, the private caches show disadvantageous behaviour concerning the number of coherence misses and the increasing capacity miss ratio, which rises from ∼0% to 0,11%. According to [WOT+95], the second working[1] set exceeds the capacity of the private caches. As the private caches become smaller the more processors are used, while the data set size stays the same, the second working set does not fit in the caches, resulting in capacity misses. In the shared cases the second working set does not exceed the cache capacity, as capacity misses do not occur. Additionally, cold and conflict misses are also decreased with the shared cache.

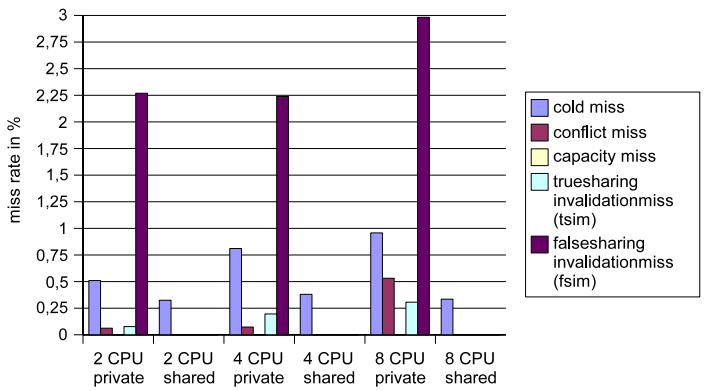


Figure 6: LU with non continuous blocks benchmark for shared and private level 2 caches.

LU with non continuous blocks

LU with non continuous blocks performs a matrix decomposition using the LU factorisation. As shown in Figure 6 the only perceived misses using a shared second level cache are cold misses with a miss rate of around 0,35%. The private case is dominated by coherence misses, precisely false sharing invalidation misses, that yield rates between 2,27% and 2,98%. The true sharing invalidation miss rate is between 0,08% and 0,32%. Thus, a wrong cache line size is indicated. Compiler techniques, padding data to full cache line sizes, are indicated to prevent false sharing invalidation misses. Capacity misses are not detected, which is due to the small benchmark working set size.

Improved Cache Configuration

The analysis of the *cholesky* and the *water-n*^{2} benchmarks reveals conflict misses. In order to reduce the conflict misses and improve the performance, we increased the level 2 cache associativity from 4-way to 8-way set-associative and repeated the simulation. The results of the *water-n*^{2} program are shown on the left hand side of Figure 7 and the *cholesky* program on the right hand side. The cache miss rate of the *water-n*^{2} program improves by at least 5.3% (4 CPUs with private caches) whereas the *cholesky* program improves by at least 1.5% (8 CPUs with private caches) compared to the 4-way set-associative level 2 caches. The increased associativity has a greater effect on the shared level 2 caches as the

[1] working set 2 corresponds to the second knee of the function in cache size and miss rate

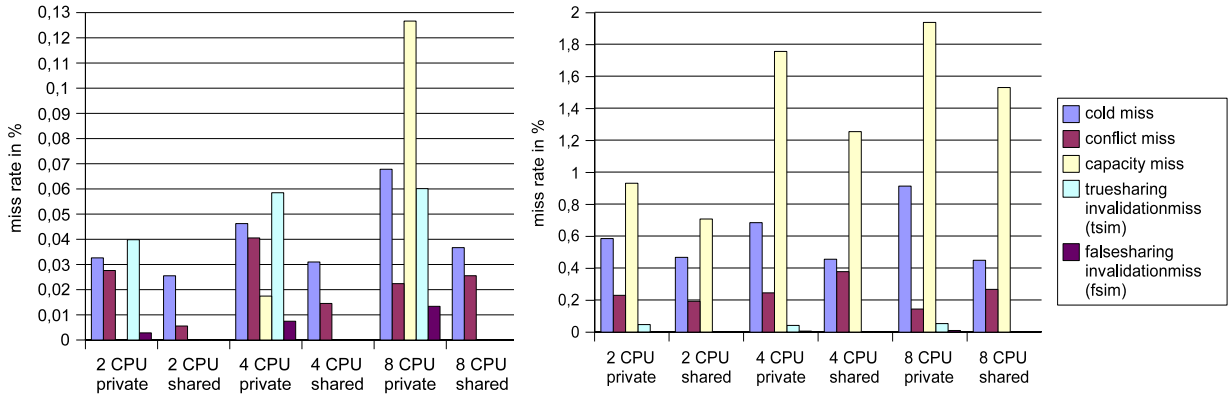


Figure 7: Water n^2 (left) and cholesky (right) with 8-way set-associative level 2 caches.

conflict misses make up a larger fraction of the overall miss rate. Therefore, the miss rate benefits more from the decreased number of conflict misses.

Summary
Overall, the shared architecture generally benefits from less cold and capacity misses. The former can be explained by the fact that shared data only causes one cold miss with the processor first accessing it. For the latter a larger cache is available for the working set. In addition, shared caches have no coherence misses. Further, the analysis tool is shown to be useful. As expected, increasing the cache associativity causes the number of conflict misses to decrease.

4.2 Visualisation

The analysis results can also be applied to understand the cache and program access pattern and further achieve an optimised application. For this, YACO is used for representing the results. YACO [QTK05] is a cache visualisation tool specifically designed for cache optimisation. It uses a set of several graphical views to guide the user to detect the problem, the reason, and the solution.

Figure 8 is a sample view used to highlight cache critical variables, i. e. the access bottlenecks. The simplicity of the graphical representation helps the programmer to clearly identify the bottlenecks throughout program execution. The relation between the name and the miss rate points the programmer to the variables worth optimising. The *fft* program shows that except **umain** all other main data structures have to be optimised. In the next step, programmers can use YACO's data access and cache views to analyse the access pattern and further to detect the optimisation strategies. Optimisation examples are found in [QTK05].

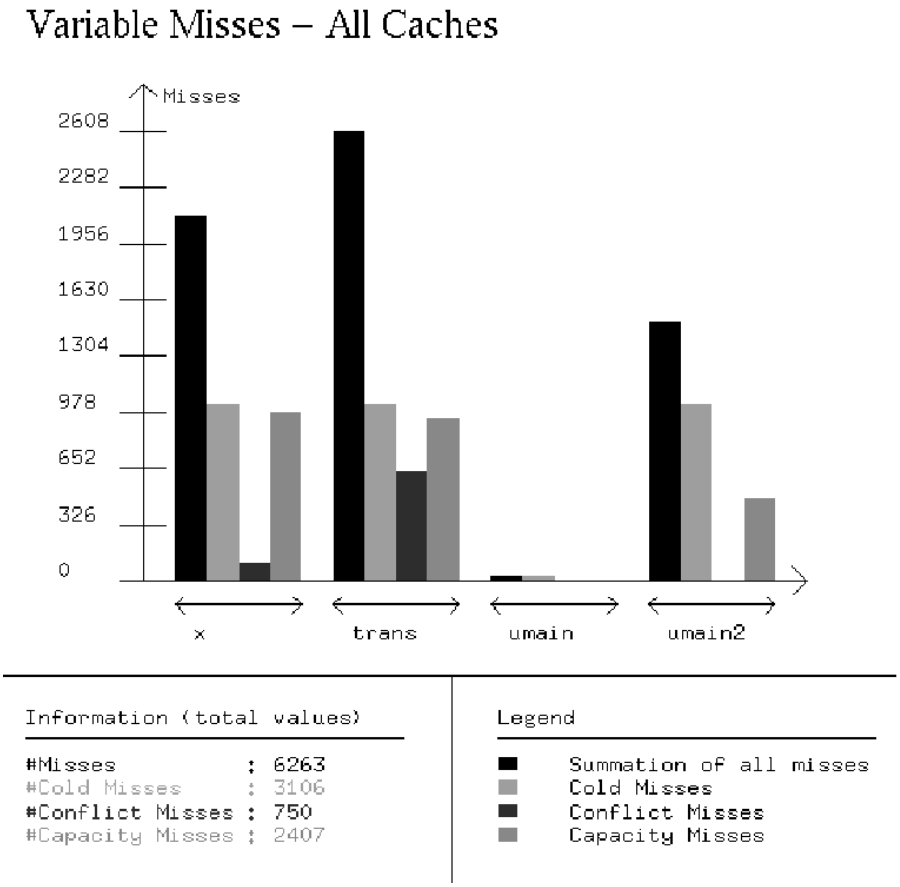


Figure 8: YACO's misses per variable.

5 Conclusion

This work uses an analysis approach to investigate the feature of cache misses on multiprocessor machines. The g-cache module of SIMICS is used to create a cache event trace. A miss classification model is applied to the cache event trace in order to distinguish cold, conflict, capacity, and sharing invalidation misses. A component for generating traces of performance facts, which can be delivered to an existing visualisation tool for graphical presentation of cache bottlenecks, is implemented as well. The following results are achieved. For all considered benchmarks the shared level 2 cache is the better choice as it improves the cache miss rate. The overall advantage of the shared level 2 cache is the lack of coherence misses.

The best example for an improved cache miss rate by eliminating coherence misses is the *lu with non continuous blocks* benchmark. The coherence misses reveal the false sharing of cache lines.

Other programs as the *water-n*^{2} yield better miss rates in the shared case, because the working set size exceeds the private caches resulting in a higher capacity miss rate.

The results obtained from the *cholesky* program indicate that the most benefit is drawn from an increased cache size, as the coherence miss rates are negligible. In the shared

cases conflict misses are visible. As shown in section 4.1.2 a higher cache associativity reduces the conflict misses and increases the performance.

Section 4.2 and section 4.1.2 show how the developed tool can guide the user to an improved adjustment of application and cache.

References

[Bar] Ken Barr. http://kbarr.net/splash2.html. Online; accessed January 2, 2008.

[BD01] K. Beyls and E. D'Hollander. Reuse distance as a metric for cache behavior. In *PDCS '01: Proceedings of the Conference on Parallel and Distributed Computing and Systems*, pages 617–662, August 2001.

[DSR+93] M. Dubois, J. Skeppstedt, L. Ricciulli, K. Ramamurthy, and P. Stenström. The Detection and Elimination of Useless Misses in Multiprocessors. In *Proceedings of the 20th International Symposium on Computer Architecture*, San Diego, CA, 1993.

[Hei] Wim Heirman. http://trappist.elis.ugent.be/~wheirman/simics/splash2/. Online; accessed January 2, 2008.

[HS89] M. D. Hill and A. J. Smith. Evaluating Associativity in CPU Caches. *IEEE Trans. Comput.*, 38(12):1612–1630, 1989.

[JE95] Tor E. Jeremiassen and Susan J. Eggers. Reducing false sharing on shared memory multiprocessors through compile time data transformations. In *PPOPP '95: Proceedings of the fifth ACM SIGPLAN symposium on Principles and practice of parallel programming*, pages 179–188, New York, NY, USA, 1995. ACM Press.

[QTK05] B. Quaing, J. Tao, and W. Karl. YACO: A User Conducted Visualization Tool for Supporting Cache Optimization. In *HPCC '05: High Performance Computing and Communications: First International Conference*, volume 3726 of *Lecture Notes in Computer Science*, pages 694–703. Springer, September 2005.

[Rev06] Virtutech AB, Nortullsgatan 15, SE-113 27 STOCKHOLM, Sweden. *Simics User Guide for Unix*, February 2006. Simics Version 3.0.

[spl] SPLASH-2: Stanford Parallel Applications for Shared Memory. http://www-flash.stanford.edu/apps/SPLASH/. Online; accessed January 2, 2008.

[Sto] Bastiaan Stougie. http://kbarr.net/files/splash2/pthread.m4.stougie. Online; accessed January 2, 2008.

[Sto03] Bastiaan Stougie. Optimization of a Data Race Detector. Master's thesis, Delft University of Technology, October 2003.

[WOT+95] Steven Cameron Woo, Moriyoshi Ohara, Evan Torrie, Jaswinder Pal Singh, and Anoop Gupta. The SPLASH-2 programs: characterization and methodological considerations. In *ISCA '95: Proceedings of the 22nd annual international symposium on Computer architecture*, pages 24–36, New York, NY, USA, 1995. ACM Press.

Parallel derivative computation using ADOL-C

Andreas Kowarz, Andrea Walther

{Andreas.Kowarz, Andrea.Walther}@tu-dresden.de
Institute of Scientific Computing, Technische Universität Dresden
01062 Dresden, Germany

Abstract: Derivative computation using Automatic Differentiation (AD) is often considered to operate purely serial. Performing the differentiation task in parallel may require the applied AD-tool to extract parallelization information from the user function, transform it, and apply this new strategy in the differentiation process. Furthermore, when using the reverse mode of AD, it must be ensured that no data races are introduced due to the reversed data access scheme. Considering an operator overloading based AD-tool, an additional challenge is to be met: Parallelization statements are typically not recognized. In this paper, we present and discuss the parallelization approach that we have integrated into ADOL-C, an operator overloading based AD-tool for the differentiation of C/C++ programs. The advantages of the approach are clarified by means of the parallel differentiation of a function that handles the time evolution of a 1D-quantum plasma.

1 Introduction

Automatic differentiation (AD) is a technique that has been developed and improved in the last decades. It allows to compute numerical derivative values within machine accuracy for a given function of basically unlimited complexity. Thus, unlike finite differences, no truncation errors must be taken into account. When calculating first order derivatives using the *forward* mode of AD, i.e., determining Jacobian-vector products, the computational effort is comparable to that of finite differences. This way, e.g., columns of the Jacobian can be computed efficiently. However, if a vector-Jacobian product is to be computed, e.g., a specific row of the Jacobian, the computational effort is proportional to the number of entries in the row when applying finite differences. The same task can be performed much more efficiently by use of the *reverse* mode of automatic differentiation. In particular, the computational effort is then independent of the rows dimension. A comprehensive introduction to AD can be found in [Gri00].

To provide reverse mode differentiation, AD tools based on operator overloading need to create an internal representation of the function $F : I\!R^n \rightarrow I\!R^m,\ y = F(x)$, to be differentiated, where $x = XI$ denotes the set of independent variables and $y = YD$ the set of dependent variables. The internal representation can be based on graphs [BS96] or sequential tapes. ADOL-C is an operator overloading based tool that provides automatic differentiation for functions given as C/C++ source code [GJU96]. The internal repre-

sentation of the considered function is created using a taping mechanism and a so-called *augmented* data type `adouble` that replaces `double` variables. Such a mechanism can be found in many AD-tools offering reverse mode differentiation based on operator overloading. Essentially, only the function value is computed within the overloaded operator or intrinsic function, respectively. In addition, information exactly describing the operation is recorded onto a tape. This information comprises the type of the operation/function, e.g. MULT, SIN, etc., as well as representations of the involved result and arguments, represented by so-called *locations*. After the evaluation of the function, the created tape represents the computational graph of the function as the sequence of operations that have been processed, in execution order. Based on this information, the program flow sequence can be easily inverted by interpreting the tape in reverse order.

Taking into account the steadily increasing demand for parallel program execution, an approach has to be found that allows to utilize the parallelization strategy of the provided user function for parallelizing the derivative computation. When ADOL-C is to be applied in a parallel environment that is created using OpenMP, several challenges have to be met. Firstly, the created tapes do not contain any information describing the parallel evaluation of the considered function. This is due to the fact that OpenMP statements cannot be overloaded. Furthermore, if all threads of the parallel environment would write onto the same tape, the serialization of the program flow would be inevitable, and, moreover, write conflicts would be very likely. Secondly, when computing derivative information applying the reverse mode of AD, the data access behavior is also reversed, i.e., read accessed function variables turn into write accessed derivative variables and vice versa. This potentially results in data access races.

In this paper, we present parallelization strategies that have been incorporated into the AD-tool ADOL-C. We mainly concentrate on the parallel reverse mode of AD but also give information on the new features that allow a parallel tape based forward mode. In the following section, a short overview of the facilities for generating a parallel AD-environment inside ADOL-C is given. Special issues for parallel reverse mode differentiation are discussed in Section 3 whereas Section 4 is dedicated to a numerical example which demonstrates the runtime advantages that can be achieved by applying the parallel reverse mode. A short summary and an outlook complete this paper.

2 Extensions to the ADOL-C

ADOL-C has been developed over a long period of time under strict sequential aspects. Although the generated tapes have been used for a more detailed analysis and the construction of parallel derivative code, e.g., [Bis91], ADOL-C could hardly be applied out of the box within a parallel environment, so far. The most convenient chance in this context is given by the use of ADOL-C in a message passing system based on distinct processes for all cooperators. This requirement is fulfilled by, e.g., MPI [HW99]. Due to separated address space and the realization of cooperators as processes of the operating system, the ADOL-C environment is multiplied. In particular, all control variables used in ADOL-C are available within each process exclusively. From the users point of view, only two

conditions must be met to allow a successful application of ADOL-C. First, the uniqueness of the created tape name, also called tag, must be ensured. This can be achieved by carefully choosing the tag, e.g., in dependence of the process' rank. Second, it must be considered that data transfer between the working processes is not reflected by the internal representation created by ADOL-C. Then, ADOL-C may be applied as usual allowing the computation of derivatives for functions that implement data partitioning techniques, especially when only a limited degree of communication is necessary.

Many parallel applications rely on a high amount of synchronization to communicate computed information at given points among involved cooperators. In a message passing environment this would also mean to invoke more or less expensive transfer routines. Therefore, such applications are typically parallelized for a shared memory environment using OpenMP [DM98]. Extensive enhancements have been added to ADOL-C to allow the application in such cases. Originally, the location of an augmented variable is assigned during its construction utilizing a specific counter. Creating several variables in parallel results in the possibility to loose the correctness of the computed results due to a data race in this counter. Initial tests based on the protection of the creation process by use of critical sections showed unambiguous behavior. Even when using only two threads in the parallel program, the runtime increased by a factor of roughly two rather than being decreased. For this reason, a separate copy of the complete ADOL-C environment is provided for every worker thread. The copy mechanism can be implemented using two different ways, either based on the OpenMP `threadprivate` clause, or by utilizing the thread number. Possible effects of the chosen strategy are discussed in Section 4.

Besides this decision, another issue had to be answered. As already identified by G. M. Amdahl [Amd67], every parallel program possesses a certain fraction that can only be handled serially. In many situations not only the parallel part of the function is object to the derivation efforts but also the serial parts. This entails the question of how to transfer information between the serial and parallel program segments and vice versa.

From serial to parallel
: Data transfer is this direction can be performed quit easily. For all variables alive at the moment when the parallel region starts, a copy may be created for each thread.

From parallel to serial
: This is the more difficult direction as it requires to decide which values from which thread should be copied to the serial part. Furthermore, the handling of variables created within the parallel part must be solved.

For the current implementation, the following decisions have been made.

- The handling of parallel regions by ADOL-C comprises only augmented variables but not user variables of standard data type.
- Control structures utilized by ADOL-C are duplicated for each thread, and are default initialized during the first creation of a parallel region. The values of these control variables are then handed on from parallel region to parallel region.

- For performance reasons, two possibilities of handling the tape information have been implemented. In the first case, control information including the values of augmented variables are transferred from the serial to the parallel variables every time a parallel region is created. Otherwise, this process is invoked only during the creation of the first parallel region. In either case, the master thread creates a parallel copy of the variables for its own use.
- No variables are copied back from parallel to serial variables after completion of a parallel region. This means, results to be preserved must be transferred using variables of standard data type.
- The creation or destruction of a parallel region is not represented within the initiating tape. Coupling of serial and parallel tapes must therefore be arranged explicitly by using the construct of external differentiated functions, see [Kow08].
- Different tapes are used within serial and parallel regions. Tapes begun within a specific region, no matter if serial or parallel, may be continued within the following region of the same type.
- Tapes created during a serial region can only be evaluated within a serial region. Accordingly, tapes written during a parallel region must be evaluated there.
- Nested parallel regions are not supported and remain object to later enhancements.

All in all, the described facts result in augmented variables with a special property that depends on the specific handling of the tape information. With the start of a new parallel region either a *threadprivate* or a *firstprivate* behavior, respectively, is simulated, [DM98]. This means that the value of the augmented variable is taken either from the previous parallel region or from the serial region, respectively. In either case, the value used within the parallel region is invisible from within the serial region.

Initializing the OpenMP-parallel regions for ADOL-C is only a matter of adding a macro to the outermost OpenMP statement. Two versions of the macro are available, which are only different in the way the tape information is handled. Using `ADOLC_OPENMP`, this information including the values of the augmented variables is always transferred from the serial to the parallel region. In the other case, i.e., using `ADOLC_OPENMP_NC`, this transfer is performed only the first time a parallel region is entered. This reduces the copy overhead for iterative processes. Due to the inserted macro, the OpenMP statement has the following structure:

```
#pragma omp ...  ADOLC_OPENMP
                 or
#pragma omp ...  ADOLC_OPENMP_NC
```

Within a parallel region, different tapes are created by the threads. Succeeding the taping phase, derivatives are computed using the various tapes. This can be done either by complete user steering, or semi-automatic by applying the concept of extern differentiated functions.

While forward mode differentiation is, in a way, straightforward, the computation of derivatives utilizing the reverse mode of AD needs special attention.

3 Reverse mode for data parallelism

For parallel reverse mode differentiation, we focus on functions that feature a special type of data parallelism. Basically, data parallelism describes the subdivision of the data domain of a given problem into several regions. These regions are then assigned to a given number of processing elements, which apply the same tasks to each of them. Data parallelism is commonly exploited in many scientific and industrial applications and exhibits a "natural" form of scalability. Since the problem size for such applications is normally expressed by the size of the input data to be processed, an upscaled problem can typically be solved using a correspondingly higher number of processing elements at only a modestly higher runtime [DFF+03].

For our purpose, data parallelism is extended beyond the pure nature described above. Accordingly, it shall be allowed that the complete set of independent variables XI of the given function F may be used by all p processing elements PE_i, $i = 1, \ldots, p$, for reading. Denoting by $XI_r^{(i)}$ and $XI_w^{(i)}$ the read and write accessed subsets of XI, respectively, for the various processing elements, one has that

$$\forall PE_i : \quad XI_r^{(i)} \subseteq XI, \qquad XI_w^{(i)} = \emptyset. \tag{1}$$

This allows for example parallel functions that handle the evolution of systems consisting of many components. There, computations for the individual components can be performed independently, provided that the interaction among them can be determined using XI. Derivative information for such applications can be provided using the scheme depicted in Figure 1. In contrast to the general read access in terms of the independents,

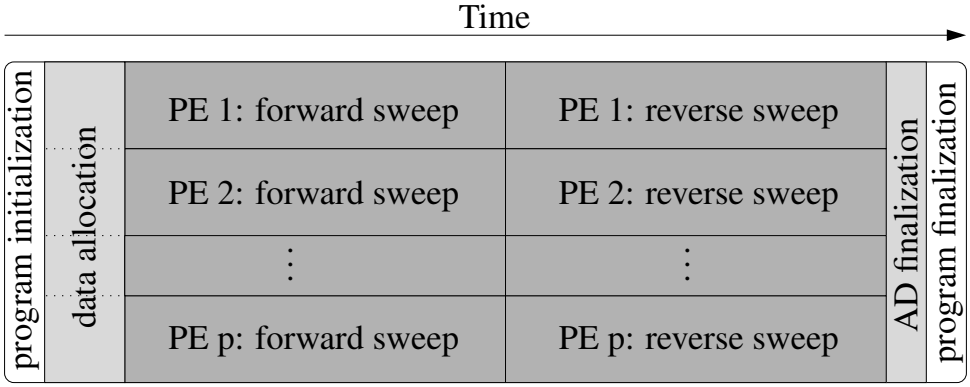


Figure 1: Basic layout of data-parallel calculations in an AD-environment

write access may only be allowed for distinct subsets $YD^{(i)}$ of the dependent variables. For a given number p of processing elements it is required that

$$YD = \bigcup_{i=1}^{p} YD^{(i)} \qquad \text{and} \qquad YD^{(i)} \cap YD^{(j)} = \emptyset \qquad \text{with} \quad i, j \in [0, p],\ i \neq j.$$

The set of intermediate variables $IV^{(s)}$ associated with the serial function evaluation is considered to be reproduced for all processing elements yielding p sets of variables $IV^{(i)}$, $i = 1, \ldots, p$. In this way, intermediate values can be used at the certain processing element without the potential of memory conflicts. Overall, considering the subset $YD^{(i)}$ and the set of intermediate variables $IV^{(i)}$, which are used exclusively by the processing elements PE_i, it holds that

$$\forall PE_i \, \forall PE_j : \begin{cases} YD^{(i)} \cap YD^{(j)} = \emptyset & IV^{(i)} \cap IV^{(j)} = \emptyset \quad \text{for} \quad i \neq j \\ YD^{(i)} = YD^{(j)} & IV^{(i)} = IV^{(j)} \quad \text{for} \quad i = j \end{cases} . \quad (2)$$

Any function exhibiting the properties (1) and (2) is considered correctly parallelized in the sense that data races are debarred.

Obviously, equation (2) allows to compute derivatives for the given function as long as only the sets $IV^{(i)}$ and $YD^{(i)}$ are involved. Due to the distinct nature of the sets defined for the processing elements, forward and reverse mode of AD can be applied safely. A less obvious situation is given as soon as independent variables are involved in the computation. As known from the theory of the reverse mode, read accessed function variables result in write accessed derivative variables. More precisely, the following relation holds

Function	Derivative (reverse mode)
$v_i = \varphi_i(v_j)_{j \prec i}$	$\bar{v}_j \mathrel{+}= \bar{v}_i * \frac{\partial \varphi_i(v_j)_{j \prec i}}{\partial v_j} \qquad \forall j \in \{j : j \prec i\}.$

Therein, the term $\varphi_i(v_j)_{j \prec i}$ denotes an elemental operation or intrinsic function to compute an intermediate value v_i, and $\prec$ refers to the dependency relation as defined in [Gri00]. As can be seen, due to the required instructions in the reverse mode, the data access layout of the function variables is reversed for the adjoint variables $\bar{v}_i$ and $\bar{v}_j$ associated with each intermediate variable v_i and v_j, respectively. Hence, read accesses on the independent variables x_k, $k = 1, \ldots, n$, induce the potential of data races in the adjoint computations.

However, similar to the handling of intermediate variables, different sets of adjoint variables $\overline{XI}^{(i)}$ can be provided for each processing element corresponding to the set XI. Adjoint values may then be updated locally by each processing element independently and thus globally in parallel. Due to the additive nature of the derivative computations, global adjoint values may later be assembled using the local information produced by the various processing elements. As this assembling results in significant computational effort, it should be executed for all relevant adjoints only once and in a single step. Thus, the assembling step must be performed after the last update of an adjoint variable $\bar{x}_k^{(i)} \in \overline{XI}^{(i)}$. However, updates of adjoints $\bar{x}_k^{(i)}$ are principally possible at any point of the reverse computations. This results from the property of the function that independent variables may be accessed at any given time of the function evaluation. Thus, a single adjoint assembling step is only possible if no $\bar{x}_k^{(i)}$ is used as an argument of a derivative instruction before all updates on the set $\overline{XI}^{(i)}$ have been performed. For the considered type of applications that feature the properties (1) and (2), this potential conflict can never occur. Since the individual independent variables are accessed for reading only, they cannot appear on the

right-hand side of an derivative instruction. Hence the assembling phase that computes the global derivative values can be safely moved to the end of the derivation process.

4 Numerical Example

The example that is used to demonstrate the parallel derivation of a given function is taken from physics. Due to its complex structure it is also necessary to apply in addition to the parallelization other techniques derived in [Kow08], in particular, nested taping, external differentiated functions, and checkpointing facilities. Only the combination of these techniques allows the derivation based on the reverse mode of AD for this example. The implementation of the function was performed by N. Gürtler [Gür06], and the differentiation was realized in a cooperation between the RWTH Aachen and the TU Dresden. To our knowledge, the parallel derivation of the given function including the coping with the high internal complexity and inherent challenges currently features uniqueness and establishes a new level of the application of operator overloading based AD.

The example describes the time propagation of a 1D-quantum plasma. Therein, the plasma particles can be represented by a N-particle wave function $\Psi(1,\ldots,N)$, and the system is modeled by multi-particle Schroedinger equations. For reduction of the complexity, spin effects are neglected. However, a direct solution is numerically highly expensive. Since an approximation is often sufficient to describe the physical behavior, the simulation is based on Quantum-Vlasov equations. Interchange and correlation effects are neglected. As an entry point into quantum plasma simulations and for reasons of complexity only the one-dimensional case is modeled. However, due to the necessary discretization in the order of N dimensions, the direct solution is still very expensive. For the analysis of expected values for many distributions, calculations based on a representative ensemble of quantum states are sufficient. Prior to the numerical simulation, a discretization of the resulting equation system is performed. Applying cyclic boundary conditions yields the sparse cyclic tridiagonal system (3).

$$U_i^{+,n+1}\Psi_i^{n+1} = U_i^{-,n}\Psi_i^n \tag{3}$$

For details on the discretization and the definition of the operator U, the reader is referred to [Gür06, Gür07]. With a system like (3), a complete description of the time propagation of the discretized wave function Ψ is given. Therein, the term Ψ_i^n denotes the wave function of the ith particle at time n.

The final step, which succeeds the time propagation of the plasma is used to compute the expected value $<\eta>$ of the particle density. The discrete version of this target functions is given by

$$<\eta> = \sum_{i=1}^{N}\sum_{j=1}^{K} z(j)\Delta z|\Psi_{i,j}|^2 \ . \tag{4}$$

The reduction of the high amount of output information resulting from the time propagation to a single value allows an easier evaluation of the entire system.

A code for simulating the time propagation of a one-dimensional ideal quantum plasma has been developed [Gür06]. It creates the source of the differentiation efforts and features the program layout that is depicted in Figure 2. There, all parallelization statements are al-

```
...
startup_calculation(..);
for (n = 0; n < T; ++n) {
   #pragma omp parallel
   {
      #pragma omp for
      for (i = 0; i < N; ++i)
         part1(..);
      #pragma omp for
      for (i = 0; i < N; ++i)
         part2(..);
   }
}
target_function(..);
...
```

Figure 2: Basic layout of the plasma code including parallelization statements

ready included. The representation is based on C++ notation, and the OpenMP statements are adjusted accordingly. Due to the layout of the overall function, the reverse mode of AD is to be preferred for the differentiation of the code.

For the proof of concept of the code as well as its derivation, we used the following parameters for all runtime measurements that are discussed in this section.

- number of wave functions $N = 24$, simulation time $t = 30$, $T = 40000$ time steps
- length of the simulation interval $L = 200$, discretized with $K = 10000$ steps
- plasma frequency $\omega_P = 1.23$

All units are transformed to the atomic scale, see [Gür06]. The most important runtime reduction in the simulation results from computing only the first 50 of the 40000 time steps. After this period, the correctness of the derivatives can already be validated and the characteristics of the runtime behavior that are of special interest are already fully visible. To preserve the numerical stability of the code, the time discretization is based on $T = 40000$ steps nevertheless. All runtime measurements have been performed using the *SGI ALTIX 4700* system installed at the TU Dresden.

Two versions of parallel derivative computations are discussed in the remainder of this section. They apply the same parallelization approach, i.e., every participating thread of

the parallel environment performs all calculations for a specific number of the considered N wave functions. The main difference is to be found in the way the data access in the parallel environment is performed. Figure 3 depicts the speedups measured for the derivation of 24 wave functions. As can be seen, the code version that is based on

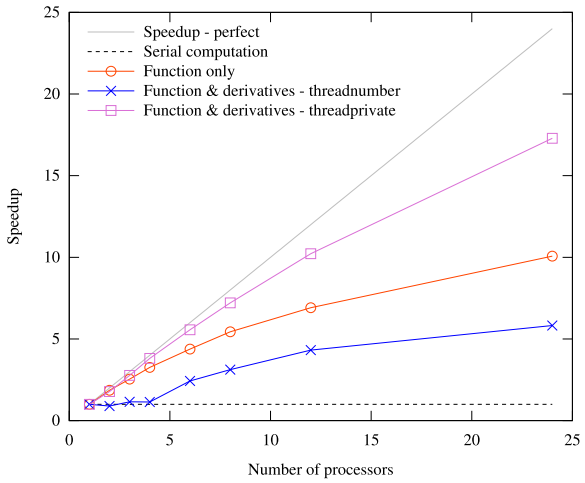


Figure 3: Speedups for the parallel differentiation of the plasma code for $N = 24$ wave functions

a `threadprivate` parallelized AD-environment achieves the highest speedups. Furthermore, it performs much better than thread number based reverse version, and it even outperforms the speedup of the original functions. This allows the conclusion that, carefully implemented, the derivative calculations can decrease the distracting effect of the necessary synchronization within the parallel environment.

The results attained through the parallelization of the differentiation of the plasma code clarify that operator overloading based AD is prepared to meet the challenges that are brought up by the most complex codes applied in science and engineering. Thus, for the parallelization of derivative calculations using operator overloading AD, an answer has been found to a question that is still open for many other applications.

5 Summary & outlook

Automatic differentiation based on operator overloading features a long history and has shown to be highly valuable for most derivation tasks. Especially for programming languages for which AD-enabled compilers are not available or miss a critical feature, the operator overloading based approach often presents the only reasonable technique. The increasing complexity of the investigated functions more and more requires the application of parallelization techniques. It is obvious that automatic differentiation must face this fact and provide adequate differentiation strategies. In this paper, we presented new parallelization approaches that have been incorporated into the tool ADOL-C. By means of the time propagation of a 1D quantum plasma we could show that parallel reverse mode differentiation can be performed efficiently using operator overloading based AD.

The main limitation of AD utilizing the overloading facilities of programming languages is always to be found in the size of created internal function representation. Within this context, the unrolling of loops presents a major drawback that may result in an unfavorable runtime behavior. Hence, one of the main challenges to be answered in the future is the development of techniques that allow a much more compact representation of loops. This not only avoids the expensive storing of every loop iteration, but also presents a major step towards an automatic parallel differentiation of parallelized user functions.

References

[Amd67] G. M. Amdahl. Validity of the single processor approach to achieving large scale computing capabilities. volume 30 of *AFIPS conference proceedings*, pages 483–485, National Press Building, Washington, D.C. 20004, USA, 1967. Thompson Book Co. Spring joint computer conference, Atlantic City.

[Bis91] C. H. Bischof. Issues in Parallel Automatic Differentiation. In A. Griewank and G. F. Corliss, editors, *Automatic Differentiation of Algorithms: Theory, Implementation, and Application*, pages 100–113. SIAM, Philadelphia, PA, 1991.

[BS96] C. Bendtsen and O. Stauning. FADBAD, a Flexible C++ Package for Automatic Differentiation. Technical Report IMM–REP–1996–17, Department of Mathematical Modelling, Technical University of Denmark, Lyngby, Denmark, 1996.

[DFF+03] J. Dongarra, I. Foster, G. Fox, W. Gropp, K. Kennedy, L. Torczon, and A. White, editors. *Sourcebook of Parallel Computing*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2003.

[DM98] L. Dagum and R. Menon. OpenMP: An Industry-Standard API for Shared-Memory Programming. *IEEE Computational Science and Engineering*, 05(1):46–55, 1998.

[GJU96] A. Griewank, D. Juedes, and J. Utke. Algorithm 755: ADOL-C: A Package for the Automatic Differentiation of Algorithms Written in C/C++. *ACM Transactions on Mathematical Software*, 22(2):131–167, 1996.

[Gri00] A. Griewank. *Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation*. Number 19 in Frontiers in Applied Mathematics. SIAM, Philadelphia, PA, 2000.

[Gür06] N. Gürtler. Simulation eines eindimensionalen idealen Quantenplasmas auf Parallelrechnern, 2006. Diploma thesis in physics, Rheinisch-Westfälische Technische Hochschule (RWTH) Aachen, Germany.

[Gür07] N. Gürtler. Parallel Automatic Differentiation of a Quantum Plasma Code, 2007. Diploma thesis in computer science, Rheinisch-Westfälische Technische Hochschule (RWTH) Aachen, Germany.

[HW99] R. Hempel and D. W. Walker. The Emergence of the MPI Message Passing Standard for Parallel Computing. *Computer Standards & Interfaces*, 21:51–62, 1999.

[Kow08] A. Kowarz. *Advanced Concepts of Automatic Differentiation based on Operator Overloading*. PhD thesis, TU Dresden, 2008. to appear.

How efficient are creatures with time-shuffled behaviors?

Patrick Ediger Rolf Hoffmann
Mathias Halbach

TU Darmstadt, FB Informatik, FG Rechnerarchitektur
Hochschulstraße 10, D-64289 Darmstadt
{ediger, hoffmann, halbach}@ra.informatik.tu-darmstadt.de

Abstract: The task of the creatures in the "creatures' exploration problem" is to visit all empty cells in an environment with a minimum number of steps. We have analyzed this multi agent problem with time-shuffled algorithms (behaviors) in the cellular automata model. Ten different "uniform" (non-time-shuffled) algorithms with good performance from former investigations were used alternating in time. We designed three time-shuffling types differing in the way how the algorithms are interweaved. New metrics were defined for such a multi agent system, like the absolute and relative efficiency. The efficiency relates the work of an agent system to the work of a reference system. A reference system is such a system that can solve the problem with the lowest number of creatures with uniform or time-shuffled algorithms. Some time-shuffled systems reached high efficiency rates, but the most efficient system was a uniform one with 32 creatures. Among the most efficient successful systems the uniform ones are dominant. Shuffling algorithms resulted in better success rates for one creature. But this is not always the case for more than one creature.

1 Introduction

The general goal of our project is to optimize the cooperative behavior of moving creatures in order to fulfill a certain global task in an artificial environment. A creature (another term: agent) behaves according to an algorithm which is stored in the creature.

We distinguish *uniform* and *time-shuffled* systems of creatures. A uniform system comprises creatures with one uniform behavior (algorithm) only whilst a time-shuffled system comprises creatures with generation-wise alternating behaviors. The goal of this investigation was to find out for the creatures' exploration problem (explained below), which algorithms "harmonize" best, meaning which combinations of algorithms with how many creatures are the most efficient. Different measures for efficiency were defined and used to compare the different systems. When we are speaking about efficiency you may think of cost (e. g., Euros) which you have to pay in total for the involved creatures to fulfill the task.

We are modeling the behavior by a finite state machine (Sec. 2). In the past we have tried to find out the best algorithm for one creature by enumeration. The number of state machines which can be coded using a state table is $M = (\#s\#y)^{(\#s\#x)}$ where $n = \#s$

is the number of states, $\#x$ is the number of different input states and $\#y$ is the number of different output actions. Note that M increases dramatically, especially with $\#s$, which makes it very difficult or even impossible to check the quality of all algorithms by enumeration in reasonable time. By hardware support (FPGA technology) we were able to simulate and evaluate all 12^{12} 6-state algorithms (including algorithms with less than 6 states and including redundant ones) for a test set of 5 initial configurations [HHB06]. The 10 best algorithms (with respect to percentage of visited cells) were used in further investigations to evaluate the robustness (using additional 21 environments) and the efficiency of $k > 1$ creatures. It turned out that more than one creature may solve the problem with less cost than a single one [HH07]. In our investigation we have concentrated on time-shuffled systems using the previously found algorithms. This time we are using 16 new environments compared to the environments used before. Now we use a field of fixed size 35×35 with a fixed number of obstacles (129). Thereby we are able to place the creatures at the beginning in regular formations and the number of obstacles becomes a constant which simplifies the analysis.

We have already started experiments using metaheuristics to optimize the algorithms. We are optimistic to find agent algorithms for more complex agent problems and we will use a cluster of FPGAs for that purpose to exploit the inherent parallelism in the metaheuristics. Our current results also give a partial answer to the question: If algorithms are combined, what are the expectation rates for good or bad combinations of them?

Modeling the behavior with a state machine with a restricted number of states and evaluation by enumerations was also undertaken in SOS [MSPPU02]. Additional work was done by these authors using genetic algorithms. The creatures' exploration problem based on our model was further investigated in [DL06]. Randomness was added which led to a higher degree of success. Our research in general is related to works like: Evolving optimal rules for cellular automata (CA) [Sip97, ST99], finding out the center of gravity by marching pixels [FKSL07, FS05, KMF07], evolving hardware [US06], using evolutionary algorithms and metaheuristics [Alb05].

The remainder of this paper is organized as follows. Sec. 2 describes how the problem is modeled in the CA model. Sec. 3 describes how well only uniform creatures under varying the environment solve the problem. Sec. 4 describes how efficient creatures with time-shuffled behavior can solve the problem.

2 CA model for the creatures' exploration problem

The problem is the following: p creatures are moving around in an environment that consists of empty cells and obstacle cells in order to visit all reachable empty cells in the shortest time. Creatures cannot move on obstacle cells, and only one creature can be located on an empty cell at the same time. Creatures can look forward one cell ahead which is in its moving direction. The creatures may perform four different actions: R (turn right only), L (turn left only), Rm (move forward and simultaneously turn right), Lm (move forward and simultaneously turn left).

If the "front cell" (the cell ahead) is not free, because it is an obstacle cell, another creature stands on it, or a collision conflict is anticipated, the action R or L is performed. In all other cases the action Lm or Rm is performed (Fig. 1). The detection of anticipated conflicts is realized by an arbitration signal from the destination cell. Each creature sends a request to its front cell, which sends back a grant signal if only one creature has sent a request [HHB06].

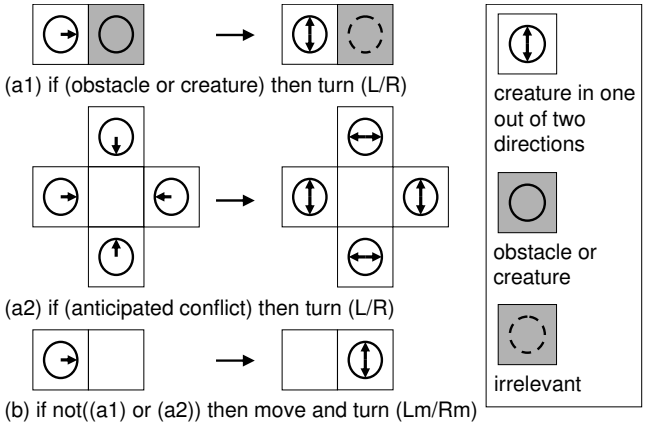


Figure 1: The conditions for the moving creatures' rules

The modeling of the behavior was done by implementing the rules with a state machine considered as a Mealy automaton with inputs (m, s), next state s' and output d (Fig. 2). An algorithm is defined by the contents of a state table assigned to the state machine. We are coding an algorithm into a string representation or a simplified string representation by concatenating the contents line by line to a string or a corresponding number, e. g.,

	1L2L0L4R5R3R-3Lm1Rm5Lm0Rm4Lm2Rm	*string representation*
=	1L2L0L4R5R3R-3L1R5L0R4L2R	*simplified string representation*

The state table can be represented more clearly as a state graph (Fig. 2). If the state machine uses n states, we call such an algorithm n-state algorithm. If the automaton is considered as a Moore automaton instead of a Mealy automaton, the number of states will be the product $n \times \#r$, where $\#r$ is the number of possible directions (4 in our case).

3 Uniform systems with one creature

In preceding investigations [HHB06] we could discover and evaluate the best 6-state algorithms for one creature by the aid of special hardware. The behavior of all relevant algorithms was simulated and evaluated for 26 initial test configurations (they are different from the ones we are using here). The following 10 best algorithms were ranked using a dominance relation with the criteria (1.) success, (2.) coverage and (3.) speed:

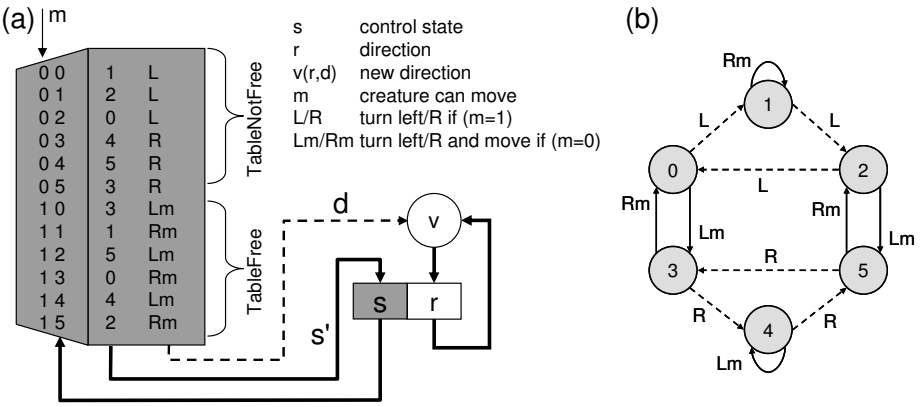


Figure 2: A state machine (a) models a creature's behavior. Corresponding 6-state algorithm (b)

1. G: 1L2L0L4R5R3R-3L1R5L0R4L2R
2. B: 1R2R0R4L5L3L-3R1L5R0L4R2L
3. C: 1R2R0R4L5L3L-3R4R2L0L1L5R
4. A: 0R2R3R4L5L1L-1R5R4R0L2L3L
5. D: 1R2R3R1L5L1L-1R0L2L4R3L1L
6. E: 1R2L0R4L5L3L-3R4R5R0L1L2R
7. F: 1R2L0L4R5R3R-3L4L5L0R1L2R
8. H: 1L2L3R4L2R0L-2L4L0R3L5L4R
9. I: 1L2L3L4L2R0L-2L4L0R3R5L4R
10. J: 1R2R3R0R4L5L-4R5R3L2L0L1L

The following definitions and metrics are used:

- k := number of creatures
- R := number of empty cells
- g := generation (time steps)
- $r(g)$: = number of visited cells in generation g
- $r_{\max}$:= the maximum number of cells which can be visited for $g \rightarrow \infty$
- $g_{\max}$:= the first generation in which $r_{\max}$ is achieved
- $e := r_{\max}/R[\%]$, the coverage or exploration rate, i. e. $\frac{\text{visited cells}}{\text{all empty cells}}$
- *speed* := $R/g_{\max}$ (only defined for successful algorithms)
- *step rate* := $1/$*speed* (the number of cells visited in one generation)

In order to find time-shuffled algorithms that are robust against changes of the environments, we have created a set of $I = 16$ environments which all contain 129 obstacle cells (Fig. 3). Then the algorithms A to J were simulated on them with one creature. The results show that none of the algorithms is capable of solving every environment successfully (Tab. 1). The best algorithms with respect to these environments are the algorithms B, G and J. Algorithm B has a *mean speed* (for $k = 1$ creature, see definition in Sec. 4) of 16.12% for the successful environments. The highest reachable mean speed would be 100% (visiting one new empty cell in each step). The *mean step rate*, which is the reciprocal of the mean speed, is 5.31 (every 5.31 steps an empty cell is visited). We can interpret the mean step rate as work or cost which has to be paid for a creature to visit a cell (e. g., 5.31€ per empty cell to visit, or to "clean" one "square meter", if you think of cleaning a room). Fig. 4 shows an example how algorithms may behave in principle. Algorithm B is successful for the environments 6, 10 and 16 and the speed is comparable (around 960 / 5400). Algorithm J is not successful for the environments 6 and 10.

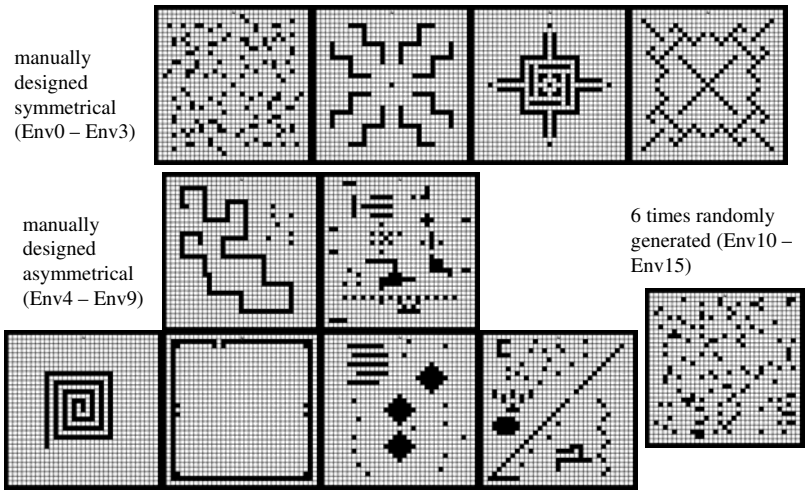


Figure 3: The 16 environments with 35×35 cells, manually designed or randomly generated. Each environment comprises $R = 960$ empty cells and 129 obstacles.

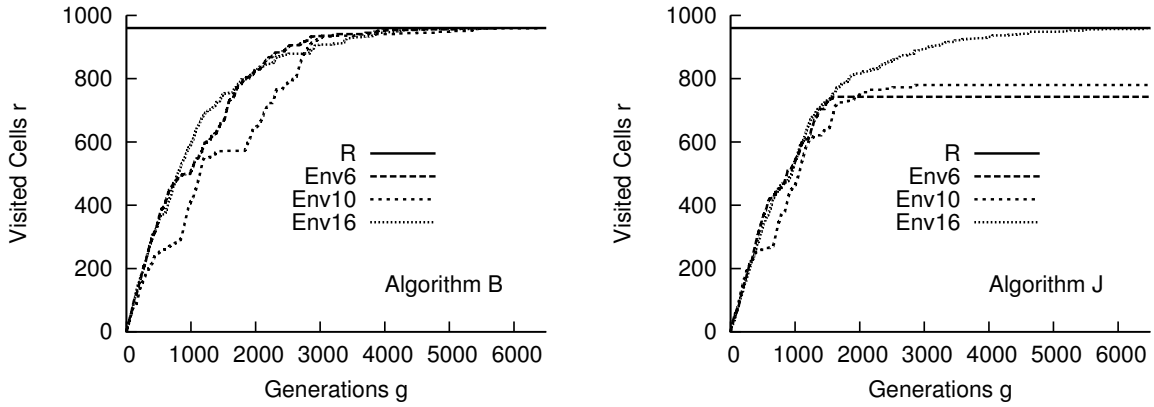


Figure 4: Curves showing the number of visited cells $r(g)$ (generations g on the x-axis, visited cells r on the y-axis) for the algorithms B and J for the environments Env6, Env10 and Env16. $R = 960$.

4 Multi creature and time-shuffled systems

In [HH07, HH06] we have evaluated that increasing the number of creatures can lead to synergy effects, i. e., the uniform creatures can work more efficiently together than by their own. 1 to 64 creatures were arranged on the cellular field symmetrically side by side at the borders (Fig. 5). The same distribution was used for the following investigation.

In order to investigate the performance of *time-shuffled systems* (multi agent system with time-shuffled algorithms), we have simulated all pairs (X, Y) of the former best single algorithms (A to J) on all 16 environments (Fig. 3) whereas the uniform pairs (X, X) are included for comparison. We used three different modes of time-shuffling the different algorithms (Fig. 6):

- *common* (c), each creature works with both algorithms alternating them generation-

Algorithm	Success Env0	Success Env1	Success Env2	Success Env3	Success Env4	Success Env5	Success Env6	Success Env7	Success Env8	Success Env9	Success Env10	Success Env11	Success Env12	Success Env13	Success Env14	Success Env15	T = No. Successful	total visiting percentage	mean g_max (successful)	mean r_max	mean speed (successful)	mean step rate
B	O	O	O	O	X	O	X	X	X	O	O	X	O	X	X	O	7	88,74%	5956	852	16,12%	5,31
G	O	O	O	O	X	O	X	X	X	O	O	X	O	X	X	O	7	87,32%	5998	838	16,01%	5,10
J	O	O	O	O	O	X	O	X	O	X	X	O	X	X	X	O	7	85,89%	6813	825	14,09%	5,78
C	X	O	O	O	X	O	X	X	O	O	O	O	O	X	X	O	6	86,92%	5626	834	17,06%	4,86
E	O	O	O	O	O	X	O	X	O	X	O	O	O	X	O	O	4	83,42%	6664	801	14,41%	5,20
A	O	O	X	X	O	O	O	O	O	O	O	O	O	O	O	X	3	82,38%	14297	791	6,71%	8,32
F	O	O	O	O	O	O	O	O	O	X	O	O	O	X	O	O	2	73,08%	6405	702	14,99%	4,60
D	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	0	47,55%	-	456	-	2,96
H	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	0	36,20%	-	348	-	4,69
I	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	0	36,20%	-	348	-	4,75

Table 1: Results of the simulation of one creature. The values are averaged over all environments, respectively over the T successfully visited environments in the case of "mean $g_{\max}$" and "mean *speed*". An *X* in the columns "Success Env$n$" indicates that the environment was successfully solved, an *O* that it was not.

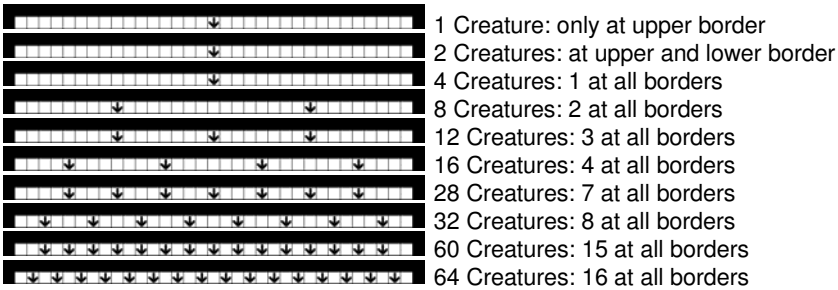


Figure 5: The arrangement of creatures in a multi creature system.

wise (odd/even). But it has only one common state s which is shared by the algorithms.

- *simultaneous* (s), each creature works with both algorithms simultaneously. Each of the algorithm has its own state (s_x, s_y). The output is taken alternating from X/Y for $t = 0/1$. Both states are updated in each generation.
- *alternate* (a), an individual state is used for each algorithm. For $t = 0$ the output is taken from X and only the state s_x is updated. For $t = 1$ the output is taken from Y and only the state s_y is updated.

The shuffling modes c and s are identical for all "pairs" (X, X). In that case they are in fact uniform systems.

One creature time-shuffled. The results (Tab. 2) show that one creature with time-shuffling can solve more environments successfully than the uniform systems. With the alternate shuffling up to 12 environments can be solved (B, J), 11 with the simultaneous variant (A, G) and the best pair with the common time-shuffling is (G, F).

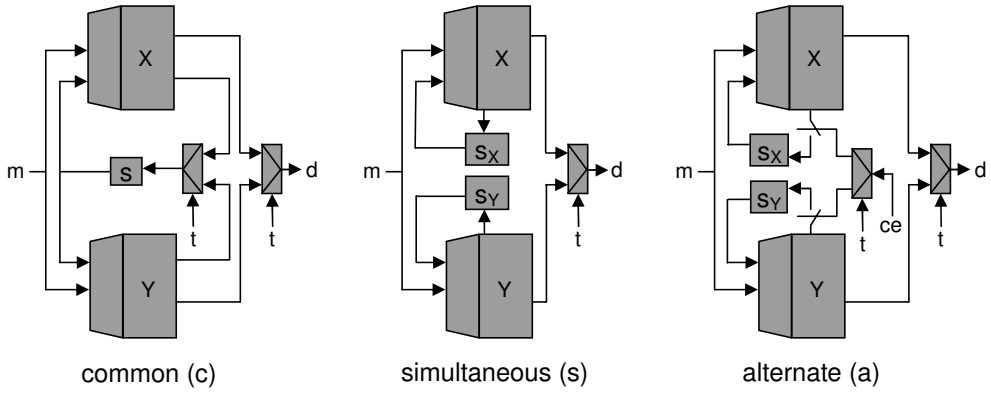


Figure 6: Types of time-shuffling of creature state machines. X is the first algorithm, Y the second. $t = 0/1$ is a function of the generation counter and controls the outputs and in the cases c and a also the state transition. ce (clock enable) is a signal that enables the state transition.

Algorithm X	Algorithm Y	Shuffling	S1	S2	S3	S4	S6	S7	S8	S9	S10	S11	S12	S13	S14	S15	S16	S17	T = No. Successful	total visiting percentage	mean speed (successful)
G	F	c	X	O	O	O	X	O	O	O	X	O	X	O	X	X	X	X	8	85,44%	9,51%
E	C	c	X	O	O	O	X	O	O	O	X	O	X	X	X	X	O	X	8	82,50%	9,61%
C	B	c	O	O	O	O	X	O	X	O	X	X	X	O	X	O	X	X	8	71,50%	11,54%
A	**G**	**s**	**O**	**X**	**O**	**X**	**X**	**X**	**X**	**O**	**X**	**O**	**X**	**O**	**X**	**X**	**X**	**X**	**11**	**96,33%**	**3,82%**
J	B	s	O	X	O	O	X	O	X	O	O	X	X	O	X	X	X	X	9	92,42%	5,07%
B	J	s	O	X	O	O	X	O	X	O	X	X	X	O	X	X	O	X	9	85,61%	6,01%
B	**J**	**a**	**X**	**X**	**X**	**O**	**X**	**O**	**X**	**O**	**X**	**O**	**X**	**X**	**X**	**X**	**X**	**X**	**12**	**85,84%**	**5,79%**
B	**A**	**a**	**X**	**O**	**O**	**X**	**X**	**O**	**X**	**O**	**X**	**X**	**X**	**X**	**X**	**O**	**X**	**X**	**11**	**99,38%**	**4,48%**
C	A	a	X	X	O	X	X	O	X	X	X	O	X	X	X	O	X	O	11	95,24%	4,90%

Table 2: The three pairs of algorithms for each kind of shuffling that have the most successes (1.) and the highest total visiting percentage (2.) with one creature. The overall most successful algorithm pairs are shown in bold letters.

In the following we will denote a system in the way *XYp-k*, where *XY* is the pair of algorithms, k the number of agents and p the mode of time-shuffling (optional). E. g., ABa-8 is a system with 8 creatures using an alternate time-shuffling of the algorithms (A,B).

Successful systems, depending on the number of creatures. By increasing the number of creatures the success rate also increases (Tab. 3). At least 8 creatures are necessary to visit successfully all environments. The systems J-8 and nine other time-shuffled systems with 8 creatures are successful. With 64 creatures 9 out of 10 uniform systems (B-64 to J-64) and 58 out of 280 time-shuffled systems are successful. In total 41/100 uniform and 185/2800 time-shuffled systems are successful for all 16 environments.

Metrics for time-shuffled systems with more than one creature. To be able to evaluate and compare time-shuffled systems (including uniform systems) with a different number of creatures we have defined additional metrics:

NO. OF CREATURES	SUCCESSFUL (UNIFORM ONLY)	NO. OF SUCCESSFUL SYSTEMS / NO. OF COMBINATIONS (TIME-SHUFFLED)			
		c+s+a	common (c)	simultaneous (s)	alternate (a)
1	-	0/280	0/90	0/90	0/100
2	-	0/280	0/90	0/90	0/100
4	-	0/280	0/90	0/90	0/100
8	J	9/280	1/90	3/90	5/100
12	B,C,J	9/280	1/90	5/90	3/100
16	B,C,E,G,J	16/280	2/90	7/90	7/100
28	B,C,D,E,I,J	25/280	6/90	8/90	11/100
32	C,D,E,F,G,H,I,J	23/280	7/90	6/90	10/100
60	B,C,D,E,F,G,H,I,J	45/280	12/90	14/90	19/100
64	B,C,D,E,F,G,H,I,J	58/280	13/90	20/90	24/100
total	41/100	185/2800	43/900	63/900	79/1000

Table 3: Percentage of algorithm pairs that solve successfully all 16 environments

- *mean speed per creature* $= ms(k) = \frac{\sum r_{\max,i}}{k \cdot \sum g_{\max,i}}$. The speed $ms(k)$ is an average over all environments i and is related to one creature. This measure expresses how fast a creature can visit a cell on average (maximum is 100%). This measure should not be used if any environment can not be successfully visited because then the mean speed might be believed higher than reasonable.

- *mean normalized work* $= mw(k) = \frac{k \cdot \sum g_{\max,i}}{\sum r_{\max,i}} = \frac{1}{ms(k)}$. This value represents the work which is necessary, or the costs one has to pay for one creature to visit a cell.

- *relative efficiency* $= \frac{ms(XY\text{-}k)}{ms(XY\text{-}k_{min})}$. First a reference system *XY*-$k_{\min}$ has to be found which can solve the problem with the lowest number $k_{\min}$ of creatures. The relative efficiency relates the mean speed of the system *XY*-k to the mean speed of the reference system *XY*-$k_{\min}$. This measure compares the costs of the reference system *XY*-$k_{\min}$ with the cost of the system *XY*-k. If the relative efficiency is higher than one, the work can be done cheaper with the system *XY*-k. Two similar measures can be defined if the uniform reference system *X*-$k_{\min}$ or the uniform reference system *Y*-$k_{\min}$ is chosen instead of *XY*-$k_{\min}$.

- *absolute efficiency* $= \frac{ms(XY\text{-}k)}{ms(UV\text{-}k_{min})}$. In distinction to the relative efficiency another reference algorithm pair is used. The reference is the fastest of any algorithm pair *UV* (including the uniform "pairs" *UU*) which can solve the problem with a minimum number of creatures. A similar measure can be defined if the fastest of any uniform reference systems *U*-$k_{\min}$ is chosen instead of *UV*-$k_{\min}$.

The efficiency measures are only defined if a $k_{\min}$ exists and if the system *XY*-k solves all 16 environments successfully. We have assumed for the reference algorithm the common time-shuffling mode because it is the least complex (only one state register).

The fastest systems. In the following evaluations we will only take into account those systems that were successful on all environments. The system BCc-64 is the fastest, need-

ing only 218 generations on average to solve the problem (Tab. 4). 8 of the 10 fastest pairs are uniform systems.

No. of Creatures (k)	Algorithm X	Algorithm Y	Shuffling	mean g_max	mean speed per creature	mean normalized work	absolute efficiency compared to J8
64	**B**	**C**	**c**	**218**	**6,89%**	**14,51 €**	**1,038**
60	J	J	u	224	7,13%	14,03 €	1,073
60	**B**	**C**	**c**	**233**	**6,86%**	**14,58 €**	**1,033**
60	G	G	u	245	6,52%	15,34 €	0,982
64	J	J	u	257	5,83%	17,15 €	0,878
60	C	C	u	257	6,22%	16,09 €	0,936
64	B	B	u	261	5,76%	17,38 €	0,867
60	B	B	u	276	5,80%	17,25 €	0,873
64	C	C	u	285	5,27%	18,99 €	0,793
64	G	G	u	314	4,78%	20,92 €	0,720

Table 4: The 10 absolute fastest systems (sorted ascending by "mean $g_{\max}$") which solve all 16 environments. "u" in shuffling means that these are actually uniform systems.

The most efficient systems. When looking at Tab. 5 you will notice that the most efficient (absolute efficiency) system J-32 is uniform with the efficiency of 1.184. Furthermore 7 of the top ten (absolute efficiency) are uniform. The reference system is J-8 for all the systems. The three time-shuffled systems with efficiencies higher than one are BCc-28/64/60. We can conclude that only common time-shuffling led to efficiencies higher than one but not in an amount that was expected when we had started this investigation. Time-shuffled systems with one creature behaved much better than uniform systems with one creature, but it cannot be deduced that time-shuffled systems always behave better than uniform systems.

Considering only successful systems, 7 out of 41 uniform systems and 3 out of 185 time-shuffled systems reach an absolute efficiency higher than one. These results give a hint on what will happen when good agents are randomly combined through time-shuffling, e. g., during genetic methods: only a low percentage will be better than their "parents".

Tab. 5 includes also values for the relative efficiencies which can get higher than the absolute efficiencies, e. g., the relative efficiency of BCc-28 compared to B-12 is 1.881.

We have also counted the different types of conflicts (Tab. 5): (a) static obstacle is on the front cell, (b) another creature is on the front cell (dynamic obstacle), (c) anticipated conflicts (2, 3 or 4 creatures want to visit the same cell). At the moment it is not clear, whether there is a significant relation between the efficiency and the types and frequency of the conflicts.

The overproportionate gain of efficiency for parallel systems with more than one creature can be interpreted as "synergy". We presume this effect is due to many reasons, like: Creatures react individually to their local environment and can be in different states, additional communication implicitely exists by the conflict anticipation mechanism and creatures can start at different positions which can be an advantage.

No. of Creatures (k)	Algorithm X	Algorithm Y	Shuffling	mean g_max	mean speed per creature	mean normalized work	mean conflicts (c)	mean conflicts (b)	mean conflicts (a)	mean conflicts per creature	conflicts per visit	relative efficiency (uniform X)	relative efficiency (uniform Y)	absolute efficiency compared to J-8	realtive efficiency (common XY)	k_min X	k_min Y	k_min XY
32	J	J	u	382	7,86%	12,72 €	8	7	60	74	0,08	1,184	1,184	1,184	3,113	8	8	8
28	**B**	**C**	**c**	**437**	**7,85%**	**12,74 €**	**11**	**6**	**70**	**88**	**0,09**	**1,881**	**1,527**	**1,182**	**4,697**	**12**	**12**	**28**
16	J	J	u	768	7,81%	12,80 €	8	6	126	140	0,15	1,176	1,176	1,176	3,092	8	8	8
28	B	B	u	456	7,52%	13,30 €	9	8	68	84	0,09	1,803	1,803	1,132	4,044	12	12	12
12	J	J	u	1069	7,48%	13,36 €	8	7	155	170	0,18	1,127	1,127	1,127	2,963	8	8	8
28	J	J	u	463	7,41%	13,49 €	8	8	69	86	0,09	1,116	1,116	1,116	2,934	8	8	8
28	C	C	u	475	7,22%	13,85 €	9	8	72	90	0,09	1,404	1,404	1,087	1,664	12	12	12
60	J	J	u	224	7,13%	14,03 €	9	10	35	54	0,06	1,073	1,073	1,073	2,822	8	8	8
64	**B**	**C**	**c**	**218**	**6,89%**	**14,51 €**	**10**	**9**	**35**	**54**	**0,06**	**1,652**	**1,341**	**1,038**	**4,126**	**12**	**12**	**28**
60	**B**	**C**	**c**	**233**	**6,86%**	**14,58 €**	**9**	**10**	**36**	**55**	**0,06**	**1,644**	**1,334**	**1,033**	**4,105**	**12**	**12**	**28**
8	*J*	*J*	*u*	*1807*	*6,64%*	*15,06 €*	*10*	*6*	*262*	*278*	*0,29*	*1,000*	*1,000*	*1,000*	*2,629*	*8*	*8*	*8*

Table 5: The top 10 most absolute efficient (lowest total costs) non-uniform systems. Constraint: all algorithm pairs are successful in all 16 environments (visiting percentage = 100%). “u” in shuffling means that these are actually uniform systems.

5 Conclusion

The creatures’ exploration problem was investigated in the CA model for multiple creatures using time-shuffled combinations of 10 algorithms (behaviors). These algorithms had shown a good performance in former investigations. The analysis was performed for 16 new environments of size 35×35 and 129 obstacles each. New metrics have been defined for such multi creature systems, especially the mean speed, the relative efficiency (comparing the work of a system with an algorithmic similar system using the minimum number of creatures which can solve the problem), and the absolute efficiency (comparing the work of a system with an algorithmic potentially different system using the minimum number of creatures which can solve the problem). A single creature is not successful for all environments. One time-shuffled creature was more successful but still could not visit all environments successfully. The problem could only be solved using at least 8 creatures for the uniform system J-8 and nine other time-shuffled systems. 185 time-shuffled systems out of all 2800 time-shuffled combinations were successful. The overall fastest system is the time-shuffled system BJc-64, but it is not the most efficient. The most efficient system is uniform: J-32. It turned out that the system BCc-28 (28 creatures, algorithms B and C, time-shuffled with a common state) is 18% more efficient than the uniform reference system J-8. Under the top ten most efficient systems are 3 time-shuffled and 7 uniform ones.

Our future work is directed to investigate other methods of time-shuffling or of combining different algorithms. It is also promising to compare creatures which are different in space (non-uniform), study the relation between conflicts and efficiency, and optimizing the behavior through heuristics.

References

[Alb05] Enrique Alba. *Parallel Metaheuristics: A New Class of Algorithms*. John Wiley & Sons, NJ, USA, August 2005.

[DL06] Bruno N. Di Stefano and Anna T. Lawniczak. Autonomous Roving Object's Coverage of its Universe. In *CCECE*, pages 1591–1594. IEEE, 2006.

[FKSL07] Dietmar Fey, Marcus Komann, Frank Schurz, and Andreas Loos. An Organic Computing architecture for visual microprocessors based on Marching Pixels. In *ISCAS*, pages 2686–2689. IEEE, 2007.

[FS05] Dietmar Fey and Daniel Schmidt. Marching-pixels: a new organic computing paradigm for smart sensor processor arrays. In Nader Bagherzadeh, Mateo Valero, and Alex Ramírez, editors, *Conf. Computing Frontiers*, pages 1–9. ACM, 2005.

[HH06] Rolf Hoffmann and Mathias Halbach. Are Several Creatures More Efficient Than a Single One? In Samira El Yacoubi, Bastien Chopard, and Stefania Bandini, editors, *ACRI*, volume 4173 of *Lecture Notes in Computer Science*, pages 707–711. Springer, 2006.

[HH07] Mathias Halbach and Rolf Hoffmann. Solving the Exploration's Problem with Several Creatures More Efficiently. In Roberto Moreno-Díaz, Franz Pichler, and Alexis Quesada-Arencibia, editors, *EUROCAST*, volume 4739 of *Lecture Notes in Computer Science*, pages 596–603. Springer, 2007.

[HHB06] Mathias Halbach, Rolf Hoffmann, and Lars Both. Optimal 6-State Algorithms for the Behavior of Several Moving Creatures. In Samira El Yacoubi, Bastien Chopard, and Stefania Bandini, editors, *ACRI*, volume 4173 of *Lecture Notes in Computer Science*, pages 571–581. Springer, 2006.

[KMF07] Marcus Komann, Andreas Mainka, and Dietmar Fey. Comparison of Evolving Uniform, Non-uniform Cellular Automaton, and Genetic Programming for Centroid Detection with Hardware Agents. In Victor E. Malyshkin, editor, *PaCT*, volume 4671 of *Lecture Notes in Computer Science*, pages 432–441. Springer, 2007.

[MSPPU02] Bertrand Mesot, Eduardo Sanchez, Carlos-Andres Peña, and Andres Perez-Uribe. SOS++: Finding Smart Behaviors Using Learning and Evolution. In R. Standish, M. Bedau, and H. Abbass, editors, *Artificial Life VIII: The 8th International Conference on Artificial Life*, pages 264–273, Cambridge, Massachusetts, 2002. MIT Press.

[Sip97] Moshe Sipper. *Evolution of Parallel Cellular Machines, The Cellular Programming Approach*, volume 1194 of *Lecture Notes in Computer Science*. Springer, 1997.

[ST99] Moshe Sipper and Marco Tomassini. Computation in artificially evolved, non-uniform cellular automata. *Theor. Comput. Sci.*, 217(1):81–98, 1999.

[US06] Andres Upegui and Eduardo Sanchez. On-chip and on-line self-reconfigurable adaptable platform: the non-uniform cellular automata case. In *IPDPS*. IEEE, 2006.

Hybrid Parallel Sort on the Cell Processor

Jörg Keller[1], Christoph Kessler[2], Kalle König[3] and Wolfgang Heenes[3]

[1]FernUniversität in Hagen, Fak. Math. und Informatik, 58084 Hagen, Germany
joerg.keller@FernUni-Hagen.de
[2]Linköpings Universitet, Dept. of Computer and Inf. Science, 58183 Linköping, Sweden
chrke@ida.liu.se
[3]Technische Universität Darmstadt, FB Informatik, 64289 Darmstadt, Germany
kalle_koenig@gmx.de
heenes@ra.informatik.tu-darmstadt.de

Abstract: Sorting large data sets has always been an important application, and hence has been one of the benchmark applications on new parallel architectures. We present a parallel sorting algorithm for the Cell processor that combines elements of bitonic sort and merge sort, and reduces the bandwidth to main memory by pipelining. We present runtime results of a partial prototype implementation and simulation results for the complete sorting algorithm, that promise performance advantages over previous implementations.

Key words: Parallel Sort, Merge Sort, Cell Processor, Hybrid Sort

1 Introduction

Sorting is an important subroutine in many high performance computing applications, and parallel sorting algorithms have therefore attracted considerable interest continuously for the last 20 years, see e.g. [Akl85]. As efficient parallel sorting depends on the underlying architecture [AISW96], there has been a rush to devise efficient parallel sorting algorithms for every new parallel architecture on the market. The Cell BE processor (see e.g. [GEMN07] and the references therein) presents a challenge in this respect as it combines several types of parallelism within a single chip: each of its 8 parallel processing units (called SPEs) is an SIMD processor with a small local memory of 256 KBytes. The SPEs communicate via one-sided message-passing over a high-speed ring network with each other and with the off-chip main memory, that is shared among the SPEs, but not kept consistent. The chip also contains a multi-threaded Power processor core, which however is not of particular interest for our work. Although the majority of applications for Cell seem to be numerical algorithms, there have been several attempts to efficiently implement sorting on that architecture. We present a sorting algorithm that incorporates those attempts by combining bitonic sort, merge sort, and pipelining to reduce the bandwidth to main memory, which is seen as one of the major bottlenecks in Cell. We report on the performance of a partial prototype implementation, and on simulation results for the

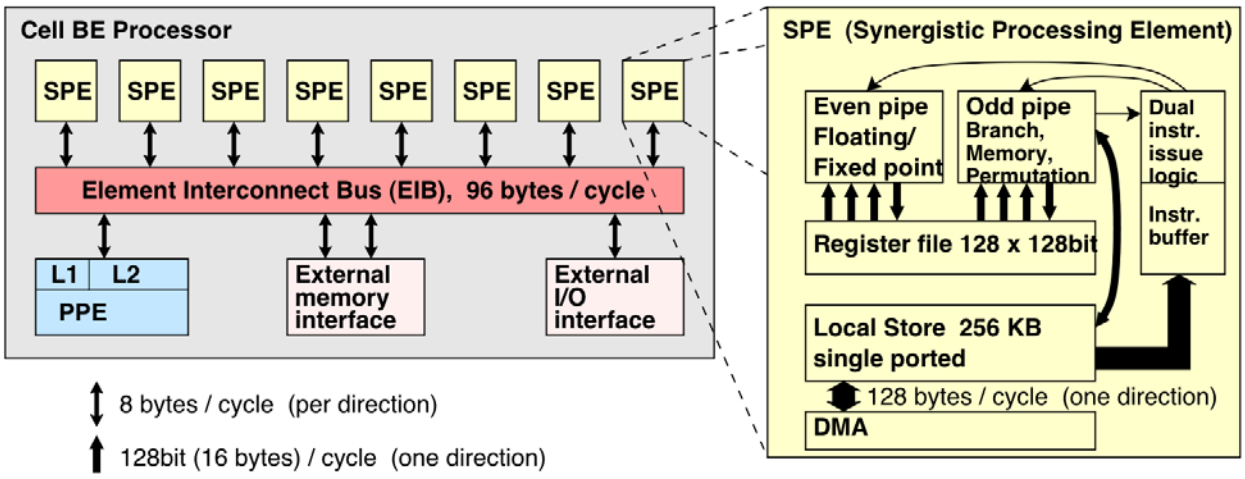


Figure 1: Cell BE Processor Architecture

full sorting algorithm. The combined results indicate that our algorithm outperforms all previous approaches.

The remainder of this article is organized as follows. In Section 2 we provide all technical information necessary to attack the problem at hand, and discuss related work. In Section 3 we detail our solution, report on the performance results for our pipelined merger components of the overall sorting algorithm on a Play Station 3, and extrapolate from these data for the analysis of the expected performance of the overall sorting algorithm. Section 4 concludes and gives an outlook on further developments.

2 Cell Processor and Parallel Sorting

The Cell BE processor [GEMN07] is a multi-core processor consisting of 8 SIMD processors called SPE and a dual-threaded Power core (PPE), cf. Fig. 1. Each SPE has a small local memory of 256 KBytes that contains code and data. There is no cache, no virtual memory, and no operating system on the SPE. The SPE has datapaths and registers 128 bits wide, and instructions to operate on them as on vector registers, e.g. perform parallel comparisons between two registers, each seen as holding four 32-bit values. Hence, control flow instructions tremendously slow down data throughput of an SPE. The main memory is off-chip, and can be accessed by all SPEs and the Power core, i.e. it is a shared memory. Yet, there is no protocol to automatically ensure coherency between local memories and main memory. Data transfer between an SPE and another SPE or the main memory is performed by DMA. Thus, data transfer and computation can be overlapped, but communications must be programmed at a very low level. The SPEs, the PPE core and the memory interface are interconnected by the Element Interconnect Bus (EIB). The EIB is implemented via four uni-directional rings with an aggregate bandwidth of 306.4 GByte/s. The bandwidth of each unit on the ring to send data over or receive data from the ring is only 25.6 GByte/s. Hence, the off-chip memory tends to become the performance

bottleneck. Programming the Cell processor is challenging. The programmer has to strive to use the SPE's SIMD architecture efficiently, and has to take care for messaging and coherency, taking into account the rather small local memories.

The Cell processor seems to have two major application fields: gaming[1] and numerical algorithms. To our knowledge, there are only a few investigations about parallel sorting algorithms for the Cell processor, most notably [GBY07, IMKN07] that use bitonic sort and merge sort, respectively. There is also an implementation of radix sort [RB07], but only with a 4 bit radix, because for an 8 bit radix the array of counters would not fit into the SPE's local store. Also [GBY07] reports to have investigated radix sort but that it "involve(s) extensive scalar updates using indirection arrays that are difficult to SIMDize and thus, degrade the overall performance."

The paper [IMKN07] is quite close to our work and appeared only a few months before this article was written. Both [GBY07] and [IMKN07] work in two phases to sort a data set of size n, with local memories of size m. In the first phase, blocks of data of size $8m$ that fit into the combined local memories of the SPEs are sorted. In the second phase, those sorted blocks of data are combined to a fully sorted data set.

In [GBY07], the first phase is realized in two stages: in the first stage, each SPE sorts data in its local memory sequentially by a variant of Batcher's bitonic sort, then the SPEs perform Batcher's bitonic sort on the sorted data in their local memories. The combined content of their local memories is then written to main memory as a sorted block of data. This is repeated $n/(8m)$ times until the complete data set is turned into a collection of sorted blocks of data. The second phase performs Batcher's bitonic sort on those blocks. Batcher's sort is chosen because it needs no data dependent control flow and thus fully supports the SPE's SIMD architecture. The disadvantage is that $O(n \log^2 n)$ data have to be read from and written to main memory, which makes the main memory link the limiting speed factor, and the reason why the reported speedups are very small.

In [IMKN07], the first phase is also realized in two stages: in the first stage, each SPE performs a variant of combsort that exploits the SIMD capability of the SPE, then the SPEs perform a mergesort on the sorted data in their local memories. As in the first approach, this is repeated $n/(8m)$ times. The second phase is mergesort, that uses a so-called vectorized merge to exploit the SIMD instructions of the SPEs, and employs a 4-to-1-merge to reduce memory bandwidth. Yet, each merge procedure reads from main memory and writes to main memory, so that $n \log_4(n/(8m))$ data are read from and written to main memory during the second phase.

Our approach focuses on the second phase, as the first phase only reads and writes n data from and to the main memory, and thus is not as critical to the overall performance as the second phase. Also, there are known approaches for the first phase. We also implemented a vectorized merge routine, similar to that of [IMKN07] (then unknown to us), only that we perform 2-to-1 merges. The vectorization uses a variant of Batcher's bitonic sort to merge chunks of four successive 32-bit integers, as those will fit into one Cell register. However, there is a notable difference between our approach and that of [IMKN07]. We run mergers of several layers of the merge tree concurrently to form a pipeline, so that

[1] In a variant with 6 SPEs, Cell is deployed in the Play Station 3.

output from one merger is not written to main memory but sent to the SPE running the follow-up merger. Thus, we can realize 16-to-1 or 32-to-1 mergers between accesses to main memory, and reduce the memory bandwidth by a factor of 2 and 2.5, respectively, in relation to [IMKN07]. In order to exploit this advantage we have to ensure that our pipeline runs close to the maximum possible speed, which requires consideration of load balancing. More concretely, if a merger M must provide an output rate of k words per time unit, then the two mergers M_1, M_2 feeding its inputs must provide a rate of $k/2$ words per time unit on average. However, if the values in M_2 are much larger than in M_1, the merger M will only take values from the output of M_1 for some time, so that the merger M_1 must be able to run at a rate of k words for some time, or the output rate of M will reduce!

In principle, we could have mapped each layer of a binary merge tree onto one SPE, each SPE working the mergers of its layer in a time-slot fashion. A time slot is the time that a merger needs to produce one buffer full of output data, provided that its input buffers contain enough data. Thus, with k SPEs, we realize a 2^k-to-1 merge. This will balance load between the layers as the combined rate from one layer to the next is the same for all layers. Also, because of finite size buffers between the mergers, if M only draws from M_1, M_2 will not be able to work further and thus M_1 will get more time slots and be able to deliver faster. The disadvantage of this model is that the larger i, the more mergers SPE i has to host, which severely restricts the buffer size, because there must be one output buffer and two input buffers for each merger, that all must fit into about half the local memory of an SPE (the other half is for code and variables). Therefore, we devised mappings that minimize the maximum amount of mergers that one SPE has to host, and thus maximize the buffer size. We present two such mappings in the next section.

3 Experiments and Simulations

We have implemented the prototype core of a merger routine on a Cell processor from a Play Station 3. Despite including only 6 SPEs, it corresponds to the processor sold in blades by IBM. Our routine provides a bandwidth of 1.5 GByte/s. This indicates that with 8 SPEs concurrently reading and writing data as in [IMKN07], a bandwidth to main memory of $2 \times 8 \times 1.5 = 24$ GByte/s would be needed which would saturate the memory link. Assuming that the fully developed merger in [IMKN07] is more efficient than our prototype, we see that the bandwidth to main memory is the limiting performance factor in their design. Conversely, if we can reduce the memory bandwidth needed, we gain a corresponding factor in performance.

In order to get an impression of the performance of our full algorithm, we implemented a discrete event simulation[2] of the sorting algorithm. As the runtime of the merger core is only dependent on the size of the output buffer, it is more or less constant. As furthermore communication and computation are overlapped, we believe the simulation to accurately reflect the runtime of the full algorithm.

[2]While the merge algorithm is not very complex, implementing the DMA transfers is cumbersome and low-level, thus we decided to postpone the full implementation.

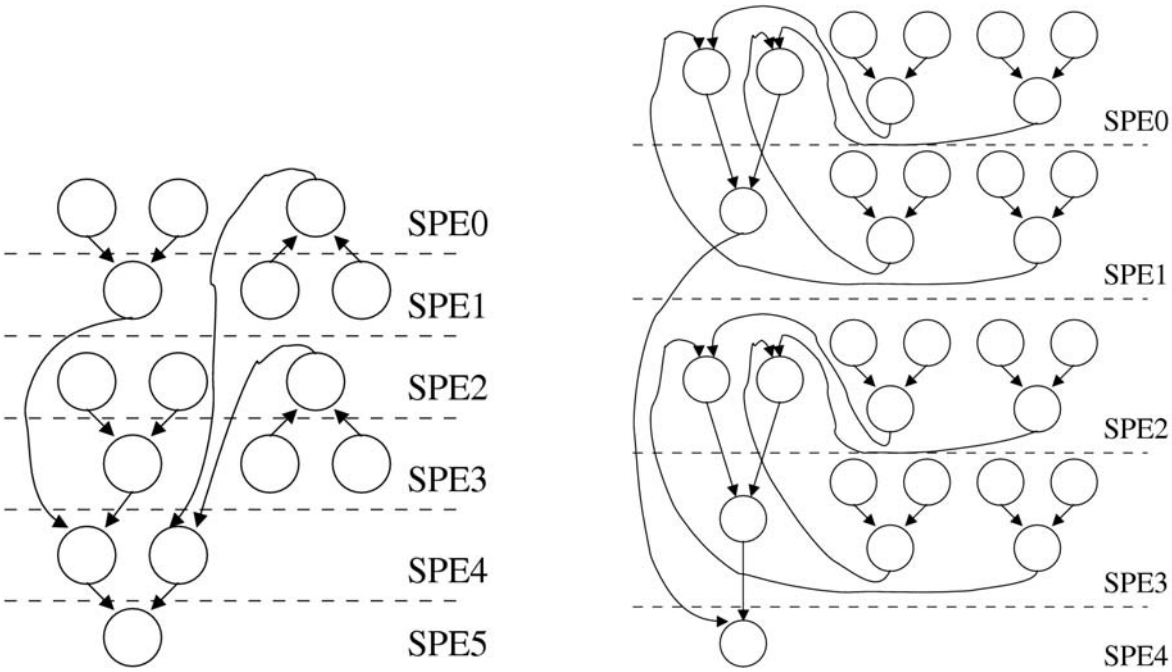


Figure 2: Mapping of merger nodes to SPEs

In each step, each SPE runs one merger with enough input data until it has produced one output buffer full of data. As buffer size, we use 4 KByte for the output buffer (holding 1,024 32-bit integers), and 2×4 KByte for the input buffers, in order to allow concurrent working of a merger and filling of its input with the output of a previous merger. Each merger receives a share of the SPE's processing time at least according to its position in the merge tree. For example, in Fig. 2(b), the merger left in SPE1 receives one half of the processing power, because it is placed in depth 1 of the merge tree, while the other mergers receive 1/8 and 1/16 of the processing power, respectively, because they are in depths 3 and 4, respectively. We use a simple round robin scheduling policy in each SPE, where a merger not ready to run because of insufficient input data or full output buffer is left out.

We have investigated two mappings, depicted in Fig. 2. Both try to place neighboring mergers in one SPE as often as possible, in order to exploit the load balancing discussed in the previous section. In mapping (a), the mergers of SPE0 and SPE1 (similarly SPE2 and SPE3) could have been placed in one SPE, but we decided to give them twice the processor share to be on the safe side and avoid load balancing and pipeline stall problems. We have simulated this mapping with 16 input blocks of 2^{20} sorted integers each. The blocks were randomly chosen and then sorted. In all experiments, the pipeline ran with 100% efficiency as soon as it was filled. As we realize a 16-to-1 merge, we gain a factor of 2 on the memory bandwidth in relation to [IMKN07]. Yet, as we need 6 instead of 4 SPEs to do this, our real improvement is only $2 \cdot 4/6 = 4/3$ in this case.

In mapping (b), we have implemented a 32-to-1 merge, with the restriction that no more than 8 mergers are to be mapped to one SPE. With 20 KByte of buffers for each merger, this seems to be upper limit. Here each merger has a processing share according to its

position in the merge tree. We used 32 input blocks of 2^{20} sorted integers each, chosen as before. The pipeline ran with an efficiency of 93%, meaning that in 93% of the time steps, the merger on SPE4 could be run and produced output. In comparison to [IMKN07], our memory bandwidth decreased by a factor of 2.5. Combined with a pipeline efficiency of 93%, we still gain a factor of 1.86 in performance.

4 Conclusion and Future Work

We have provided a new sorting algorithm for the Cell Processor Architecture that uses a vectorized merge sort in a pipelined variant to reduce memory bandwidth. Our simulation results indicate that the performance of a full implementation of our algorithm will show better performance than previous algorithms. Future work will consist of obtaining this implementation.

Note that our algorithm is also able to run on multiple Cell processors, as does [IMKN07]. At the beginning, there will be many blocks, and hence many 16-to-1 or 32-to-1 mergers can be employed. In the end, when nearing the root, we are able to employ a method already known and mentioned in [IMKN07]: we partition the very large data blocks and perform merges on the partitions in parallel.

References

[AISW96] Nancy M. Amato, Ravishankar Iyer, Sharad Sundaresan, and Yan Wu. A Comparison of Parallel Sorting Algorithms on Different Architectures. Technical Report 98-029, Texas A&M University, January 1996.

[Akl85] Selim G. Akl. *Parallel Sorting Algorithms*. Academic Press, 1985.

[GBY07] Bugra Gedik, Rajesh Bordawekar, and Philip S. Yu. CellSort: High Performance Sorting on the Cell Processor. In Christoph Koch, Johannes Gehrke, Minos N. Garofalakis, Divesh Srivastava, Karl Aberer, Anand Deshpande, Daniela Florescu, Chee Yong Chan, Venkatesh Ganti, Carl-Christian Kanne, Wolfgang Klas, and Erich J. Neuhold, editors, *VLDB*, pages 1286–1207. ACM, 2007.

[GEMN07] Michael Gschwind, David Erb, Sid Manning, and Mark Nutter. An Open Source Environment for Cell Broadband Engine System Software. *Computer*, 40(6):37–47, 2007.

[IMKN07] Hiroshi Inoue, Takao Moriyama, Hideaki Komatsu, and Toshio Nakatani. AA-Sort: A New Parallel Sorting Algorithm for Multi-Core SIMD Processors. In *Proc. 16th Int.l Conf. on Parallel Architecture and Compilation Techniques (PACT)*, pages 189–198. IEEE Computer Society, 2007.

[RB07] N. Ramprasad and Pallav Kumar Baruah. Radix Sort on the Cell Broadband Engine. In *Int.l Conf. High Perf. Comuting (HiPC) – Posters*, 2007.

An optimized ZGEMM implementation for the Cell BE

Timo Schneider[1], Torsten Hoefler[2], Simon Wunderlich[1], Torsten Mehlan[1], and Wolfgang Rehm[1]

[1]Technical University of Chemnitz, Strasse der Nationen 62,
Dept. of Computer Science, Chemnitz, 09107 GERMANY
{timos, siwu, tome, rehm}@hrz.tu-chemnitz.de

[2]Indiana University, Open Systems Lab,
150 S Woodlawn Ave, Bloomington, IN, 47405 USA
htor@cs.indiana.edu

Abstract:
The architecture of the IBM Cell BE processor represents a new approach for designing CPUs. The fast execution of legacy software has to stand back in order to achieve very high performance for new scientific software. The Cell BE consists of 9 independent cores and represents a new promising architecture for HPC systems. The programmer has to write parallel software that is distributed to the cores and executes subtasks of the program in parallel. The simplified Vector-CPU design achieves higher clock-rates and power efficiency and exhibits predictable behavior. But to exploit the capabilities of this upcoming CPU architecture it is necessary to provide optimized libraries for frequently used algorithms. The Basic Linear Algebra Subprograms (BLAS) provide functions that are crucial for many scientific applications. The routine ZGEMM, which computes a complex matrix–matrix–product, is one of these functions. This article describes strategies to implement the ZGEMM routine on the Cell BE processor. The main goal is achieve highest performance. We compare this optimized ZGEMM implementation with several math libraries on Cell and other modern architectures. Thus we are able to show that our ZGEMM algorithm performs best in comparison to the fastest publicly available ZGEMM and DGEMM implementations for Cell BE and reasonably well in the league of other BLAS implementations.

1 Introduction

Matrix multiplication is used for many standard linear algebra problems, such as inverting matrices, solving systems of linear equations, and finding determinants and eigenvalues [Kny01]. Therefore if a new architecture wants to be successful in the scientific computing environment it is crucial that optimized libraries for problems like matrix multiplication and alike are freely available.

Our initial intent was to port ABINIT, a quantum mechanical ab-inito simulator [GBC+02, GCS+00, BLKZ07], to the Cell BE architecture.[1] ABINIT heavily uses the BLAS [LHKK79, DCHH88] function `ZGEMM` which multiplies two complex matrices and adds

[1]This research is supported by the Center for Advanced Studies (CAS) of the IBM Böblingen Laboratory as part of the NICOLL Project.

them to a third one. All input matrices and scalars are given in double precision. Different groups have already developed optimized matrix multiplication codes for the Cell BE architecture, but those were not meant to be used by other applications but to demonstrate the good single-precision capabilities of the architecture [D.07]. They operate on matrices on a fixed input size, partly use the rather uncommon block data layout for storing the matrices and only work for single precision floating point numbers. Another possibility would have been to use PPC64 optimized BLAS libraries [CGG02, DDE+05] like Atlas [WD98] ot Goto [KR02]. But these libraries do not leverage the potential of the Cell BE completely because they only use the PPC64 core.

Thus, we decided to implement a Cell optimized version of `ZGEMM`. In this paper we will describe the basic algorithm we used as well as the optimization principles we had to apply to get the current result which we will benchmark, too. This paper is organized as follows: Section 2 contains background information on relevant aspects of the Cell Broadband Engine architecture, Section 3 gives an overview of the `ZGEMM` Fortran interface; Section 4 shows how to vectorize and optimize the naive implementation of the used algorithm, Section 5 gives some benchmarking results and tries to explain them, Section 6 draws some conclusions and shows directions for further improvement.

2 Cell Broadband Engine Overview

The Cell BE architecture [CD07] is a multicore microprocessor with one general purpose core called *Power Processing Element* (PPE) and multiple vector co-processing elements, called *Synergistic Processing Elements* (SPE). The PPE is a stripped-down general purpose core to administer the SPEs, which handle the computational workload. It is easy to run conventional software on the PPE due to its compatibility to the PPC64 architecture. The PPE is connected to the system memory and the SPEs via the *Element Interconnect Bus* (EIB), a high-bandwidth circular data bus. Each SPE hosts some local memory, called *Local Store* (LS), an *Synergistic Execution Unit* (SXE) and a Memory Flow Controller (MFC) which connects the SPE to the EIB. The MFC operates independently of the SPU, so memory transactions can be overlapped with computations. Figure 1 gives an overview of the Cell BE architecture.

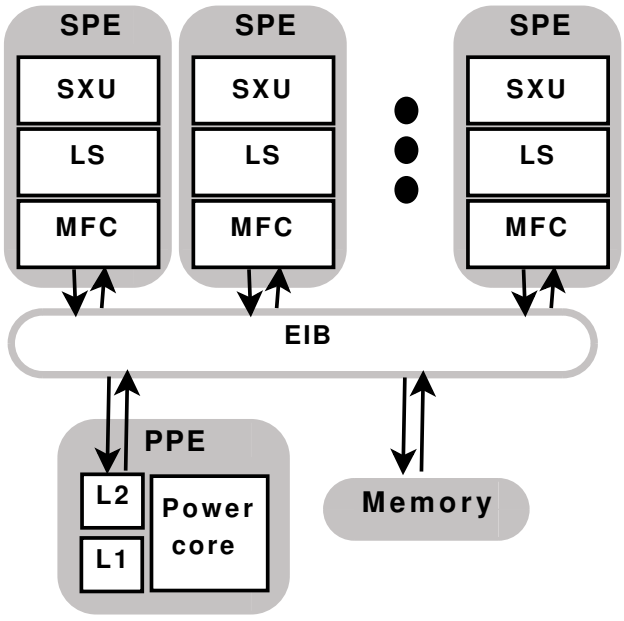


Figure 1: Cell BE architecture

Let's take a closer look at the SPEs now, as the performance of our implementation depends solely on our ability to use their full potential. This is not always easy, as SPEs are special cores: They are meant to fit in the gap between standard desktop PC cores and special number crunching devices like the Graphics Processing Units (GPUs) in graphics

cards. The SPEs can not access the main memory directly, they can only operate on their Local Store which is capable of holding 256 KiB data. One should not think of the LS as of a cache in a standard CPU as it is not updated automatically or transparently to the running process. To get new data into the LS one has to use the Memory Flow Controller to issue a DMA PUT or GET transfer. The boundaries of DMA transfers from and to SPUs have to be 16 byte aligned. DMA transfers yield the best performance when multiples of 128 byte are transferred.

The Synergistic Execution Unit is a vector processor which operates on 128 registers, each 128 bit wide. That means when coping with double precision floating point numbers (which are 8 byte wide) we can do two similar operations simultaneously if we manage to put our input data in a single vector. Unfortunately the Cell BE processors available at the time of writing are very slow when doing double precision arithmetic (1.83 GFlop/s per SPE [WSO+06], which is 14 times lower than the single precision performance). But this should improve with future generations of this chip. The performance cited above can only be reached when fused multiply add instructions are used. These instruction perform the operation $c := a * b + c$ or similar and therefore count as two floating point instructions (FLOP). As all double precision arithmetic instructions need the same number of clock cycles, these instructions yield the best floating point operation per second (Flop/s) ratio.

3 BLAS/ZGEMM

Basic Linear Algebra Subprograms (BLAS) is an widely used application programming interface for libraries to perform basic linear algebra operations such as matrix multiplication. They were first published in 1979 [LHKK79]. Highly optimized implementations of the BLAS interface have been developed by different vendors or groups for many architectures.

`ZGEMM` performs the matrix-matrix operation on input of complex numbers:

$$C := \alpha \cdot op(A) \cdot op(B) + \beta \cdot C$$

Where $op(A)$ specifies if the normal, transposed or conjugated version of the matrix is to be used. A,B and C are matrices consisting of complex numbers and α and β are complex scalars. The Fortran interface is:

```
SUBROUTINE ZGEMM(TRANSA, TRANSB, M, N, K, ALPHA, A, LDA, B,
LDB, BETA, C, LDC)
```

`TRANSA` and `TRANSB` contains the operator to be used on matrix `A` and `B` as a single character which can be `n` (normal), `t` (transposed) or `c` (transposed, conjugated). $op(A)$ is `M` by `K` matrix, $op(B)$ is a `K` by `N` matrix, and `C` is a `M` by `N` matrix. Note that `M`, `N` and `K` refer to the matrices after the operators are applied, not the original input matrices. `ALPHA` and `BETA` correspond to α and β in the equation above. `LDA`, `LDB` and `LDC` specify the first dimension of the input matrices so it is possible to use `ZGEMM` the top-left part of the input matrices only.

The input matrices `A`, `B` and `C` are stored in column major order, as they come from a program written in Fortran. Figure 2 illustrates the meaning of the different `ZGEMM` parameters which deal with the representation of the input matrices.

The tricky part is the operator: Depending on if its normal or not, elements which are stored sequentially in memory can be in one row or one column. As one result element is computed based on one row of $op(A)$ and one column of $op(B)$, we will always have to consider the operators for our memory access.

We investigated works that use Strassen's or Winograd's implementation to reduce the asymptotic complexity of the matrix multiplication [DHSS94]. However, those optimized algorithms work only with well conditioned matrixes which we can not guarantee in the general case. Thus, we chose to implement a traditional $O(N^3)$ algorithm for our ZGEMM.

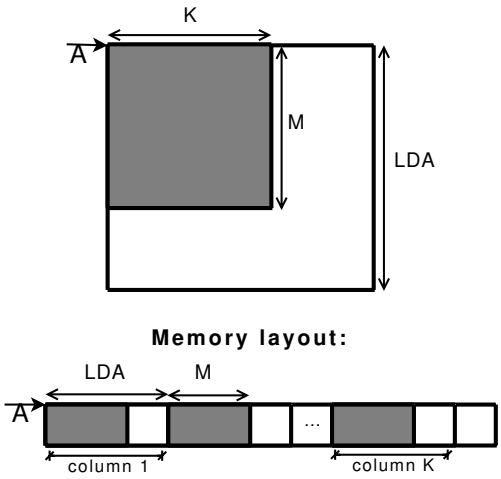


Figure 2: Fortran/ZGEMM Matrix representation

4 Our ZGEMM implementation

We had to apply two important concepts to be able to design a well-performing `ZGEMM` implementation: We partitioned the input data, distributed it among the available SPEs and vectorized all calculations on order to exploit the SIMD architecture.

4.1 Data Partitioning

As the Local Store of an SPE space is limited to 256KiB, the goal should be to save space and memory transfers. A first idea was to load parts of a row of $op(A)$ and a column of $op(B)$ and to compute exactly one element of C. There are some problems with this: depending on the operator, the rows (or columns) of the matrices are stored sequentially in memory or scattered with a displacement (of `LDx`), forcing us to get each element separately. This would decrease performance, as the MFC operates best with memory chunks that are multiples of 128 byte in size.

A better idea is to load blocks instead of lines, and perform small matrix-matrix multiplications instead of scalar products. This gives us independence from the operator: the decision whether rows or columns should be used in the scalar product of the matrix multiplications on the SPEs does not affect performance, as we have random access to the Local Store. Another advantage is the number of operations. For n elements which fit in each input buffer of our Local Store, $\mathcal{O}(n)$ multiply and add operations can be done with the scalar product, but $\mathcal{O}(\sqrt{n^3}) = \mathcal{O}(n^{1.5})$ operations can be achieved with small matrix multiplications. Of course, with more operations on the same amount of local data the total number of memory transfers is reduced.

4.2 Work Assignment

With our partitioning approach, each part of the result matrix can be independently computed with the block row of $op(A)$ and the block column of $op(B)$. The blocks to be

computed are simply distributed circular on the SPEs. Figure 3 illustrates the assignment scheme for 6 SPEs. The shaded result block is computed using the shaded row in $op(A)$ and the shaded column in $op(B)$.

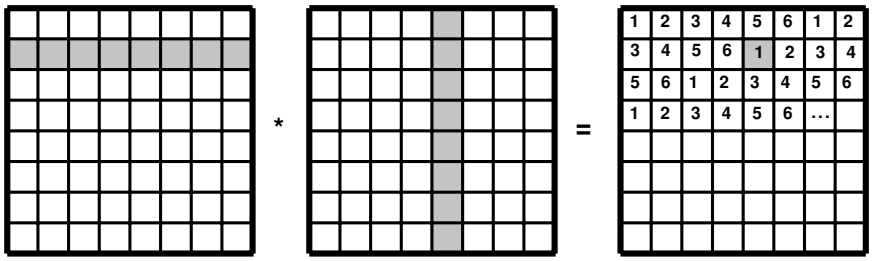


Figure 3: SPE Block assignment

We investigated the use of the PPE with an experimental implementation. The PPE has a theoretical peak performance of 6.4 GFlop/s. Our code spawns N threads on the PPE, each of them computes the same chunk of $op(C)$ as an SPE does[2], using a PPC970 optimized BLAS implementation to perform the computation. Despite the given peak performance of the SPE, we achieved only 1.7 GFlop/s with ATLAS on the PPE, which makes this partitioning scheme suboptimal. Thus, we did not include the PPE measurements in our benchmarks.

4.3 Vectorization

In our matrix multiplication, each element is a 128 bit complex number, consisting of 64 bit double precision floating point values for real part and imaginary part. We can safely assume that only fused multiply add operations are used, as two elements of each matrix are multiplied and added to the temporary scalar product. One multiply-add operation of complex numbers a and b added to y ($y = y + a \cdot b$) is split up like this for its real and imaginary parts:

$$y_{re} := y_{re} + a_{re}b_{re} - a_{im}b_{im}$$
$$y_{im} := y_{im} + a_{re}b_{im} + a_{im}b_{re}$$

This makes 4 fused multiply add operations, with 64 bit operands. With the SIMD-ability of the SPU, two complex multiply-adds can be done instead of one. To use SIMD instructions, the real parts and imaginary parts have to be splitted and packed into separate registers. This can be done with the SPU shuffle instruction. Now the calculation can be done as described above, and the only thing left to do is to separate the real and imaginary part into the result registers before we write back into `C`.
One little obstacle remains: The fused multiply subtract operation on the SPU `spu_msub(a, b, c)` calculates $a \cdot b - c$, but we would need $c - a \cdot b$. To achieve this without adding further instructions to change the sign, the real part can be calculated as follows:

$$y_{re} := a_{re}b_{re} - ((a_{im}b_{im}) - y_{re})$$

In Figure 4 you can see how the blockwise matrix multiplication can be implemented in C, using the SPU intrinsics. [3]

[2] theoretically, $N = 3$ should be optimal

[3] Our code and the tests that were used to obtain the presented benchmark results can be fetched from `http://files.perlplexity.org/zgemm.tar.gz`.

```
#define VPTR "(vector double *)"
vector char high_double = { 0, 1, 2, 3, 4, 5, 6, 7,
                           16,17,18,19,20,21,22,23};
vector char low_double  = { 8, 9,10,11,12,13,14,15,
                           24,25,26,27,28,29,30,31};
vector double rre={0,0}, rim={0,0}, tre, tim, sre, sim;

for (k=0; k < klen; k++, aa += astep, bb += bstep) {
  fim = spu_shuffle(*(VPTR aa), *(VPTR (aa+atstep)), low_double);
  gim = spu_shuffle(*(VPTR bb), *(VPTR bb), low_double);
  fre = spu_shuffle(*(VPTR aa), *(VPTR (aa+atstep)), high_double);
  gre = spu_shuffle(*(VPTR bb), *(VPTR bb), high_double);
  tre= spu_nmsub( fim, gim, sre);
  tim= spu_madd( fre, gim, sim);
  sre= spu_msub( fre, gre, tre);
  sim= spu_madd( fim, gre, tim);
}

rre = spu_shuffle(sre, sim, high_double);
rim = spu_shuffle(sre, sim, low_double);
*(VPTR cc) = spu_add(*(VPTR cc), rre);
*(VPTR (cc+1)) = spu_add(*(VPTR (cc+1)), rim);
```

Figure 4: Inner loop of the blockwise matrix multiplication, implemented in C

5 Benchmarks

This section provides a performance evaluation of our implementation and a qualitative and quantitative comparison to BLAS implementations on other modern architectures.

The current Cell BE chip's SPEs are capable of issuing one double precision arithmetic instruction every six clock cycles. This instruction needs another seven cycles until the result is available in the target register. But if we assume to execute a very large number of data-independent double precision operations we would get a cycles per instruction (CPI) value of 6. Considering FMADD operations and a vector size of two, the theoretical peak performance of a single Cell BE CPU with 8 SPE and a clock

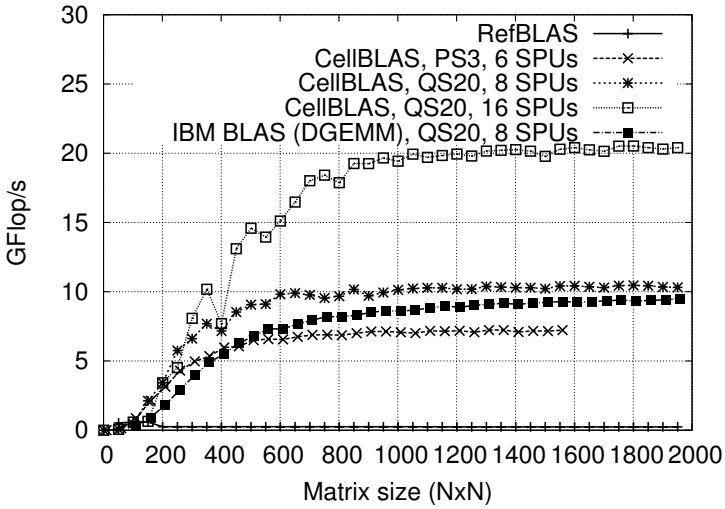


Figure 5: Performance Comparison

rate of 3.2 GHz is

$$R_{peak} = \frac{3.2 * 10^9\,\mathrm{Hz}}{6} \cdot 8\,\mathrm{SPE} \cdot 4\,\mathrm{Flop/SPE} = 17.07\,\mathrm{GFlop/s}$$

This is the number in theory, in practical tests (back to back execution of fused multiply add instructions with no data dependencies) we were able to measure up to 14.5 GFlop/s. This number is said to be the Cell BE double precision peak performance. [WSO+06]

Even though our implementation supports arbitrary matrices, we benchmarked square matrices to enable easy comparisons to other publications. We used `ppu-gcc`, version 4.1.1 with the flags `-O3 -mabi=altivec -maltivec` to compile all PPE code and `spu-gcc`, version 4.1.1 with `-O3` for the SPE code. The Cell BE specific benchmarks were run on a 3.2 GHz IBM QS20 Cell Blade, which contains 2 Cell BE processors with 8 SPEs per processor and two 512 MiB RAM banks and a Playstation 3 running at 3.2 GHz with 200 MiB memory. Both systems run Linux 2.6 (with IBM patches applied).

In our first benchmark (Figure 5), we compare the performance of netlib.org's refblas `ZGEMM` with the IBM `DGEMM` implementation[4] and our optimized implementation for different matrix sizes.

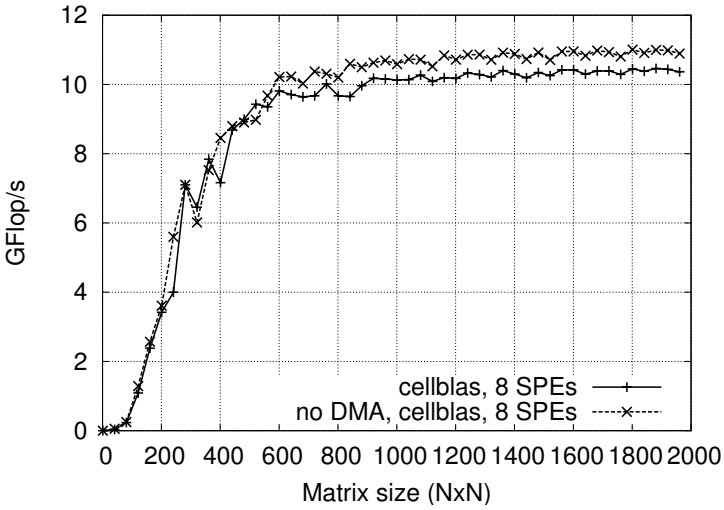


Figure 6: Effects of overlapping the memory accesses with computation

The results show that the our implementation performs very well on Cell BE CPUs. Even though we tried to tune refblas by using different numactl configurations (numactl controls which CPU uses which memory bank), we were not able to achieve more than one Gflop. This is due to the fact that the current compilers do not automatically generate code for the SPUs. Thus, the refblas implementation used only the rather slow PPC core. We outperform the IBM DGEMM implementation by large for all different matrix sizes and our code scales very well to up to 16 SPUs. We can also reproduce similar performance on the specialized Playstation 3 (PS3) hardware (only 6 SPEs are accessible with Linux).

Another optimization technique that has been proposed [CRDI07] is to overlap memory (DMA) accesses with computation. However, this increases the code complexity significantly. To evaluate the potential benefit, we removed all the memory (DMA) accesses from our implementation to simulate the overlap. This invalidates the results but provides an upper bound to the performance-gain due to overlap. Figure 6 shows the comparison to our implementation. Our experiments show that we could gain up to one Gflop/s performance with this overlap technique.

[4]The current IBM BLAS implements no `ZGEMM`. Thus, we used `DGEMM` for comparison, because of its similarity to `ZGEMM`

Our next benchmark compares the Cell BE and our optimized implementation (currently the fastest available) with different modern High Performance Computing (HPC) architectures. We chose a variety of different systems to be able to evaluate the suitability of the Cell BE for scientific calculations using `ZGEMM` as an example. The different systems and their peak floating point performances are described in the following. We leveraged all available processing units (CPUs/Cores) that share a common system memory (are in the same physical node). Thus we compare our multi-core Cell BE implementation with other multi-core BLAS implementations. The test systems are described in the following: a node in Big Red has two dual-core PowerPC 970 MP processors (2.5GHz) with 8GB RAM per node. The peak-performance (with FMADD) is 40 GFlop/s and we ran the IBM ESSL library. We used the Goto BLAS [KR02] library 1.19 on Odin, a dual CPU dual-core Opteron running at 2 GHz with a peak performance of 16 GFlop/s, and Sif, a dual CPU quad-core 1.86 GHz Intel Xeon with 59.5 GFlop/s peak. The theoretically fastest tested system, Jeltz, as two quad-core Intel Xeon 3.0 GHz and a peak performance of 96 GFlop/s. Jeltz runs Mac OS X Tiger and we used the vendor supplied vecLib for our experiments. The absolute performance results for all those systems are plotted in Figure 5.

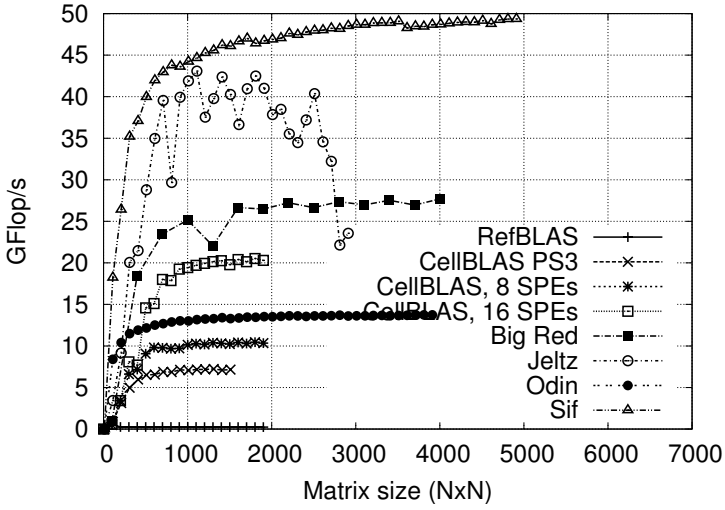


Figure 7: Architectural comparison of ZGEMM Performance

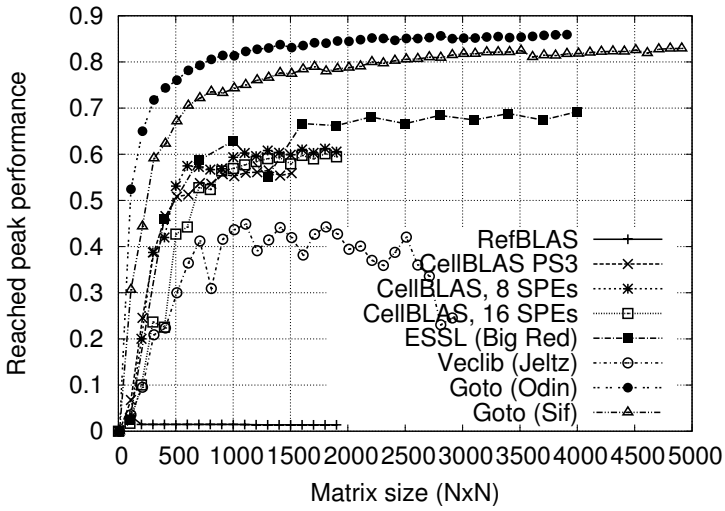


Figure 8: Relative efficiency of different BLAS implementations

Due to memory and CPU time limits, not all matrix sizes could be run on all systems (e.g., the PS3 had only 200 MiB). Our benchmarks show that the current generation Cell BE is not really suited to perform double precision floating point calculations because it is largely outperformed by systems in the same and lower price-range. However, the specialized low-cost Playstation 3 makes a big difference in this price-performance game but its limited memory might be a big obstacle to scientific use.

Those absolute value comparisons do not allow any qualitative comparisons between the different libraries. The main problem is the high variance in peak performance. To compare our implementation to other BLAS libraries, we normalized the measured performance to the peak performance of the architecture to get an estimate of the efficiency of use of the floating point units. We expect a pretty high efficiency on the standard superscalar and cache-based architectures due to the high spatial and temporal locality in matrix multiplication algorithms and decades of development. However, the Cell BE represents a completely new approach of the "explicit cache" (Local Store). Additionally to that, the Cell architecture introduces additional overheads for loading the code to the SPUs. The relative performance results are presented in Figure 7. The highly optimized Goto BLAS implementation delivers the best performance on the available architectures. IBM's Engineering and Scientific Subroutine Library (ESSL) delivers good performance in Power PPC. Our implementation which explores a new CPU architecture is performing very well in comparison to the well established ones and even better than Apple's Veclib.

6 Conclusion and Future Work

Since scientific simulations heavily rely on optimized linear algebra functions we presented in this article an optimized `ZGEMM` implementation for the IBM Cell BE processor. As a part of the BLAS package, the `ZGEMM` routine performs a complex matrix–matrix multiplication. We discussed the strategies to distribute data and to exploit the double precision floating point elements of the SPEs.

The benchmarks showed that the performance of our `ZGEMM` algorithm achieves up to 70% of the peak performance and scales linearly from 1 to 16 SPEs. We assume that our code will also perform well on the next generation Cell BE which supports a fully-pipelined double precision unit that does not stall 6 cycles after every instruction. We compared the algorithm with the IBM `DGEMM` implementation since there is no `ZGEMM` implementation available for Cell. We also showed that even without applying double buffering techniques, the SPEs can be used efficiently under the condition that the number of calculations grow faster with problem size than the access to memory.

Our `ZGEMM` implementation shows the best performance of all publicly available `ZGEMM` or `DGEMM` implementations for Cell BE. Thus, our work may serve as guideline for implementing similar algorithms.

References

[BLKZ07] F. Bottin, S. Leroux, A. Knyazev, and G. Zerah. Large scale parallelized ab initio calculations using a large 3D grid of processors. *submitted to Computational Material Sciences*, 2007.

[CD07] Johns C.R. and Brokenshire D.A. Introduction to the Cell Broadband Engine Architecture. *IBM Journal of Research and Development*, 51:503–519, 2007.

[CGG02] J. Cuenca, D. Gimenez, and J. Gonzalez. Towards the design of an automatically tuned linear algebra library. *Parallel, Distributed and Network-based Processing, 2002. Proceedings. 10th Euromicro Workshop on*, pages 201–208, 2002.

[CRDI07] T. Chen, R. Raghavan, J. N. Dale, and E. Iwata. Cell Broadband Engine Architecture and its first implementationA performance view. *IBM Journal of Research and Development*, 51:559–572, 2007.

[D.07] Hackenberg D. Fast Matrix Multiplication on Cell (SMP) Systems. Technical report, TU Dresden, Center for Information Services, 2007.

[DCHH88] J. J. Dongarra, J. Du Croz, S. Hammarling, and R. J. Hanson. An extended set of FORTRAN Basic Linear Algebra Subprograms. In *In ACM Trans. Math. Soft., 14 (1988), pp. 1-17*, 1988.

[DDE+05] J. Demmel, J. Dongarra, V. Eijkhout, E. Fuentes, A. Petitet, R. Vuduc, R.C. Whaley, and K. Yelick. Self-adapting linear algebra algorithms and software. *Proceedings of the IEEE*, 93(2):293–312, Feb 2005.

[DHSS94] Craig C. Douglas, Michael Heroux, Gordon Slishman, and Roger M. Smith. GEMMW: a portable level 3 BLAS Winograd variant of Strassen's matrix-matrix multiply algorithm. *J. Comput. Phys.*, 110(1):1–10, 1994.

[GBC+02] X. Gonze, J.-M. Beuken, R. Caracas, F. Detraux, M. Fuchs, G.-M. Rignanese, L. Sindic, M. Verstraete, G. Zerah, F. Jollet, M. Torrent, A. Roy, M. Mikami, Ph. Ghosez, J.-Y. Raty, and D.C. Allan. First-principles computation of material properties: the ABINIT software project. *Computational Materials Science*, 25:478–493, 2002.

[GCS+00] X. Gonze, R. Caracas, P. Sonnet, F. Detraux, Ph. Ghosez, I. Noiret, and J. Schamps. First-principles study of crystals exhibiting an incommensurate phase transition. *AIP Conference Proceedings*, 535:163–173, 2000.

[Kny01] Andrew V. Knyazev. Toward the Optimal Preconditioned Eigensolver: Locally Optimal Block Preconditioned Conjugate Gradient Method. *SIAM J. Sci. Comput.*, 23(2):517–541, 2001.

[KR02] Goto K. and Geijn R. On reducing TLB misses in matrix multiplication. Technical report tr-2002-55, The University of Texas at Austin, Department of Computer Sciences, 2002.

[LHKK79] C. L. Lawson, R. J. Hanson, D. Kincaid, and F. T. Krogh. Basic Linear Algebra Subprograms for FORTRAN usage. In *In ACM Trans. Math. Soft., 5 (1979), pp. 308-323*, 1979.

[WD98] R. Clint Whaley and Jack J. Dongarra. Automatically tuned linear algebra software. In *Supercomputing '98: Proceedings of the 1998 ACM/IEEE conference on Supercomputing (CDROM)*, pages 1–27, Washington, DC, USA, 1998. IEEE Computer Society.

[WSO+06] Samuel Williams, John Shalf, Leonid Oliker, Shoaib Kamil, Parry Husbands, and Katherine Yelick. The potential of the cell processor for scientific computing. In *CF '06: Proceedings of the 3rd conference on Computing frontiers*, pages 9–20, New York, NY, USA, 2006. ACM.

1. Aktuelle und zukünftige Aktivitäten (Bericht des Sprechers)

Die 25. Ausgabe der PARS-Mitteilungen enthält die Beiträge des neunten PASA-Workshops, welcher die wesentlichen Aktivitäten der Fachgruppe im Jahr 2008 darstellt.

Der neunte PASA-Workshop fand am 26. Februar 2008 im Rahmen der Fachtagung ARCS 2008 in Dresden statt. Fast 50 Teilnehmer fanden sich an der Technischen Universität Dresden ein. Die zehn Vorträge deckten ein umfangreiches Themenspektrum ab. Als eingeladener Sprecher hielt Herr Dr. *Norbert Attig* einen Vortrag über den Supercomputer JUGENE (Jülich Blue Gene).

Den zum dritten Mal ausgeschriebenen und mit 500 € dotierten Nachwuchspreis hat Martin Schindewolf von der Universität Karlsruhe mit seinem Beitrag *A Generic Tool Supporting Cache Designs and Optimisation on Shared Memory Systems* gewonnen. Der zweite und dritte Preis gingen an Fabian Nowak und Emeric Kwemou, beide ebenfalls von der Universität Karlsruhe. Das Bild zeigt die Preisträger zusammen mit den Organisatoren des Workshops beim abendlichen Social-Event.

v.l.n.r.: Prof. Dr. Wolfgang E. Nagel (lokale Organisation), Martin Schindewolf (1. Preis), Fabian Nowak (2. Preis), Prof. Dr. Wolfgang Karl (als Vertreter des 3. Preisträgers), Prof. Dr. Rolf Hoffmann (Sprecher der Fachgruppe PARS)
Bild: K. D. Reinartz

Herrn Prof. Dr. Wolfgang E. Nagel (TU Dresden) sei für die lokale Organisation und die Gestaltung der Abendveranstaltung gedankt. Prof. Dr. Christian Hochberger und Prof. Dr. Rainer G. Spallek (beide TU Dresden) haben durch eine perfekte Organisation der ARCS 2008 in Dresden den Rahmen für den Erfolg des PASA-Workshops geschaffen. Herr Prof. Dr. Andreas Koch (TU Darmstadt) hat als Verantwortlicher für die Workshops die Zusammenstellung der Tagungsbeiträge übernommen. Herrn Dr.-Ing. Wolfgang Heenes (TU Darmstadt) sei für die technische Organisation des Workshops gedankt.

Im Anschluss an den PARS-Workshop fand eine Sitzung des PARS-Leitungsgremiums statt. Aus dem Leitungsgremium der Fachgruppe schied Prof. Dr. Werner Erhard (Univ. Jena) auf eigenen Wunsch aus. Neu berufen wurden Prof. Dr. Dietmar Fey (Univ. Jena), Prof. Dr. Wolfgang E. Nagel (TU Dresden) und Dr. Andreas Döring (IBM Zürich). Der Sprecher der Fachgruppe, Prof. Dr. Rolf Hoffmann und der stellvertretende Sprecher, Dr. Karl Dieter Reinartz, wurden bei der turnusmäßigen Wahl in Ihren Ämtern bestätigt.

PARS hat derzeit 270 Mitglieder, in etwa so viele wie im letzten Jahr. Die Umstellung der Webseiten auf das zentrale Typo3-System der GI ist mittlerweile erfolgt.

Unser nächster Workshop ist der

22. PARS-Workshop am 4./5. Juni 2009 in Parsberg in der Oberpfalz.

Die nächste Sitzung des PARS-Leitungsgremiums, zu dem auch interessierte PARS-Mitglieder eingeladen sind, wird am Rande des PARS-Workshops 2009 in Parsberg stattfinden.

Aktuelle Information finden Sie auch auf der PARS-Webpage

http://www.pars.gi-ev.de/

Anregungen und Beiträge für die Mitteilungen können wie üblich an den Sprecher (hoffmann@informatik.tu-darmstadt.de) gesendet werden. Besonders gedankt sei meinem Mitarbeiter Herrn Dr. Heenes, der mich in technischen und allgemeinen Verwaltungsaufgaben unterstützt. Er hat diese PARS-Mitteilungen für den Druck zusammengestellt und gestaltet u. a. den Web-Auftritt von PARS.

Ich wünsche Ihnen ein gesundes und erfolgreiches Jahr 2009.

Darmstadt, im Dezember 2008

Rolf Hoffmann
(PARS-Sprecher)

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.euro-par.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

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 (jeweils gemeinsam mit der GI-Fachgruppe 0.1.3 ‚Parallele und verteilte Algorithmen (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" (Thesenpapier 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)

4. Mitteilungen (ISSN 0177-0454)

Bisher sind 25 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 gimv@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.

GESELLSCHAFT FÜR INFORMATIK E.V.

PARALLEL-ALGORITHMEN, -RECHNERSTRUKTUREN UND -SYSTEMSOFTWARE

PARS

INFORMATIONSTECHNISCHE GESELLSCHAFT IM VDE

CALL FOR PAPERS

22. PARS - Workshop am 4./5. Juni 2009

Parsberg in der Oberpfalz

http://www.ra.informatik.tu-darmstadt.de/pars/2009/

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

- **Parallele Algorithmen (Beschreibung, Komplexität, Anwendungen)**
- **Parallele Rechenmodelle und parallele Architekturen**
- **Parallele Programmiersprachen und Bibliotheken**
- **Parallele Programmierung und Programmparallelisierung**
- **Software Engineering für parallele und verteilte Systeme**
- **Vernetzte Systeme und Anwendungen, Grid Computing, Cloud Computing**
- **Verbindungsstrukturen und Hardwareaspekte (z. B. rekonfigurierbare Systeme)**
- **Neue Technologien und Architekturen (SoC, Multicores, PIM , STM etc.)**
- **Alternative Technologien (Quantencomputing, DNA-Computing)**
- **Parallel Organic Computing**
- **Parallelverarbeitung im Unterricht (Erfahrungen, E-Learning)**
- **Parallele Methoden und verteiltes Rechnen in der Bioinformatik**

Die Sprache des Workshops ist Deutsch. Vorträge und Beiträge in Englisch sind ebenfalls willkommen. Für jeden Beitrag sind maximal 10 DIN A4 Seiten vorgesehen. Die Workshop-Beiträge werden als PARS-Mitteilungen (ISSN 0177-0454) publiziert. Es ist eine Workshopgebühr von ca. 100 € geplant.

Termine: Vortragsanmeldungen als Volltext oder in einer Kurzfassung von 2 bis 4 Seiten sind bis zum **6. März 2009** in elektronischer Form unter folgendem Link einzureichen:
http://www.ra.informatik.tu-darmstadt.de/pars/2009/papersubmission/
Benachrichtigung der Autoren bis **10. April 2009**
Druckfertige Ausarbeitungen bis **30. September 2009** (nach dem Workshop)

Programmkomitee: *H. Burkhart, Basel • A. Döring, Zürich • D. Fey, Jena • R. Hoffmann, Darmstadt • F. Hoßfeld, Jülich W. Karl, Karlsruhe • J. Keller, Hagen • Chr. Lengauer, Passau • E. Maehle, Lübeck • E. W. Mayr, München F. Meyer auf der Heide, Paderborn • W. E. Nagel, Dresden • K. D. Reinartz, Höchstadt • H. Schmeck, Karlsruhe • P. Sobe, Lübeck • T. Ungerer, Augsburg • H. Weberpals, Hamburg Harburg*

Nachwuchspreis: Der beste Beitrag, der auf einer Bachelor/Studien-, Diplom/Masterarbeit oder Dissertation basiert, und von dem Autor/der Autorin selbst vorgetragen wird, soll auf dem Workshop von der Fachgruppe PARS mit einem Preis (dotiert mit 500 €) ausgezeichnet werden.

Veranstalter: GI/ITG-Fachgruppe PARS, http://www.pars.gi-ev.de

Organisation: Prof. Dr. Wolfgang Karl, Institut für Technische Informatik
Universität Karlsruhe (TH), Kaiserstraße 12, D-76128 Karlsruhe
Tel.: 0721-608-3771, Fax: 0721-608-3962, E-Mail: karl@ira.uka.de

Prof. Dr.-Ing. Rolf Hoffmann (PARS-Sprecher), FG Rechnerarchitektur, FB Informatik,
TU Darmstadt, Hochschulstr. 10, D-64289 Darmstadt
Tel.: 06151-16-3611/3606, Fax: 06151-165410, E-Mail: hoffmann@informatik.tu-darmstadt.de

Technische Organisation: Dr.-Ing. Wolfgang Heenes, FG Rechnerarchitektur, FB Informatik, TU Darmstadt, Hochschulstr. 10, D-64289 Darmstadt, Tel.: 06151-165312, Fax: 06151-165410, E-Mail: heenes@ra.informatik.tu-darmstadt.de

Dr. Rainer Buchty, Institut für Technische Informatik, Universität Karlsruhe (TH), Kaiserstraße 12, D-76128 Karlsruhe, Tel.: 0721-608-8768, Fax: 0721-608-3962, E-Mail: buchty@ira.uka.de

22ND INTERNATIONAL CONFERENCE ON ARCHITECTURE OF COMPUTING SYSTEMS

- SYSTEM ARCHITECTURE AND ENERGY AWARENESS -

Delft, Netherlands

March 10th till 13th, 2009

http://www.ida.ing.tu-bs.de/arcs09/

CALL FOR PAPERS

Submission Deadline (extended): October 19, 2008

The ARCS series of conferences has over 30 years of tradition reporting top notch results in computer architecture and operating systems research. This year's focus is energy awareness viewed from two different points. Firstly, this deals with the improvement of computer systems to be as energy efficient as possible (particularly for specific applications). One can think of heterogeneous multi-core architectures or reconfigurable architectures for this purpose. Secondly, this addresses the usage of computer systems to reduce the energy consumption of other systems, which might lead to problems of communication and cooperation. Both aspects are relevant for the conference. Like the previous conferences in this series, it continues to be an important forum for computer architecture research. In 2009, ARCS will be hosted by the Delft University of Technology, which has one of the leading information technology schools in Europe. The proceedings of ARCS 2009 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 a best presentation award will be presented at the conference.

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

- Energy-awareness, green computing.
- Computer architecture topics such as multi-cores, memory systems, and parallel computing.
- Adaptive system architectures such as reconfigurable systems in hardware and software.
- Operating Systems including but not limited to scheduling, memory management, power management, and RTOS.
- System aspects of ubiquitous and pervasive computing such as sensor nodes, novel input/output devices, novel computing platforms, architecture modeling, and middleware.
- Organic and Autonomic Computing including both theoretical and practical results on self-organization, self-configuration, self-optimization, self-healing, and self-protection techniques.
- Embedded systems including but not limited to architecture, communication, design methodologies, and applications.
- Network Centric and Grid Computing issues with a focus on middleware.

Submissions should be done through the link provided at the conference website http://www.ida.ing.tu-bs.de/arcs09/. Papers should be submitted in pdf or postscript format. They should be formatted according to Springer LNCS style (see: http://www.springer.de/comp/lncs/authors.html) and not exceed 12 pages.

Workshop and Tutorial Proposals: Proposals for workshops and tutorials within the technical scope of the conference are solicited. Submissions should be done through email directly to the workshops and tutorials chair Jörg Hähner (haehner@sra.uni-hannover.de).

Important Dates

Paper submission deadline:	19th October 2008
Notification of acceptance:	28th November 2008
Camera ready papers:	12th December 2008

Organizing Committee

General Chairs:
Mladen Berekovic , TU Braunschweig, DE
Christian Müller-Schloer, University of Hannover, DE

PC Chairs:
Christian Hochberger, Technische Universität Dresden, DE
Stephan Wong, Delft University of Technology, NL

Workshop and Tutorial Chair:
Jörg Hähner, University of Hannover, DE

Program Committee

Wael Adi, TU Braunschweig, DE
Tughrul Arslan, University of Edinburgh, UK
Nader Bagherzadeh, University of California Irvine, US
Michael Beigl, TU Braunschweig, DE
Guillem Bernat, Rapita Systems and University of York, UK
Arndt Bode, TU Munich, DE
Koen De Bosschere, Ghent University, BE
Uwe Brinkschulte, University of Karlsruhe, DE
Jiannong Cao, The Hong Kong Polytechnic University, CN
Joao Cardoso, NESC-ID, Lisboa, Portugal, PT
Luigi Carro, UFRGS, BR
Henk Corporaal, TU-Eindhoven, NL
Francisco J. Cazorla, UPC, ES
Steven Derrien, INRIA-Rennes, FR
Nikitas Dimopoulos, University of Victoria, CA
Alois Ferscha, University of Linz, AT
Fabrizio Ferrandi, Polimi, IT
Bjorn Franke, Edinburgh, UK
Werner Grass, University of Passau, DE
Soonhoi Ha, Seoul National University, KR
Andreas Herkersdorf, TU München, DE
Seongsoo Hong, Seoul National University, KR
Paolo Ienne, EPFL Lausanne, Switzerland
Tohru Ishihara, Kyushu University, JP
Jadwiga Indulska, University of Queensland, AU
Murali Jayapala, IMEC, BE
Gert Jervan, Tallin University of Technology, EE
Ben Juurlink, TU-Delft, NL
Wolfgang Karl, University of Karlsruhe, DE
Manolis Katevenis, FORTH and Univ. of Crete, Greece
Andreas Koch, TU Darmstadt, DE
Krzysztof Kuchcinski, Lund University, SE
Spyros Lalis, University of Thessaly, GR
Paul Lukowicz, University of Passau, DE
Jianhua Ma, Hosei University, JA
Erik Maehle, Universität zu Lübeck, DE
Jan Madsen, Technical University of Denmark, DK
Tom Martin, Virginia Tech, US
Peter Marwedel, University of Dortmund, DE
Dragomir Milojevic, ULB Brussel, BE
Nacho Navarro, UPC, ES
Alex Orailoglu, UCSD, US
Emre Ozer, ARM, UK
Andy Pimentel, University of Amsterdam, NL
Burghardt Schallenberger, Siemens AG, DE
Pascal Sainrat, Université Paul Sabatier, Toulouse, FR
Yiannakis Sazeides, University of Cyprus, CY
Hartmut Schmeck, University of Karlsruhe, DE
Karsten Schwan, Georgia Tech, US
Gerard Smit, University of Twente, NL
Leonel Sousa, TU Lisbon, PT
Rainer G. Spallek, TU Dresden, DE
Peter Steenkiste, Carnegie-Mellon University, US
Bassel Soudan, University of Sharjah, AE
Jarmo Takala, Tampere University of Technology, FI
Jürgen Teich, Universität Erlangen, DE
Lothar Thiele, ETH Zurich, CH
David Thomas, Imperial College London, UK
Pedro Trancoso, University of Cyprus, CY
Gerhard Tröster, ETH Zurich, CH
Theo Ungerer, University of Augsburg, DE
Mateo Valero, UPC, ES
Stephane Vialle, Supelec, FR
Lucian Vintan, Lucian Blaga University of Sibiu, RO
Klaus Waldschmidt, University of Frankfurt, DE
Laurence T. Yang, St Francis Xavier University, CA
Sami Yehia, Thales Group, FR

The 15th International European Conference on Parallel and Distributed Computing (Euro-Par 2009)

http://europar2009.ewi.tudelft.nl

europar2009@tudelft.nl

Delft University of Technology, Delft, The Netherlands

August 25-28, 2009

SUBMISSION DEADLINES:

Abstracts: January 24, 2009

Full papers: January 31, 2009

Euro-Par is an annual series of international conferences dedicated to the promotion and advancement of all aspects of parallel and distributed computing.

Euro-Par focuses on all aspects of hardware, software, algorithms and applications in this field. The objective of Euro-Par is to provide a forum for the promotion of parallel and distributed computing both as an industrial technique and an academic discipline, extending the frontier of both the state of the art and the state of the practice.

The following topics will be covered by regular Euro-Par 2009 sessions:

1. Support tools and environments
2. Performance prediction and evaluation
3. Scheduling and load balancing
4. High performance architectures and compilers
5. Parallel and distributed databases
6. Grid, cluster, and cloud computing
7. Peer to peer computing
8. Distributed systems and algorithms
9. Parallel and distributed programming
10. Parallel numerical algorithms
11. Multicore and manycore programming
12. Theory and algorithms for parallel computation
13. High performance networks
14. Mobile and ubiquitous computing

Full details on the topics, including topic descriptions and chairs, are available on the Euro-Par 2009 website (http://europar2009.ewi.tudelft.nl). The conference will feature tutorials and invited talks. Co-located workshops are also planned.

Paper Submission guidelines

Full papers should not exceed 10 pages in the Springer LNCS style. Paper submission will be performed electronically via the conference website in PDF format. Papers accepted for publication must also be supplied in source form (LaTeX or Word).

Papers must offer original contributions regarding the theory and practice of parallel and distributed computing. Full submission guidelines are available on the conference website. Only contributions not submitted elsewhere for publication will be considered.

Authors must submit their papers to the specific topic they judge most appropriate. Details on topics, including descriptions and chairs, are available on the conference website.

All accepted papers will be included in the conference proceedings, published by Springer in the LNCS series. Authors of accepted papers will be requested to sign a Springer copyright form.

Important Dates

Abstracts due	January 24, 2009
Full papers due	January 31, 2009
Decision notification	May 2, 2009
Camera-ready full papers	May 25, 2009

Call for Additional Conference Workshops

Euro-Par 2009 will feature a series of satellite workshops on August 24-25.

Proposals for workshops, covering a specific theme and lasting between half a day and a full day, are encouraged and solicited until April 1, 2009. Collective workshop proceedings are published in a separate LNCS Euro-Par 2009 workshop volume after the conference. The principal coordinator of each workshop will appear as editor of the workshop volume.

Please contact the Conference Chairs (europar2009@tudelft.nl) for additional details and proposals.

Venue

The Euro-Par 2009 conference will be held on the campus of Delft University of Technology, which was founded in 1842 by King William II and which is the oldest and largest Technical University in the Netherlands. It is well established as one of the leading technical universities in the world.

Delft is a small, historical town dating back to the 13th century. Delft has many old buildings and small canals and has a lively atmosphere. The city offers a large variety of hotels and restaurants. Many other places of interest (e.g., Amsterdam and The Hague) are within one hour distance of traveling.

Traveling to Delft is easy. Delft is close to Amsterdam Schiphol Airport (60 km, 45 minutes by train), which has direct connections from all major airports worldwide. Delft also has excellent train connections to the rest of Europe.

Conference Co-Chairs

- Henk Sips, TU Delft, General Chair
- Dick Epema, TU Delft, Program Chair
- Hai Xiang Lin, TU Delft, Workshop Chair

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. Jena
Prof. Dr. Rolf Hoffmann, Sprecher, TU Darmstadt
Prof. Dr. Wolfgang Karl, Univ. Karlsruhe
Prof. Dr. Jörg Keller, FernUniversität Hagen
Prof. Dr. Christian Lengauer, Univ. Passau
Prof. Dr.-Ing. Erik Maehle, Universität zu Lübeck
Prof. Dr. Ernst W. Mayr, TU München
Prof. Dr. Wolfgang E. Nagel, TU Dresden
Dr. Karl Dieter Reinartz, stellv. Sprecher, Univ. Erlangen-Nürnberg
Prof. Dr. Hartmut Schmeck, Univ. Karlsruhe
Prof. Dr. Theo Ungerer, Univ. Augsburg
Prof. Dr. Helmut Weberpals, TU Hamburg Harburg

Sprecher

Prof. Dr.-Ing. Rolf Hoffmann
Technische Universität Darmstadt
Fachgebiet Rechnerarchitektur
Hochschulstraße 10
D-64289 Darmstadt
Tel.: (06151) 16-3611/3606
Fax: (06151) 165410
E-Mail: hoffmann@informatik.tu-darmstadt.de
URL: http://www.pars.gi-ev.de/

PARS
Einladung zur Mitarbeit
in der GI/ITG-Fachgruppe

✂.....

Gesellschaft für Informarik e. V., Tel. (0228) 302145, FAX (0228) 302167, E-Mail: gs@gi-ev.de

☐ GI-Mitglied Nr. .....

☐ ITG-Mitglied Nr. .....

☐ .....
Sonstige

☐ Ich beantrage die Mitgliedschaft in der Fachgruppe PARS
‚Parallel-Algorithmen, -Rechnerstrukturen
und Systemsoftware'.
Ich bin bereit, meinen entsprechenden PARS-Beitrag
an die Geschäftsstelle der GI zu entrichten.

Gesellschaft für Informatik e.V.
Wissenschaftszentrum
Ahrstr. 45

D-53175 Bonn

FRG

.....
Titel, Name, Vorname

.....
Firma/Diensstelle

.....
Straße

.....
PLZ, Ort, Land

.....
Telefon, Datum, Unterschrift

.....
Telefax. Email-Adresse

ITG PARS '08

25