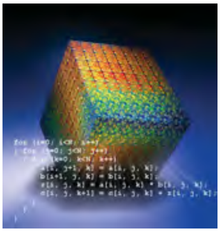
Sonderforschungsbereich/Transregio 89
Transregional Collaborative Research Center 89

Invasive Computing
Annual Report 2011



TRR 89

Friedrich-Alexander-Universität Erlangen-Nürnberg
Karlsruher Institut für Technologie
Technische Universität München

Transregional Collaborative Research Centre 89

Invasive Computing

Friedrich-Alexander-Universität Erlangen-Nürnberg
Karlsruher Institut für Technologie
Technische Universität München

Annual Report 2011
July 2010 - December 2011

Coordinator
Prof. Dr.-Ing. Jürgen Teich
Lehrstuhl für Informatik 12
Universität Erlangen-Nürnberg
Cauerstraße 11, 91058 Erlangen

Managing Director
Dr.-Ing. Jürgen Kleinöder
Lehrstuhl für Informatik 4
Universität Erlangen-Nürnberg
Martensstr. 1, 91058 Erlangen

Public Relations
Dr. rer. nat. Katja Lohmann
Lehrstuhl für Informatik 12
Universität Erlangen-Nürnberg
Cauerstraße 11, 91058 Erlangen

Preface

This report summarises the activities and scientific progress of the Transregional Collaborative Research Centre 89 “Invasive Computing” (InvasIC) in the first 18 months since its establishment in July 2010.

The main idea of InvasIC is to develop and investigate a completely novel paradigm for designing and programming future parallel computing systems. To support these ideas of self-adaptive and resource-aware programming, not only new programming concepts, languages, compilers and operating systems are necessary but also revolutionary architectural changes in the design of Multi-Processor Systems-on-a-Chip (MPSoCs).

InvasIC is funded by the Deutsche Forschungsgemeinschaft for initially four years and aggregates 58 of the best researchers from three excellent sites in Germany (Friedrich-Alexander-Universität Erlangen-Nürnberg, Karlsruher Institut für Technologie, Technische Universiät München). This scientific team includes specialists in algorithm engineering for parallel algorithm design, hardware architects for reconfigurable MPSoC development as well as language, tool and application, and operating system designers.

This report gives an overview of the current transregional activities of InvasIC and summarises the scientific progress made in the projects and working groups. Most important – not only, but especially in the InvasIC starting phase – were meetings and workshops bringing together scientists from the three sites. In the meantime, several meetings took place and periodical project and working group meetings as well as the regular plenary meetings were established.

A highlight – especially for the doctoral researchers – was the first InvasIC Doctoral Researchers’ Retreat (DRR), held as a continuation of the annual meeting from Oct. 5–7, 2011. The participants considered the retreat as very productive and helpful for their daily work so that they suggested to implement a semi-annual schedule for the DRR.

We hope that you will find the following report interesting and stimulating. For a complete overview of our goals and current activities please visit our website `http://www.invasic.de`.

Jürgen Teich
Coordinator

Contents

I

Invasive Computing

1 About InvasIC

The Idea of Invasive Computing

The main idea and novelty of *invasive computing* is to introduce resource-aware programming support in the sense that a given program gets the ability to explore and dynamically spread its computations to neighbour processors similar to a phase of invasion, then to execute portions of code of high parallelism degree in parallel based on the available (invasible) region on a given multi-processor architecture. Afterwards, once the program terminates or if the degree of parallelism should be lower again, the program may enter a *retreat* phase, deallocate resources and resume execution again, for example, sequentially on a single processor. To support this idea of self-adaptive and resource-aware programming new programming concepts, languages, compilers and operating systems are necessary as well as architectural changes in the design of MPSoCs (Multi-Processor Systems-on-a-Chip) to efficiently support invasion, infection and retreat operations by involving concepts for dynamic processor, interconnect and memory reconfiguration.

Decreasing feature sizes have already led to a rethinking in the design of multi-million transistor system-on-chip (SoC) architectures, envisioning dramatically increasing rates of temporary and permanent faults and feature variations. The major question will be how to deal with this imperfect world in which components will become more and more unreliable. As we can foresee SoCs with 1000 or more processors on a single chip in the year 2020, static and central management concepts to control the execution of all resources might have met their limits long before and are therefore not appropriate. Invasion might provide the required *self-organising* behaviour to conventional programs for being able to provide scalability, higher resource utilisation numbers and, hopefully, also performance gains by adjusting the amount of allocated resources to the temporal needs of a running application. This thought opens a new way of thinking about *parallel algorithm design*. Based on algorithms utilising invasion and negotiating resources with others, we can imagine that corresponding programs become *personalised* objects, competing with other applications running simultaneously on an MPSoC.

Scientific Goals of InvasIC

In the Transregional Collaborative Research Center InvasIC, we provide the necessary extensions to support invasive programming efficiently by introducing and investigating new concepts of dynamic and *resource-aware programming* as well as required new architectural processor, interconnect and memory structures providing the necessary extensions to support invasive programming efficiently – in particular through concepts of dynamic hardware reconfiguration. Legacy programs shall still be executable within an invasive computing architecture, thus we will show a migration path from traditional programming to the new invasive programming paradigm.

We believe that only a Transregional Collaborative Research Center aggregating the best researchers from three excellent sites in Germany will allow us to investigate these revolutionary ideas. Our InvasIC researching team includes specialists in algorithm engineering for parallel algorithm design, hardware architects for reconfigurable MPSoC development as well as language, tool and application and operating system designers. Invasive computing is the central theme of InvasIC that can be found in each project. A carefully worked out validation concept with an FPGA-based demonstrator has been proposed in order to fully demonstrate the benefits of invasive computing quantitatively.

2 Participating University Groups

Friedrich-Alexander-Universität Erlangen-Nürnberg

Lehrstuhl für Hardware-Software-Co-Design

- Prof. Dr.-Ing. Jürgen Teich
- Dr.-Ing. Frank Hannig

Lehrstuhl für Verteilte Systeme und Betriebssysteme

- Prof. Dr.-Ing. Wolfgang Schröder-Preikschat
- Dr.-Ing. Daniel Lohmann

Karlsruher Institut für Technologie

Institut für Anthropomatik

- Prof. Dr.-Ing. Rüdiger Dillmann
- Dr.-Ing. Tamim Asfour

Institut für Technik der Informationsverarbeitung

- Prof. Dr.-Ing. Jürgen Becker
- Dr.-Ing. Michael Hübner

Institut für Programmstrukturen und Datenorganisation

- Prof. Dr.-Ing. Gregor Snelting

Institut für Technische Informatik

- Prof. Dr. Jörg Henkel
- Dr.-Ing. Lars Bauer

Institut für Theoretische Informatik

- Prof. Dr. Peter Sanders

Technische Universität München

Lehrstuhl für Integrierte Systeme

- Prof. Dr. Andreas Herkersdorf
- Prof. Dr.-Ing. Walter Stechele
- Dr.-Ing. Thomas Wild

Lehrstuhl für Wissenschaftliches Rechnen

- Prof. Dr. Hans-Joachim Bungartz

Lehrstuhl für Rechnertechnik und Rechnerorganisation

- Prof. Dr. Michael Gerndt

Lehrstuhl für Entwurfsautomatisierung

- Prof. Dr.-Ing. Ulf Schlichtmann

Lehrstuhl für Technische Elektronik

- Prof. Dr. Doris Schmitt-Landsiedel

II

Research Program

3 Overview of Research Program

To investigate the novel paradigms for designing and programming future parallel computing systems the InvasIC is organised in 5 project areas:

Area A: Fundamentals, Language and Algorithm Research

Research in project area A focuses on the basic concepts of invasion and resource-aware programming as well as on language issues, algorithmic theory of invasion and on load balancing and scheduling.

B:Architectural Research

Project area B investigates micro- and macroarchitectural requirements, techniques and hardware concepts to enable invasive computing in future MPSoCs.

C: Compiler, Simulation and Run-Time Support

The focus of project area C is on software support for invasive computing including compiler, simulation and operating system functionality with a special focus on run-time management.

D: Applications

Applications serve as demonstrators for the diverse and efficient deployment of invasive computing. Therefore, the applications have been chosen carefully from the domains of robotics and scientific computing in order to demonstrate distinct and complementary features of invasive computing.

Z2: Validation and Demonstrator

A hardware demonstrator will serve as the concept validation platform for invasive computing. It will allow for co-validation and demonstration of invasive computing through tight integration of hardware and software research results at the end of the first project phase and to decide on the further roadmap of specific hardware for invasive computing.

The three working groups **Language and Applications**, **System Software** and **Architectures** defined on top of these project areas support the interdisciplinary research.

Research Area	Project	
A: Fundamentals, Language and Algorithm Research	Basics of Invasive Computing *Prof. Dr.-Ing. Jürgen Teich, Prof. Dr.-Ing. Gregor Snelting*	A1
	Scheduling and Load Balancing *Prof. Dr. Peter Sanders*	A3
B:Architectural Research	Adaptive Application-Specific Invasive Microarchitecture *Prof. Dr. Jörg Henkel, Dr.-Ing. Michael Hübner, Dr.-Ing. Lars Bauer*	B1
	Invasive Tightly-Coupled Processor Arrays *Prof. Dr.-Ing. Jürgen Teich*	B2
	Invasive Loosely-Coupled MPSoCs *Prof. Dr. Andreas Herkersdorf, Prof. Dr. Jörg Henkel*	B3
	Hardware Monitoring System and Design Optimisation for Invasive Architectures *Prof. Dr. Doris Schmitt-Landsiedel, Prof. Dr.-Ing. Ulf Schlichtmann*	B4
	Invasive NoCs — Autonomous, Self-Optimising Communication Infrastructures for MPSoCs *Prof. Dr.-Ing. Jürgen Becker, Prof. Dr. Andreas Herkersdorf, Prof. Dr.-Ing. Jürgen Teich*	B5
C: Compiler, Simulation and Run-Time Support	Invasive Run-Time Support System (iRTSS) *Prof. Dr.-Ing. Wolfgang Schröder-Preikschat, Dr.-Ing. Daniel Lohmann, Prof. Dr. Jörg Henkel, Dr.-Ing. Lars Bauer*	C1
	Simulation of Invasive Applications and Invasive Architectures *Dr.-Ing. Frank Hannig, Prof. Dr. Michael Gerndt, Prof. Dr. Andreas Herkersdorf*	C2
	Compilation and Code Generation for Invasive Programs *Prof. Dr.-Ing. Gregor Snelting, Prof. Dr.-Ing. Jürgen Teich*	C3
D: Applications	Invasive Software–Hardware Architectures for Robotics *Prof. Dr.-Ing. Rüdiger Dillmann, Dr.-Ing. Tamim Asfour, Prof. Dr.-Ing. Walter Stechele*	D1
	Multilevel Approaches and Adaptivity in Scientific Computing *Prof. Dr. Hans-Joachim Bungartz, Prof. Dr. Michael Gerndt*	D3
Z: Administration	Central Services *Prof. Dr.-Ing. Jürgen Teich, Dr.-Ing. Jürgen Kleinöder, Dr. Katja Lohmann*	Z
	Validation and Demonstrator *Prof. Dr.-Ing. Jürgen Becker, Dr.-Ing. Frank Hannig, Dr.-Ing. Thomas Wild*	Z2
WG: Working Groups	Working Group Architecture *Prof. Dr. Andreas Herkersdorf*	WG1
	Working Group System Software *Prof. Dr.-Ing. Wolfgang Schröder-Preikschat*	WG2
	Working Group Language and Applications *Prof. Dr.-Ing. Gregor Snelting*	WG3

4 Research Projects

A1: Basics of Invasive Computing

Jürgen Teich, Gregor Snelting

Andreas Weichslgartner, Andreas Zwinkau

We investigate the basics of invasion – the fundamental programming model for invasive and resource-aware computation. This includes a) the functionality requirements and b) the development of a mathematical model for quantitative performance, resource efficiency (utilisation) and overhead analysis for different types of invasive architecture targets. We also research a programming notation for invasive programming: Here, c) the syntax, semantics and type system of an abstract invasive core language is defined, which is the basis for d) the definition of a concrete invasive language and its system interfaces. The goal of Project A1 is not to build a new programming language from the scratch, but to make the invasive programming model usable for the programmer. For this purpose, the abstract invasive constructs have been embedded as a concrete language in the parallel computing language X10 from IBM.

Concrete Language Design

The concrete language must exploit modern language theory, respect the above-mentioned abstraction levels, and provide interfaces to dynamic resource parameters as well as to existing technology for parallel programming. It should utilise generic mechanisms, instead of introducing lots of low-level, adhoc syntactic constructs. The concrete language must be exercised on real algorithms and problems. Expressiveness and usability must be evaluated for all different types of potential target architectures ranging from tightly- over loosely-coupled MPSoC architectures to high-performance computing (HPC) machines.

A language developed in isolation is unlikely to meet the needs of actual programmers of a diverse project, like InvasIC, which includes hardware and software design. So we used an iterative process, where everybody could participate to identify required functionality for each project. For the first Language Working Group Workshop on December 10th, 2010 in Karlsruhe, each project produced invasive pseudo code examples, which demonstrated their vision. Project A1 integrated this data in a cohesive framework called invadeX10.

For an example, consider the following program fragment, which shows the three basic constructs invade, infect and retreat:

```
val claim = Claim.invade( constraints );
claim.infect( ilet );
claim.retreat();
```

The static method `Claim.invade` takes constraints and returns a claim object, which represents the allocated resources, typically a set of processing elements (PEs). A claim provides an infection method to distribute computations across the PEs. The argument of `infect` is an `ilet` object, which contains the code to execute together with initial data. The `infect` call blocks, until all `ilet` computations finish. Afterwards, the `retreat` method frees all resources within a claim, such that the claim is empty. While such a claim can still be infected, this would do nothing. Now consider the `ilet` variable of the example above. A hello-world `ilet` could be declared as follows:

```
val ilet = (id: IncarnationID ) => {
  Console.OUT.println("Hello! ("+id+")");
};
```

Since the invadeX10 framework provides a concrete X10 API, the pseudo code examples may then be translated into actual X10 code, which was presented at the second Language Working Group Workshop in July 4th, 2011 in Karlsruhe. This API provides a common basis on the language level. Project C2 currently develops a functional simulation implementation of the invasive API, it is possible to execute invasive applications like those developed within the project area D. The interplay of concrete invasive language and functional simulation was published in [HRST+11].

Constraints

While the API demonstrated in the example above is very generic, the `constraints` variable provides access to specific hardware and operating system features. To avoid a hodgepodge of syntactic constructs the constraints were modelled as an extensible class hierarchy. The set of constraints can be classified as follows.

The first class of constraints we identified were so-called predicate constraints, which specify predicates for processing elements to be claimed. An application might require the demanded processing elements to (1) be under a certain load, (2) be under a certain temperature, (3) have an FPU, (4) have certain amount of local memory, (5) have a scratch pad memory, (6) be of a certain type, (7) have a certain cache size, (8) be migratable, or (9) not be scheduled preemptively. Such constraints impose a simple filter operation over the set of available processing elements within an invade operation. The second class of constraints are order constraints, which provide an ordering of processing elements according to (1) load, (2) temperature, (3) memory, or (4) speed. Using these constraints an application can communicate its preferences, whether it is IO- or CPU-bound. By giving multiple of these constraints, the programmer can impose a secondary or tertiary ordering.

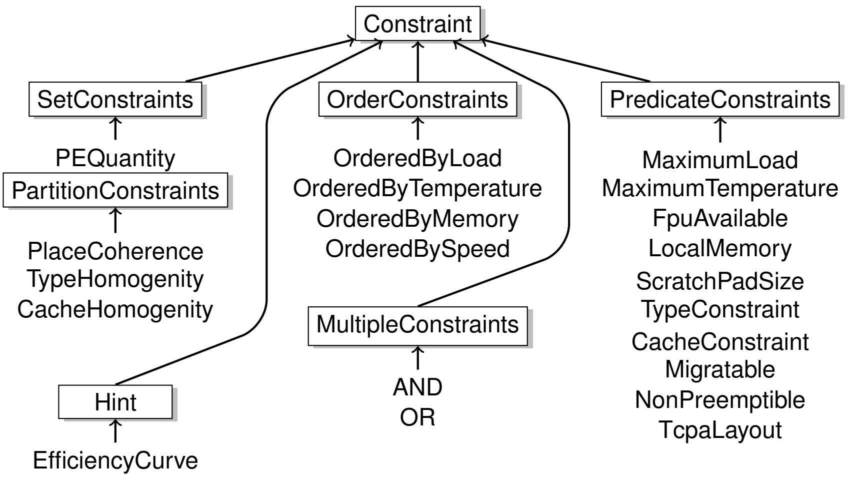


Figure 4.1: Constraint class hierarchy

The third class of constraints are set constraints, as they specify conditions for a set of processing elements as a whole. The most common constraint is the (1) quantity of processing elements to be claimed, but there are also partition constraints, (2) a certain physical layout of the PEs, (3) place coherence, which means that the PEs have shared memory, (4) type homogeneity, in terms of the instruction set architecture,

(5) cache type homogeneity. Additionally, there are the two operators AND and OR to combine constraints. At last, the programmer can give so-called nonbinding hints, which can be used to hand complex information like efficiency curves of parameters to the run-time system of the underlying MPSoC architecture. These constraints are implemented as a class hierarchy as shown in Figure 4.1, which is available to the programmer. The constraints above form the leafs of the hierarchy tree and abstract classes, drawn as boxes, partition the tree into categories.

Overhead Analysis

When considering benefits of invasive programing, also a mathematical model for performance and overhead analysis is a must. Dynamic decisions about invasion will depend on data distribution, load balancing and the dynamic state of resources. The definition of corresponding overhead functions for resource utilisation and efficiency is important to show if invasive programming allows to achieve utilisation close to 100%. Such overhead functions can be used in the design of invasive algorithms and help to dynamically decide if and where invasion should happen.

To achieve these goals, Project A1 worked on an overhead calculus and obtained first results for the many-core architectures Intel's 48-core Single-Chip Cloud Computer (SCC) and the Tilera's 64-core TILEPro64. These results were based on experimental evaluations and were published in [MTKB+11] and [KHLT11]. Also investigations of the invasive overheads for invasion in TCPAs (cooperation with Project B2) and in single RISC tiles, with a shared memory and a minimal implementation of the invasive core primitives, are ongoing. Until now, heterogeneous targets and complex communications structures are not considered, so we will focus on these challenges in cooperation with the Project B5 in the next year.

Outreach

For the Summer School of HiPEAC 2010 in Autrans (Grenoble), Teich held a tutorial on invasive computing [Tei10]. He also presented invasive computing at ParLab (UC Berkeley) [Tei11a] and PPL (Stanford) [Tei11b]. An overview book chapter [THHSL+11] on the goals of the TCRC was also published.

Publications

[HRST+11] F. Hannig, S. Roloff, G. Snelting, J. Teich, and A. Zwinkau. “Resource-Aware Programming and Simulation of MPSoC Architectures through Extension of X10”. In: *Proceedings of the 14th International Workshop on Software and Compilers for Embedded Systems (SCOPES)*. St. Goar, Germany: ACM Press, June 27–28, 2011, pp. 48–55. ISBN: 978-1-4503-0763-5. DOI: 10.1145/1988932.1988941.

[KHLT11] G. Kouveli, F. Hannig, J. Lupp, and J. Teich. “Towards Resource-Aware Programming on Intel’s Single-Chip Cloud Computer Processor”. In: *3rd Many-core Applications Research Community (MARC) Symposium*. Vol. 7598. KIT Scientific Reports. Ettlingen, Germany: KIT Scientific Publishing, July 5–6, 2011, pp. 111–114. ISBN: 978-3-86644-717-2.

[MTKB+11] P. Marwedel, J. Teich, G. Kouveli, I. Bacivarov, L. Thiele, S. Ha, C. Lee, Q. Xu, and L. Huang. “Mapping of Applications to MPSoCs”. In: *Proceedings of the IEEE International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*. Taipei, Taiwan, Oct. 9–14, 2011, pp. 109–118.

[Tei10] J. Teich. “Invasive Computing – Basic Concepts and Foreseen Benefits”. Artist Network of Excellence on Embedded System Design Summer School Europe 2010, Autrans, France, Invited Tutorial. Sept. 7, 2010.

[Tei11a] J. Teich. “Invasive Parallel Computing – An Introduction”. Par Lab and AMP Lab Seminar Talk, UC Berkeley, CA, USA. July 22, 2011.

[Tei11b] J. Teich. “Programming Invasively Parallel – An Introduction”. Pervasive Parallelism Laboratory (PPL) Seminar Talk, Stanford University, CA, USA. July 25, 2011.

[THHSL+11] J. Teich, J. Henkel, A. Herkersdorf, D. Schmitt-Landsiedel, W. Schröder-Preikschat, and G. Snelting. “Invasive Computing: An Overview”. In: *Multiprocessor System-on-Chip – Hardware Design and Tool Integration*. Ed. by M. Hübner and J. Becker. Springer, Berlin, Heidelberg, 2011, pp. 241–268.

A3: Scheduling and Load Balancing

Peter Sanders

Jochen Speck

Newly introduced computer systems usually have CPUs with many cores and many jobs can be executed in parallel. Usually there is more than one job executing in parallel on one machine. Lots of different measuring devices are implemented in hard- and software in order to get a picture of the system status. To use this information to efficiently utilise the computing power of the parallel cores is an important task usually called scheduling and load balancing.

The research goal of this project is to provide theoretically and practically efficient scheduling algorithms for the parallel and flexible invasive systems. We are also working on fundamental algorithms for the invasive systems.

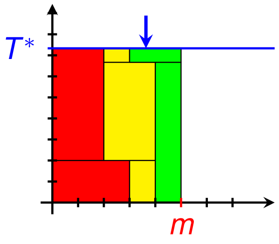


Figure 4.2: Energy Efficient Malleable Schedule

For the first step we considered an easier scheduling problem – the scheduling of malleable tasks where only computing resources are considered. A task is called malleable if it is possible to change the number of cores on which the task runs during its run time. In [SS11] we sped up an existing algorithm for the scheduling of malleable tasks and gave a parallel scheduling algorithm for the same problem. Currently we are working on the inclusion of energy efficiency into our solution of the scheduling problem of malleable tasks. In the energy efficient setting the tasks are malleable and additionally the speed of execution of the cores can be changed. Additionally we have a time interval in which all tasks have to be processed. Our goal is to minimise the energy used for the processing of all tasks in this time interval. A core which works with a higher clock frequency uses more energy. If the power consumption grows superlinearly in the speedup of a core one can save energy if one parallelises a task on more slower cores which process the task in the same time interval. So the challenge in this setting is to distribute the resources (cores) among jobs in order to save energy.

We are considering energy efficiency as important for computer architectures in the future. So more work on energy efficient scheduling is planned in the future.

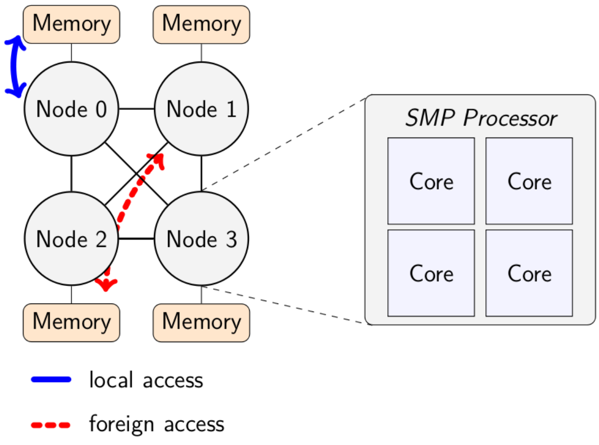


Figure 4.3: NUMA-System

In a joint work with SAP we considered modelling problems and NUMA-awareness[1]. NUMA means non-uniform-memory-architecture (see figure 4.3) NUMA-systems are systems where the memory access time depends on the memory location relative to a processor. Hence it is important to do work on a processor for which most of the needed data is available on a local memory. We did this based on a real-life problem which was the scheduling of queries in a database system. The problem comes from databases of middle sized companies where one NUMA-system runs the entire database of the company and the

[1]See also the diploma thesis of Jochen Seidel: `http://algo2.iti.kit.edu/1883.php`

complete database can be stored in the main memory of that machine. Usually there are several employees working on the database so usually there are several database queries running at the same time. The NUMA-awareness brought an improvement in query time. For the modelling we considered a two resource problem with the resources computing power and memory bandwidth. In our point of view the problems encountered in this work are very similar to problems which will arise in Invasive Computing.

A further important area is the scheduling of task-DAGs (DAG = directed acyclic graph). Many applications can be modelled as small task with dependencies. Especially numerical applications from Project D3 fit into this model. Here we work together with Michael Bader and others from Project D3. Additionally we are working on the parallelization of some basic dense linear algebra algorithms.

As a first example of fundamental algorithms we developed a malleable sorter which is a flexible sorting algorithm that can handle changes in the number of assigned processors very well. On a typical system with many parallel applications there always is an amount of unused resources but this amount changes frequently. This application will be able to use the idle resources left unused by other applications. Currently this sorter is slower than the fastest parallel sorters from the MCSTL if run on a system without other tasks but it shows an improvement if some cores are blocked. Our goal is to be nearly as fast as the MCSTL sorters in all circumstances and to be much faster in a scenario where the number of available processors changes very frequently. This application raised some interest among researchers of Project C1 and will lead to a collaboration.

Our next steps are to finish the malleable sorter and some of the base dense linear algebra algorithms. We will also further try to modell a real system. Together with Project D3 we plan to work on scheduling problems arising with the *i*OpenMP.

Publications

[SS11] P. Sanders and J. Speck. "Efficient Parallel Scheduling of Malleable Tasks". In: *International Parallel and Distributed Processing Symposium (IPDPS)*. Anchorage, AL, USA: IEEE Computer Society, 2011, pp. 1156–1166. DOI: 10.1109/IPDPS.2011.110.

B1: Adaptive Application-Specific Invasive Microarchitecture

Jörg Henkel, Michael Hübner, Lars Bauer

Lars Braun, Artjom Grudnitsky, Florian Thoma

Current trends in many-core architectures go towards heterogeneity at core-level to allow efficient use of resources by applications with differing computing requirements. The invasive architecture [HHBW+12] is a many-core platform, designed to support software systems that manage computing resources dynamically.

The goal of this project is the research of concepts for an adaptive Microarchitecture (μArch) and Instruction Set Extension (ISE), culminating in the design of a processing element that is reconfigurable at run time– the *i*-Core, shown in Figure 4.4, and its integration into the invasive hardware architecture. In particular, we focus on applying the invasive paradigm on the hardware resources of the *i*-Core itself, thus allowing applications to invade the different components of the *i*-Core.

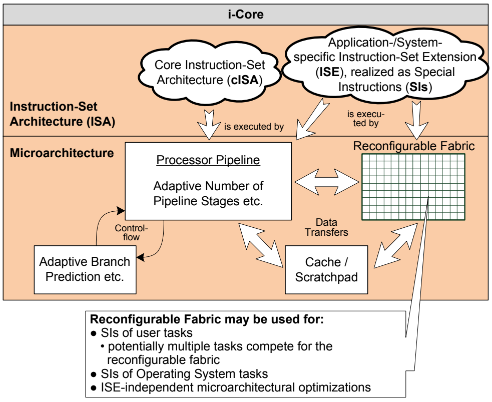

i-Core
Core Instruction-Set Architecture (cISA)
Application-/System-specific Instruction-Set Extension (ISE), realized as Special Instructions (SIs)
Instruction-Set Architecture (ISA)
is executed by
is executed by
Microarchitecture
Processor Pipeline
Adaptive Number of Pipeline Stages etc.
Reconfigurable Fabric
Data Transfers
Control-flow
Adaptive Branch Prediction etc.
Cache / Scratchpad
Reconfigurable Fabric may be used for:
• SIs of user tasks
• potentially multiple tasks compete for the reconfigurable fabric
• SIs of Operating System tasks
• ISE-independent microarchitectural optimizations

Figure 4.4: *i*-Core Architecture

A common technique for performance improvement of microarchitectures is pipelining. Splitting instructions in smaller and smaller operations allows higher clock frequencies resulting in higher throughput. However, pipelining requires discarding of partially executed instructions in case of branches and other control hazards. This fundamental trade-off between throughput and latency is normally decided at design time. The adaptive μArchof the *i*-Core overcomes this problem with the concept of dynamically fusing and splitting pipeline stages during run-time under control of the executed application.

We have modelled this concept in the architecture description language LISA for evaluation purposes. First step was the modelling of the integer pipeline of the basic LEON processor. This LEON model was also provided to Project C2 for evaluation of integration into the simulator. In a second step this model was extended with the required instructions and infrastructure for pipeline fusion. Additionally, it allows to choose between two simple branch prediction strategies and to configure some cache strategies. Benchmarking with different applications showed an improvement of up to 10% but mostly between 4% and 7% in terms of cycle counts over the unmodified seven stage pipeline. Execution of quicksort as benchmark is displayed in Figure 4.5 as an example and demonstrates the influence of branch prediction, write allocation, and write back strategies on performance. These promising results have been presented in [HTGB+11].

As a preview for a possible ASIC implementation of *i*-Core the concept of virtual FPGA cores has been evaluated for integrating in a heterogeneous MPSoC architecture [FHGB+11; HFGS+11].

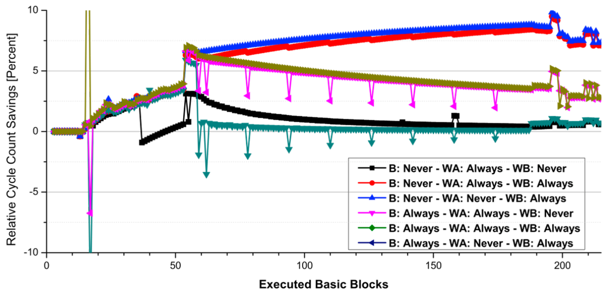


Figure 4.5: Comparison of various combinations of branch prediction and cache write strategies with a fused fetch/decode stage against an unmodified pipeline running an example program [HTGB+11]

As the experiments and benchmarks with the LISA modelling of pipeline fusion have shown potential benefit we are currently working on an implementation in VHDL. This will include extensive benchmarking to find characteristic code features which can be used as guide for optimisation strategies for pipeline fusion. This concludes our work on the *i*-Core μArch. We will now discuss the reconfigurable fabric and Instruction Set Extension (ISE).

The ISE provides applications with access to run-time reconfigurable hardware accelerators located in the reconfigurable fabric of the *i*-Core. The ISE consist of Special Instructions (SIs), which serve as the interface for the applications to the accelerators on the fabric. In [HBHG11] we have provided an overview of a first version of the fabric architecture and ISE, as well as an application case study of an H.264 Video Encoder utilising accelerators in the fabric.

SIs are described as data-flow graphs (DFGs, see Figure 4.6a), and do not include the exact configuration of the fabric (i.e. the locations where accelerators are loaded on the fabric, see Figure 4.6b), as this information would severely constrain the amount of SIs that can be present on the fabric at the same time, thus impairing the efficient use of the fabric. Instead, we have investigated placement of accelerators and binding of the DFG to the placed accelerators by the fabric management system (Figure 4.6c) *online*. To compare our online approach against an offline approach, a multi-tasking scenario was used, resulting in a speedup of 1.79x (Figure 4.7). These results have been published in [GBH12].

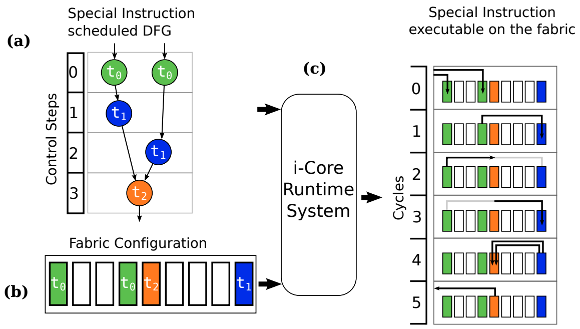


Figure 4.6: Tasks of a fabric management system

As part of the joint effort of defining the invasive language, we have provided pseudo-code which utilises the capabilities of the *i*-Core and allows management of the fabric through the use of basic invasion primitives (invade, infect, retreat).

We are currently implementing the *i*-Core on a Virtex-5 based demonstrator, in preparation for the joint architecture demonstration. Research

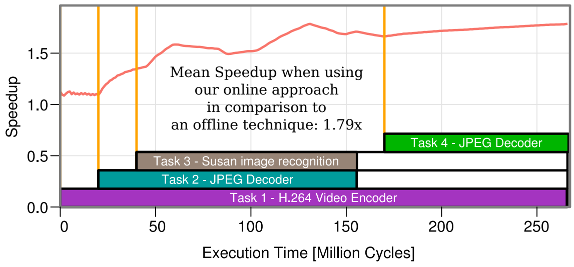


Figure 4.7: Evaluation of our approach

of the fabric hardware architecture is currently focused on interconnect structures for accelerators, and for the fabric management system we are investigating the reconfiguration sequence for accelerators in multi-tasking environments.

Based on our ongoing cooperation with Project C3, we have designed and implemented an extension of the i-Core, that was not foreseen in the proposal. It provides hardware support for the fast resolution of the static single assignment form used during compilation. This feature is used by the compiler from Project C3, therefore applications benefit from improved execution speed transparently.

Publications

[FHGB+11] P. Figuli, M. Hübner, R. Girardey, F. Bapp, T. Bruckschlögl, F. Thoma, J. Henkel, and J. Becker. “A heterogeneous SoC Architecture with embedded virtual FPGA Cores and runtime Core Fusion”. In: *NASA/ESA 6th Conference on Adaptive Hardware and Systems (AHS)*. San Diego, CA, USA, June 2011.

[GBH12] A. Grudnitsky, L. Bauer, and J. Henkel. “Partial Online-Synthesis for Mixed-Grained Reconfigurable Architectures”. In: *IEEE/ACM 15th Design Automation and Test in Europe Conference (DATE)*. to appear. Dresden, Germany, Mar. 2012.

[HBHG11] J. Henkel, L. Bauer, M. Hübner, and A. Grudnitsky. “i-Core: A run-time adaptive processor for embedded multi-core systems”. In: *Proceedings of the International Conference on Engineering of Reconfigurable Systems and Algorithms (ERSA)*. invited paper. Las Vegas, NV, USA, July 2011.

[HHBW+12] J. Henkel, A. Herkersdorf, L. Bauer, T. Wild, M. Hübner, R. Pujari, A. Grudnitsky, J. Heisswolf, A. Zaib, B. Vogel, V. Lari, and S. Kobbe. “Invasive Manycore Architectures”. In: *Proceedings of the 17th Asia and South Pacific Design Automation Conference (ASP-DAC)*. Jan. 30–Feb. 2, 2012, pp. 193–200.

[HFGS+11] M. Hübner, P. Figuli, R. Girardey, D. Soudris, K. Siozios, and J. Becker. “A Heterogeneous Multicore System on Chip with Run-Time Reconfigurable Virtual FPGA Architecture”. In: *Proceedings of the International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*. Anchorage, AK, USA, May 16–17, 2011.

[HTGB+11] M. Hübner, C. Tradowsky, D. Göhringer, L. Braun, F. Thoma, J. Henkel, and J. Becker. “Dynamic Processor Reconfiguration”. In: *Proceedings of the International Conference on Reconfigurable Computing and FPGAs (ReConFig)*. Cancun, Mexico: IEEE Computer Society, Nov. 30–Dec. 2, 2011.

B2: Invasive Tightly-Coupled Processor Arrays

Jürgen Teich

Srinivas Boppu, Frank Hannig, Vahid Lari, Shravan Muddasani

In this project, fundamental architectural (hardware) concepts to support invasion by clock cycle-based hardware signalling for *tightly-coupled processor arrays* (*TCPAs*, as shown in Figure 4.8) are investigated including the development of proper hardware controller circuitry.

Architectural Overview

Such processor arrays shall be designed to support a certain type or class of application-specific or domain-specific functions in image and signal processing domains. It is our goal to show that such processor arrays are particularly well-suited for implementing invasive nested loop programs and other computationally intensive kernels.

In the first year of research, we have proven that an invasion of an array of processing elements can be accomplished at an overhead of only a few clock cycles by exploiting regularity and by inventing and providing necessary hardware structures to support invasion efficiently in hardware and avoiding the overheads of thread creation at this level [LNHT11], which we expect to be at least two to three order of magnitude slower. In the InvasIC tiled-architecture [HHBW+12], TCPAs are used as a hardware accelerator for computationally intensive tasks. A simplified drawing of a TCPA with 36 processor elements (PEs) is sketched in Figure 4.8. The different rectangular areas denote two applications running simultaneously on the array.

Hardware-Supported Invasion

In order to determine and claim resources at run time, we investigated different invasion strategies suitable for invasion of PEs at the hardware level. As a result, we proposed two classes of strategies, namely *linear* and *rectangular invasion* [LNHT11]. A linear invasion reserves a linearly connected set of PEs, suitable for one-dimensional signal processing

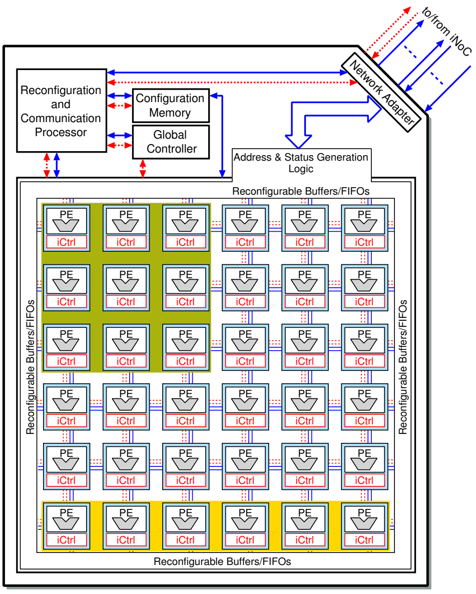

to/from iNoC
Network Adapter
Reconfiguration and Communication Processor
Configuration Memory
Global Controller
Address & Status Generation Logic
Reconfigurable Buffers/FIFOs
PE
iCtrl

Figure 4.8: Tightly-Coupled Processor Array (TCPA)

applications like filters for digital signal processing in the time domain. A rectangular invasion reserves rectangular regions of PEs, suitable for image processing algorithms such as optical flow or Harris corner detection, which are used in the computer vision applications of Project D1. Figure 4.8 depicts a rectangular invasion of an area of 3×3 PEs and a linear array with 6 PEs in green and yellow, respectively. Both types of investigated invasion strategies run in a distributed way using local propagation of invade instructions.

We designed hardware circuits for orchestrating such distributed invasion procedures, that are called invasion controllers (iCtrl in the figure). Two variants of invasion controllers were proposed, namely *FSM-based* and *programmable invasion controllers*. On these, we ported and evaluated various 1D and 2D invasion strategies with respect to invasion time, area, power and flexibility [LNHT11]. Briefly, we can summarise:

- FSM-based invasion controllers are hardwired, implementing a single invasion strategy. They are very fast since they can invade a linear array of PEs in time $O(N)$ (i.e., two clock cycles per PE).

In addition, they require only little area and power overheads. However, they lack the flexibility and extendability to support potentially other and more complex invasion strategies.

- Programmable invasion controllers, as the name suggests, are programmable small processors that enable to implement different invasion strategies. Their flexibility, however, comes at the cost of increased invasion time, area and power.

Furthermore, we developed concepts for efficiently signalling the claim results (i.e., information about the number of invaded PEs and their locations) of invasions back to the PE, originating an invasion [LHT11]. This information is crucial for calculating the interconnect configuration of the invaded PEs, specially for architectures with local connectivity like TCPA architectures. For exploring various design options and invasion strategies, a cycle-accurate C++ simulator for invasive TCPA architectures was developed also in close cooperation with Project C2. This simulator is capable of modelling both FSM-based and programmable invasion controllers at the functional and architectural abstraction levels. It was extensively used to develop and validate the above mentioned invasion strategies and to study the aforementioned signalling concepts published in [LHT11; LNHT11].

Using the simulator, we finally investigated a self-adaptive hierarchical power gating methodology [KGSH+11] to reduce the power consumption of TCPAs. In this approach, PEs and invasion controllers are grouped into different switchable power domains, whose power control signals are derived from the *invade*, *claim* and *retreat* signals. In the proposed strategy, we also explored the grouping of different a number of invasion controllers into one power domain in order to study the tradeoff between the power consumption and invasion time [LBMH+11]. We developed initial ideas for code generation and symbolic reconfiguration of TCPAs in cooperation with Project C3 [BHTP11]. An FIR filter application running on a first small TCPA array configuration was successfully implemented on the CHIPit system in cooperation with Project Z2, and was demonstrated at the InvasIC "Summer of Code" meeting in August 2011.

Outlook

Our current work and future activities include developing a back end for TCPA code generation in cooperation with Project C3, investigating and designing missing architectural components like the global controller, which controls the program execution on the TCPA based on the iteration bounds of a loop program, development of address generators to support data access from/to buffers with in order and out of order access capabilities, and interface modules for the efficient coupling of TCPA tiles to the *i*NoC (invasive Network-on-a-Chip developed in Project B5). Our future research will also include the investigation of invadable and reconfigurable buffer architectures for efficient decoupling of incoming and outgoing data-rates between TCPA tiles and other tiles, which is crucial for the performance.

Publications

[BHTP11] S. Boppu, F. Hannig, J. Teich, and R. Perez-Andrade. “Towards Symbolic Run-Time Reconfiguration in Tightly-Coupled Processor Arrays”. In: *Proceedings of the International Conference on Reconfigurable Computing and FPGAs (ReConFig)*. Cancun, Mexico: IEEE Computer Society, Nov. 30–Dec. 2, 2011, pp. 392–397. ISBN: 978-1-4577-1734-5. DOI: `10.1109/ReConFig.2011.91`.

[HHBW+12] J. Henkel, A. Herkersdorf, L. Bauer, T. Wild, M. Hübner, R. Pujari, A. Grudnitsky, J. Heisswolf, A. Zaib, B. Vogel, V. Lari, and S. Kobbe. “Invasive Manycore Architectures”. In: *Proceedings of the 17th Asia and South Pacific Design Automation Conference (ASP-DAC)*. Jan. 30–Feb. 2, 2012, pp. 193–200.

[KGSH+11] D. Kissler, D. Gran, Z. Salcic, F. Hannig, and J. Teich. “Scalable Many-Domain Power Gating in Coarse-grained Reconfigurable Processor Arrays”. In: *IEEE Embedded Systems Letters* 3.2 (June 2011), pp. 58–61. ISSN: 1943-0663. DOI: `10.1109/LES.2011.2124438`.

[LBMH+11] V. Lari, S. Boppu, S. Muddasani, F. Hannig, and J. Teich. “Hierarchical Power Management for Adaptive Tightly-Coupled Processor Arrays”. Talk, International Workshop on Adaptive Power Management with Machine Intelligence at International Conference on Computer-Aided Design (ICCAD), San Jose, CA, USA. Nov. 10, 2011.

[LHT11] V. Lari, F. Hannig, and J. Teich. “Distributed Resource Reservation in Massively Parallel Processor Arrays”. In: *Proceedings of the International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*. Anchorage, AK, USA: IEEE Computer Society, May 16–17, 2011, pp. 313–316. ISBN: 978-0-7695-4385-7. DOI: `10.1109/IPDPS.2011.157`.

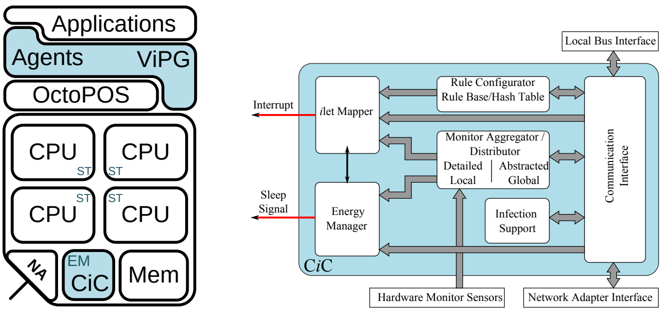
[LNHT11] V. Lari, A. Narovlyanskyy, F. Hannig, and J. Teich. “Decentralized Dynamic Resource Management Support for Massively Parallel Processor Arrays”. In: *Proceedings of the 22nd IEEE International Conference on Application-specific Systems, Architectures, and Processors (ASAP)*. Santa Monica, CA, USA: IEEE Computer Society, Sept. 2011, pp. 87–94. ISBN: 978-1-4577-1291-3. DOI: 10.1109/ASAP.2011.6043240.

B3: Invasive Loosely-Coupled MPSoCs

Andreas Herkersdorf, Jörg Henkel

Lars Bauer, Ravi Kumar Pujari, Benjamin Vogel, Thomas Wild

Performance-optimised and energy-efficient invasive computing on loosely-coupled RISC processing elements (PEs) is the major goal of this project. Functionalities with computational overhead within the *i*-let assignment process will be offloaded to dedicated macroarchitectural hardware extensions. These dynamic Many-Core *i*-let Controllers (C*i*Cs) supplement the software-based invasive run-time system (*i*RTSS). To increase the energy efficiency of invasive computing, the application knowledge provided by the invasive language is used to determine the resource requirements of the applications. By abstracting the power management decisions from the hardware towards the software layer, additional leakage power savings become possible. We call this approach Virtual Power Gating (ViPG).



(a) Tiled architecture **(b)** Core *i*-let Controller

Figure 4.9: Envisaged architecture

Figure 4.9a gives an overview of the envisaged architecture of a RISC-based compute tile [HHBW+12]. The *i*-let mapping decisions are C*i*C-accelerated and cooperate with the operating system (OctoPOS/*i*RTSS, Project C1). The agent system/*i*RTSS and the ViPG are tightly-coupled and receive the applications' resource and performance requirements. To support power gating in hardware, each C*i*C has an energy manager (EM) module to control the sleep transistors (ST) for each PE. The C*i*Cs aggregate the monitor data (e.g., performance counters or temperature values) provided by Project B4.

C*i*C

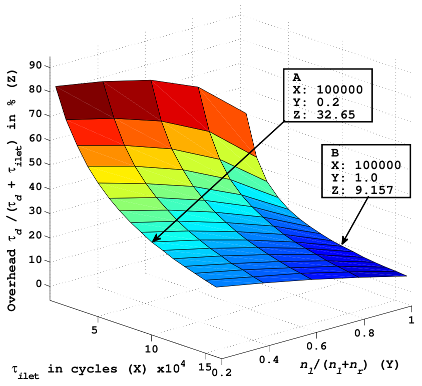


Figure 4.10: Reduction in *i*-let assignment decision overhead on prefetching the monitor data of cores from neighbouring tiles

The invasion overhead due to the monitoring data aggregation in the *i*-let assignment process is studied using an empirical formulation as a function of access delays for fetching the monitor data over the bus and the *i*NoC. The processing time overhead during the *i*-let mapping is above 30% for an average thread run-time of 100 kcycles when performed solely in software as depicted in Figure 4.10. By offloading certain functionalities from software (SW) into hardware (HW), this invasion overhead can be reduced to under 10%.

In order to let the invasion concept scale to over hundreds of cores connected over the *i*NoC, a HW–SW co-design for the *i*-let assignment policy is performed. Concretely, the *i*-let distribution strategy is hierarchically partitioned wherein (a) the coarse level *i*-let mapping decisions are performed by the agent system/*i*RTSS and (b) the fine-grained *i*-let assignments to individual cores within a tile are done by a dedicated HW extension block, the C*i*C. The complete analysis of the overhead along with the details of the hierarchical *i*-let assignment strategy and the functional block description (Figure 4.9b) of the C*i*C is presented in [PWHV+11].

Following a design review and refinements of the functional blocks of the C*i*C and network adapter (*i*NA), the DMA feature for transparent and fast transportation of *i*-lets across tiles is moved into the *i*NA. As a first step for building these HW extensions, an initial version of the C*i*C (C*i*C_v0) is prototyped on an FPGA. In this prototype design, *i*-let assignments are influenced by simulated temperature sensors per core within a tile. An *i*-let generator block is also built in the design to simulate the *i*-lets inflow over the *i*NoC.

Further work is under progress on building a rule based evaluator to perform fast *i*-let assignment decisions based on up-to-date monitor values and a monitor aggregation block to abstract and accumulate monitor data within the C*i*C. Collaboration with *i*RTSS developers (Project C1) for defining and synchronising the interaction between the *i*-let-mapper within the C*i*C and the OctoPOS is ongoing.

ViPG

In the ViPG part, the application knowledge about the current and anticipated resource requirements drive the power gating decisions. It is to be noted that shutting down and waking up PEs requires considerable energy and time. This wakeup overhead has to be hidden from the applications to increase the performance as well as the savings potential. While an incomplete shutdown only hampers the power savings, an incomplete wakeup has an unsolicited impact on the applications performance. To hide the wakeup overhead from the applications, the resource requirements have therefore to be expressed as early as possible.

This application knowledge is expressed by the invasive programming language based on X10 (Project A1/Project C2), by either constraints and hints or by the `reinvade()` command. Constraints express static resource requirements (e.g., no FPU necessary) while the hints express dynamic workload classifications. Whenever the applications resource requirements change, a `reinvade()` instruction informs the agent system (Project C1) and the ViPG system cooperatively. An example of changing workload requirements for an application is depicted in Figure 4.11. The information 'how often', and when, the application expresses their resource requirements is key for the ViPG system. We currently evaluate different prediction schemes about their applicability for the ViPG system.

To determine the applicability as well as the exploitable savings potential, a simulation infrastructure has been built. We combined a cycle accurate performance simulator (gem5) with a power model (McPAT) in a tool chain. To justify our work, we present a case study of different

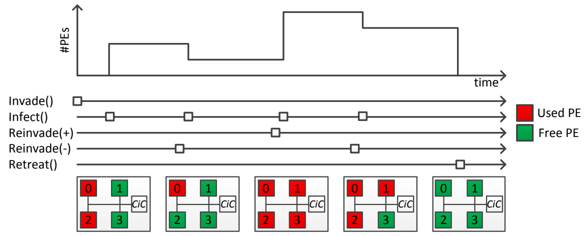
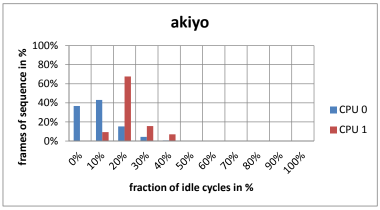


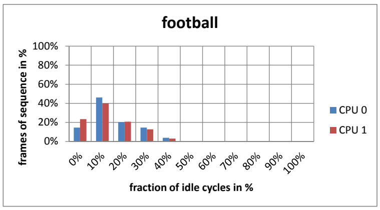
Figure 4.11: Changing workload requirements of an application and its impact on the PE usage.

video sequences encoded with H.264/AVC, a computational intensive application with input-dependent run-time behaviour. Exemplary, a very static scene (news spokesman, akiyo) and a scene with very high movement (sports game, football) are analysed on a 2 PE system. We can consider these video sequences as corner cases, real-world video sequences contain static and high movement scenes in quick succession. While the computing power of one PE may be sufficient for the static scenes, real-time encoding for high-movement scenes may not be possible. The question therefore is, when and for how long the idle second PE can be put into a sleep mode.

The determined fraction of idle cycles in these two video sequences are presented in Figure 4.12. Underlying our assumptions, the static scene doesn't need the second PE most of the time and even the first PE is often idle. Therefore, the second PE can be power gated while unused. This knowledge is available at the application level and can be provided to the ViPG. If it is not provided, the power management has to rely on a prediction and the changing PE requirements are likely to be detected too late. As pointed out, this would lead to performance degradation on the one hand and energy waste on the other hand.



(a) akiyo



(b) football

Figure 4.12: Fraction of idle cycles in two video sequences

Furthermore, the LEON3 processor is currently enhanced with clock gating capabilities. The `halt` instruction of the LEON3, which is executed in the idle loop, is used to trigger the clock gating. Ongoing work will capacitates applications to directly influence the power gating decisions. An FPGA board with power meters in the voltage regulators is used to determine the achievable savings. As power gating cannot be presented on the multi-FPGA demonstration platform (Project Z2) for technical reasons, we want to mimic the behaviour with clock gating and estimate the savings with the simulation infrastructure.

Publications

[HHBW+12] J. Henkel, A. Herkersdorf, L. Bauer, T. Wild, M. Hübner, R. Pujari, A. Grudnitsky, J. Heisswolf, A. Zaib, B. Vogel, V. Lari, and S. Kobbe. "Invasive Manycore Architectures". In: *Proceedings of the 17th Asia and South Pacific Design Automation Conference (ASP-DAC)*. Jan. 30–Feb. 2, 2012, pp. 193–200.

[PWHV+11] R. K. Pujari, T. Wild, A. Herkersdorf, B. Vogel, and J. Henkel. "Hardware Assisted Thread Assignment for RISC based MPSoCs in Invasive Computing". In: *Proceedings of the 13th International Symposium on Integrated Circuits (ISIC)*. Singapore, Dec. 2011.

B4: Hardware Monitoring System and Design Optimisation for Invasive Architectures

Doris Schmitt-Landsiedel, Ulf Schlichtmann

Ning Chen, Elisabeth Glocker, Christoph Knoth, Dominik Lorenz, Martin Wirnshofer

Resource-aware programming and operation is one of the essential innovative points of Invasive Computing. In this context the decision on where to execute an application should be based on actual physical hardware properties. This becomes even more important when considering the fact, that integrated circuits today and even more in the future are subject to significant variations - between different manufactured components (resulting from fluctuations in manufacturing process) and also over space (e.g., "hot spots") and time (short-term: resulting from fluctuations in operating conditions, long-term: resulting from degradation effects due to ageing).
The research goal of Project B4 is to provide means to measure, preprocess and communicate the specific status of a processing element to higher hardware layers, the operating system and applications. So these other layers are able to act considering the monitor data when, for example, choosing processing elements to execute applications on, or react when a critical status is detected. In turn the actions taken may influence the status of involved processing elements. This feedback-loop is shown in Figure 4.13. Therefore different monitor types are necessary. These include power consumption and temperature evolution, maximum performance in terms of speed or maximum clock frequency, also regarding age-dependent performance capabilities and reliability.
This project considers optimisation strategies and design of a monitoring system and corresponding circuits and interfaces including the demonstration by simulation, emulation on the FPGA hardware prototype platform and later on by preparing the implementation on ASIC hardware prototypes.

In [WHGSL11a] and [WHGSL11b] we demonstrated the monitoring of the maximum performance by in-situ delay monitors. These monitors are enhanced flip-flops that observe the timing of the circuit. Critical,

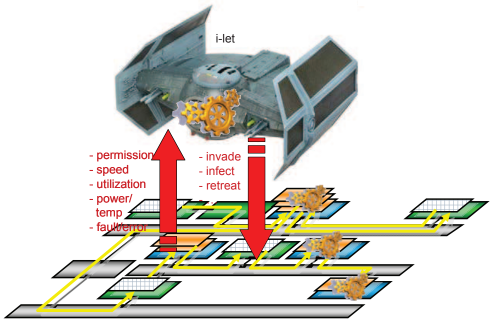


Figure 4.13: Resource-aware programming is a main feature of Invasive Computing. By providing a feedback-loop between the underlying physical hardware and higher layers the current state of the underlying parallel hardware platform may be taken into account when deciding if and which resources to invade, infect or retreat.

but not yet erroneous signal transitions, are detected as pre-errors. The pre-error rate is used as indicator for the remaining timing slack of the circuit. By use of these in-situ delay monitors, all kinds of variation and ageing effects are determined inside the real circuit and thus reliable performance information is provided. In [AGSLW11] different designs to implement these monitors are presented and the reliability of the timing information as well as the power overhead are carefully analysed. Our next research activities target further optimisation of these in-situ monitors especially for all kinds of ageing effects.

In [LBS11], we demonstrate an innovative approach to periodically monitor the ageing of ICs during operation. The basic concept is to identify all paths that potentially might become critical during the lifetime of an IC. As different paths can age at different rates, the critical path can change during the life of an IC. Ageing depends on operating and environmental conditions and therefore cannot be determined exactly before an IC is actually being used. But it is possible to identify a range within which the delay of a path will always be, regardless of where specifically it resides within the manufacturing window and what operating conditions (Temperature, Supply Voltage, Switching Activity) it will experience. It turns out that if this window is considered, for many circuits the number of paths that can potentially become critical is reduced significantly. Therefore it appears to be an option to test these paths periodically during the operation of an IC to detect any ageing that might endanger correct computation. This approach can be

considered as an alternative to the methods discussed above.
The research presented in [CLS11], [KUKS11] and [KJS12] addresses related topics. This will become especially useful for the ASIC design envisioned for the 2nd and 3rd phase of InvasIC.
In [KUKS11] SWAT, a highly optimised statistical timing analyser for digital circuits is presented that combines the accuracy of a transistor-level analysis with the performance of a gate-level analysis. SWAT is based upon a CSM (Current Source Model) for logic cells which considers transistor ageing and process variation and employs waveform truncation and dedicated solvers to significantly improve analysis performance without noticeable loss of accuracy. Parameter variations and ageing can be handed by Monte Carlo simulations and by a special sensitivity propagation mode, which expresses arrival times as a function of local and global parameter variations. This will allow very fast, but accurate analysis of the ASIC design, considering variations and ageing to ensure very robust InvasIC ASIC design. In [KJS12], the emphasis is put on power analysis instead of timing analysis.
In [CLS11] a flip-flop timing model is given that allows interdependency of different computation stages to be analysed via a static timing analysis at gate level. This is done by breaking the timing boundaries by explicitly building the functional relationship between clock-to-q delay and timing parameters at flip-flop data input. Ageing effects HCI and NBTI are also considered in the modelling to pave the way for ageing analysis. Application of this approach in InvasIC ASIC design will improve design performance even further.

We are currently working on the implementation of temperature monitors: We established a temperature model that will be extended by simulations of exemplary operating scenarios and we analysed temperature sensor circuits and chose suitable sensor circuits for both the FPGA hardware prototype platform and the implementation on ASIC hardware prototype.

To integrate the different monitors in the overall invasive architecture we will investigate possible solutions for the communication and abstraction of the hardware status data to higher layers adequate for each monitor type and with that implement the whole feedback-loop. In Figure 4.14 the necessary and involved parts for realising the data communication and abstraction in the loosely-coupled case is shown. Due to the nature of the FPGA hardware prototype platform not the same monitor sensors can be taken than the ones that will be imple-

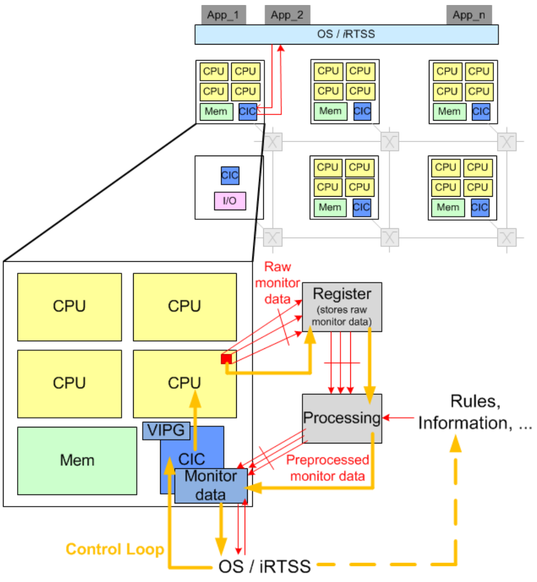


Figure 4.14: To integrate the monitors into the loosely-coupled invasive architecture the raw status data is stored, processed and the processed status data is then stored within range of the CiC. From there it will be accessible for higher layers like Agents or applications. Also the control loop going from the monitor sensors itself via the CiC's to the Agents and back is shown.

mented on the ASIC hardware prototype. So, dependent on the monitor type, we choose suitable monitor sensors for the FPGA or emulate the behaviour of the later ASIC monitor sensors by the use of models on the FPGA platform.

Publications

[AGSLW11] N. P. Aryan, G. Georgakos, D. Schmitt-Landsiedel, and M. Wirnshofer. “Comparison of In-situ Delay Monitors for Use in Adaptive Voltage Scaling”. In: *Kleinheubacher Tagung 2011*. to appear in Volume 10 (2012) in Advances in Radio Science (ARS) Journal. 2011.

[CLS11] N. Chen, B. Li, and U. Schlichtmann. “Timing Modeling of Flipflops Considering Aging Effects”. In: *International Workshop on Power and Timing Modeling, Optimization and Simulation (PATMOS)*. Vol. 6951. Lecture Notes in Computer Science (LNCS). Sept. 2011, pp. 63–72.

[KJS12] C. Knoth, H. Jedda, and U. Schlichtmann. “Current Source Modeling for Power and Timing Analysis at Different Supply Voltages”. In: *Design Automation and Test in Europe (DATE)*. To appear. Mar. 2012.

[KUKS11] C. Knoth, C. Uphoff, S. Kiesel, and U. Schlichtmann. “SWAT: Simulator for Waveform-Accurate Timing including Parameter Variations and Transistor Aging”. In: *International Workshop on Power and Timing Modeling, Optimization and Simulation (PATMOS)*. Vol. 6951. Lecture Notes in Computer Science (LNCS). Sept. 2011, pp. 193–203.

[LBS11] D. Lorenz, M. Barke, and U. Schlichtmann. *Finding Possible Critical Paths for On-line Monitoring Of Aging in Integrated Circuits*. Technical Report. Technische Universität München, Dec. 2011.

[WHGSL11a] M. Wirnshofer, L. Heiss, G. Georgakos, and D. Schmitt-Landsiedel. “A Variation-Aware Adaptive Voltage Scaling Technique Based on In-Situ Delay Monitoring”. In: *IEEE 14th International Symposium on Design and Diagnostics of Electronic Circuits & Systems*. 2011, pp. 261–266.

[WHGSL11b] M. Wirnshofer, L. Heiss, G. Georgakos, and D. Schmitt-Landsiedel. “An Energy-Efficient Supply Voltage Scheme using In-Situ Pre-Error Detection for on-the-fly Adaptation to PVT Variations”. In: *International Symposium on Integrated Circuits*. 2011.

B5: Invasive NoCs – Autonomous, Self-Optimising Communication Infrastructures for MPSoCs

Jürgen Becker, Andreas Herkersdorf, Jürgen Teich

Jan Heißwolf, Andreas Weichslgartner, Aurang Zaib

In recent years, Networks-on-Chip (NoCs) have emerged as interconnect for on-chip systems with many cores. The invasive architecture [HHBW+12] targets systems with hundreds of heterogeneous cores thereby offering features that enable software to dynamically allocate architectural resources depending on the current resource utilisation.

The research goal of this project is to provide a communication infrastructure for such large scale, heterogeneous architectures, as can be seen in Figure 4.15, that is able to efficiently handle diverse communication patterns dynamically. As the invasive Network-on-Chip (*i*NoC) is an integral part of a decentralised resource management strategy, all its components – that is, the network adapters (NA) and *i*NoC routers – have to support also link invasion [THHSL+11], this requires novel protocols and hardware methodologies.

As part of the overall decentralised resource management, we investigated the feasibility of decentralised embedding of communication topologies called *self-embedding* within the *i*NoC. In [WWT11], we presented the formal model for algorithms and showed that the decentralised mapping of tree-shaped applications and reserving of bandwidth can compete with central approaches in terms of average network utilisation, but offering a better scalability. Currently we are working on a hardware prototype of the *Invasive-Command-Initiator* (ICI) that will implement the self-embedding functionality as a part of the *i*NoC router.

Along with the research on self-embedding, we developed protocols and routers that support dynamic link invasion that also will provide the basis for autonomous mapping of communication graphs. The developed router, network adapter and the protocols support different types of communication that cooperatively share router's resources. Hard Quality-of-Service (QoS) guarantees can be given by reserving Guaranteed Service (GS) communication channels while Best Effort

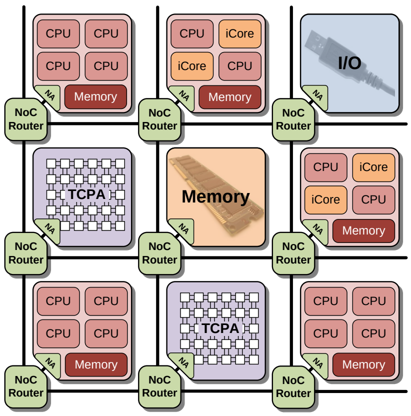


Figure 4.15: Heterogeneous invasive architecture consisting of heterogeneous tiles which are connected through the *i*NoC

(BE) communication channels can be used where no guarantees are required without the need of channel reservation. Thereby, BE may utilise resources that are not occupied by GS at that moment. Figure 4.16 shows such a scenario. In addition, the *i*NoC router is equipped with a dedicated control channel. This special control channel is necessary to enable *i*NoC-internal invasive features, such as, self-embedding, end-to-end flow control and global self-optimisation. In that way, the *i*NoC can efficiently process different communication requirements and achieve high resource utilisation.

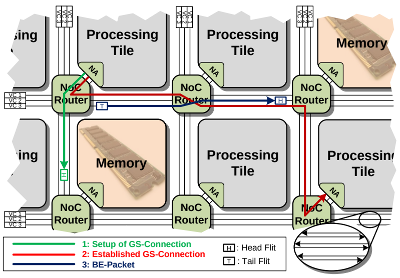


Figure 4.16: *i*NoC Data Transmission Example for QoS Support

To investigate the above mentioned invasion strategies and mechanisms at a higher level of abstraction, we developed a cycle-accurate SystemC model of the *i*NoC. It is highly parameterisable and can be used for design space exploration as well as to provide performance characteristics for the simulation of invasive applications and invasive architectures of Project C2. Along with the SystemC model, a parameterisable RTL model of the *i*NoC router was realised in SystemVerilog. It enables to obtain power, area and speed figures that are propagated to higher level simulation models. The RTL model will also be used as part of the final integrated demonstrator platform [BFHK+12] of the Project Z2. To approach that goal, a rudimentary architecture consisting of RTL models of LEON3 tiles, the NA and the *i*NoC is realised in collaboration with Project B3. A modular approach is followed to design the network adapter. It consists of a tile interface, the FIFO and the *i*NoC interface layer. The modular approach shall simplify the connection of the different tile types of our heterogeneous architecture. Only the tile interface layer needs to be replaced depending on the tile type. In the first year, RTL implementation of the network adapter to interface loosely coupled MPSoC tiles with *i*NoC is carried out. The current implementation provides support for load/store and message passing mechanism to LEON cores. Our next research activities target to investigate a novel self-optimisation strategy called *Re-Routing* to balance the link utilisation of the *i*NoC transparently from the tile perspective. Links with a high utilisation are identified by the *i*NoC itself. Alternative routes for connections through these links are searched, reserved and afterwards used for communication to relieve the high utilised link. Our SystemC simulator is currently extended to support that feature. To support the agent system (Project C1) in performing mapping decisions, we started investigating mechanisms for aggregation of monitoring data by the *i*NoC.

Publications

[BFHK+12] J. Becker, S. Friederich, J. Heisswolf, R. Koenig, and D. May. “Hardware Prototyping of Novel Invasive Multicore Architectures”. In: *Proceedings of the 17th Asia and South Pacific Design Automation Conference (ASP-DAC)*. Sydney, Australia, Jan. 30–Feb. 2, 2012, pp. 201–206.

[HHBW+12] J. Henkel, A. Herkersdorf, L. Bauer, T. Wild, M. Hübner, R. Pujari, A. Grudnitsky, J. Heisswolf, A. Zaib, B. Vogel, V. Lari, and S. Kobbe. “Invasive Manycore Architectures”. In: *Proceedings of the 17th Asia and South Pacific Design Automation Conference (ASP-DAC)*. Jan. 30–Feb. 2, 2012, pp. 193–200.

[THHSL+11] J. Teich, J. Henkel, A. Herkersdorf, D. Schmitt-Landsiedel, W. Schröder-Preikschat, and G. Snelting. “Invasive Computing: An Overview”. In: *Multiprocessor System-on-Chip – Hardware Design and Tool Integration*. Ed. by M. Hübner and J. Becker. Springer, Berlin, Heidelberg, 2011, pp. 241–268.

[WWT11] A. Weichslgartner, S. Wildermann, and J. Teich. “Dynamic Decentralized Mapping of Tree-Structured Applications on NoC Architectures”. In: *Proceedings of the Fifth ACM/IEEE International Symposium on Networks-on-Chip (NOCS)*. Pittsburgh, PA, USA, May 1–4, 2011, pp. 201–208.

C1: Invasive Run-Time Support System (*i*RTSS)

Wolfgang Schröder-Preikschat, Daniel Lohmann, Jörg Henkel, Lars Bauer

Benjamin Oechslein, Jens Schedel, Christoph Erhardt, Sebastian Kobbe

Project C1 investigates basic system-software support for application-aware static/dynamic configuration and on-demand adaptation of the invasive computing platform – bridging the gap from invasive hardware (project area B) to invasive applications (project area D). The scientific objective is a flexible run-time system coping with massively parallel, heterogeneous and dynamic workloads and requirements of the envisioned invasive applications. The challenge is to map (static/-dynamic) application properties to *i*RTSS configuration variants and to efficiently instantiate schemes for *i*-let entities considering cross-cutting and nonfunctional properties (e.g., energy consumption, timeliness).

Architectural Overview

Figure 4.17 provides a high-level view of the current *i*RTSS architecture. Key elements are OctoPOS,[2] the parallel operating system (POS) which implements the *mechanisms* of *i*RTSS to make all capabilities of the underlying hardware available to higher (software) levels, and the agent system, which provides global *i*RTSS *strategies* for resource management through means of self-adaption in order to cope with the scalability problem in large multi-core systems.

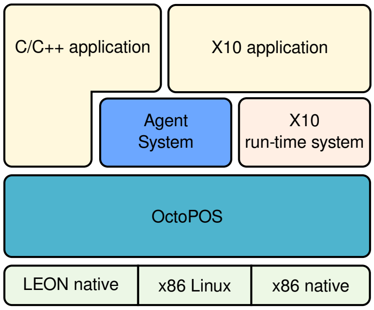


Figure 4.17: *i*RTSS Architecture

[2]The prefix "Octo" stems from the denotation of a nature which is highly parallel in its actions as well as adaptable to its particular environment: the octopus, being able to act in parallel by means of its tentacles, adapt itself through colour change, and, due to its highly developed nervous system, attune to dynamic environmental conditions and impact.

Due to its close interactions with the application it serves, the agent system is currently implemented in X10. The OctoPOS kernel is implemented in C++ and shall provide "bypasses" to the agent system for the quick and efficient bookkeeping of hardware resources and monitoring of hardware events. However, we are also considering an alternative architecture with a "bottom half" of the agent system residing directly within the OctoPOS kernel.

In a similar realm, OctoPOS provides specific system abstractions that serve the particularities of the X10 language and run-time system (Project C3), which we are currently investigating jointly with Project C3.

The Configurable OctoPOS Kernel

One key aspect in the design and development of OctoPOS is to make all the capabilities of the underlying hardware available to higher (software) levels in an "unfiltrated" way – especially to application programs – and yet leave these levels gradationally hardware-independent. In this realm, OctoPOS is provided as a *family of operating systems*. We have published a first paper about the general design rationale and the implementation of a first member of the OctoPOS family [OSKB+11]. Our results show, on the one hand, that on a six-core LEON3 processor the overhead of invasion (complete *invade/infect/retreat* cycle from one to six cores and back to one core) can be as low as 480 CPU cycles – it pays off to parallelise a workload as soon as its serial execution time surpasses 576 CPU cycles. On the other hand, the results (Figure 4.18) also show that the LEON does not scale beyond four shared-memory cores. As a consequence, the shared-memory tiles of the InvasIC hardware, developed in project area B, will be equipped with at most four cores.

The numbers in [OSKB+11] denote a lower bound for the overhead of invasive computing: We measured them in a single-application setting without multiplexing of the hardware in time and space; yet this setting already represents a valid member of the OctoPOS family. However, as soon as functional enrichments get introduced by extension aspects, concurrency on kernel-internal data structures does naturally increase.

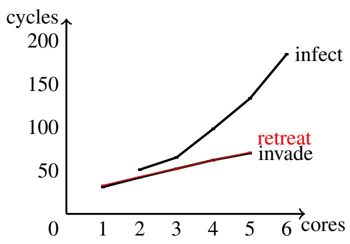


Figure 4.18: Execution times for invasive-computing primitives on a LEON MPSoC [OSKB+11]

The overall scientific objective is to define an operating system architecture that does not harm massive internal parallelism – the operating system shall not become the bottleneck of invasive computing. Herein, the challenge is to find lower-level (central) abstractions whose software implementations are lock-free and maximise parallelism. An important step towards lock-free algorithms is RTNCAS, a linearisable, wait-free and disjoint-access parallel multiword CAS (compare and swap) library [SSP11]. However, we also exploit and demand specific hardware support from the B projects in order to hide kernel latencies. In [HLSP11] we have investigated and quantified the benefits (up to 20x lower latencies) of (mis-)using the interrupt controller of commodity hardware for the very efficient scheduling and dispatching of thread control flows by a clever kernel design. These insights have been an important driver in our ongoing collaboration with Project B3 towards the hardware-assisted pre-scheduling and dispatching of *i*-lets by the CiC.

Nevertheless, we provide OctoPOS for a variety of platforms – with and without dedicated hardware support. The upcoming implementations for x86-based commodity hardware and a Linux "guest mode" (Figure 4.17) ease the development and evaluation of the invasive software stack (applications, X10 extensions, compiler). They also make it possible to compare the approach of invasive computing [THHSL+11] with other players in the field that run on standard hardware only. From the perspective of OctoPOS, these implementations become further members of the family, which, however, requires a fine-grained configuration and adaptation of its low-level abstractions. Internally, OctoPOS uses aspect-oriented programming techniques for this purpose instead of the "`#ifdef` hell" commonly found in configurable systems software. While investigating on a clean integration of *i*RTSS into Linux as a host system, we found such an "`#ifdef` hell" in the Linux kernel, where it had already led to hundreds (!) of previously undetected maintenance bugs [TLSSP11]. These results underline the necessity for advanced configuration techniques in OctoPOS.

The Agent System

The horizontal adaptation using agent-based self-adapting resource allocation is implemented in a first version. In [KBHL+11] we have shown that managing resources of an on-chip many-core system in a distributed manner is feasible and beneficial, especially if the system has hundreds or thousands of cores. We have also shown that managing the resources of a large on-chip many-core system (an NP-hard problem) in only one central place leads to issues in terms of computational complexity. We have compared our approach to a centralised resource manager in multiple system size configurations with various workload profiles. The comparison of the computational complexity is shown in Figure 4.19. The agent system simulation environment for design space exploration (see Figure 4.20) has been developed and is used to evaluate the agent system. As each of the agents only needs to take care of a reduced subset of the large search space, the computational complexity for each of these parallel working agents is also significantly smaller. On a simulated 1024-core system running 32 applications, less than one percent of the calculations need to be performed in each agent compared to a centralised approach. Our approach required to send more messages trough the NoC, but we were able to show that in average the messages are shorter and most often only require a few hops on the NoC when distributed throughout the entire chip. Therefore, our approach requires notably less network bandwidth compared to a centralised solution (12,75% for the simulated 1024 core system with 32 applications). In [HHBW+12] the role and impact of our agent system with respect to the InvasIC hardware architecture have been presented.

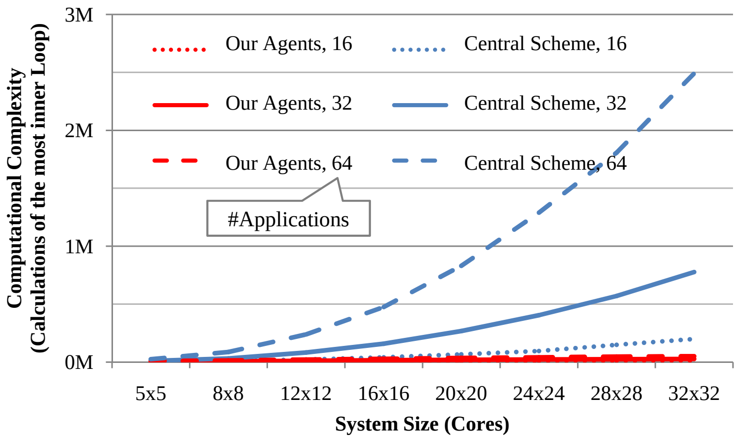


Figure 4.19: Comparison of the computational complexity of our agent system and a centralised resource manager for different system sizes and various workload profiles [KBHL+11].

We have proposed the concept of “hints” as an extension of the constraint system developed for the X10 language extensions (Project C3). Hints within an `invade` statement are necessary for the application to be able to express not only which exact resources the application re-

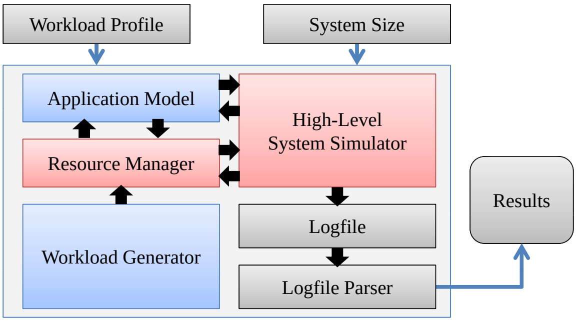


Figure 4.20: Components of the agent system simulation environment for design space exploration. The agent system is implemented as the resource manager and is triggered by the high-level workload generator and the application model.

quires, but also how beneficial it would be to obtain these resources – for instance, the expected performance gain if the application acquires an *i*-Core instead of a regular RISC core or the expected gain of each additional core. This information is required for the agents to make decisions about which resources are actually given to which application in order to optimise the overall system performance. If no hints are provided, only a default behaviour could be assumed, which leads to potentially worse decisions.

A topic of ongoing research is the integration of dynamic information (e.g., monitoring data (Project B4)) into the mapping-decision process. In this realm we cooperate with the *i*NoC (Project B5 and Project B3) towards a hardware-supported aggregation and distribution of dynamic information. The application's knowledge about the anticipated resource requirements can then be used to save power. While the agent system is responsible for the resource mapping, the Virtual Power Gating (ViPG, Project B3) aims at activating and deactivating already mapped resources. We envision to enhance the agent system's mapping decisions with the ViPG gating decisions and vice versa. As a first step towards this goal, a combination of both simulation infrastructures is planned to evaluate the power-saving potentials.

Outlook

In 2012 we will further investigate, and possibly finalise, the interfaces towards the application/compiler interface (project area D, Project C3) and the invasive hardware platform (project area B). An important milestone in this realm is the stepwise integration of our results into the first demonstrator scenario of Project Z2. Residing between “a rock and a hard place” (application and hardware projects) the operating system naturally plays a central role in this setting. To obtain additional data for latency estimations, we will furthermore implement a version of our agent system for the Intel SCC 48-core system and OctoPOS for an x86 AMD 48-core including the Linux guest support.

Publications

[HHBW+12] J. Henkel, A. Herkersdorf, L. Bauer, T. Wild, M. Hübner, R. Pujari, A. Grudnitsky, J. Heisswolf, A. Zaib, B. Vogel, V. Lari, and S. Kobbe. “Invasive Manycore Architectures”. In: *Proceedings of the 17th Asia and South Pacific Design Automation Conference (ASP-DAC)*. Jan. 30–Feb. 2, 2012, pp. 193–200.

[HLSP11] W. Hofer, D. Lohmann, and W. Schröder-Preikschat. “Sleepy Sloth: Threads as Interrupts as Threads”. In: *Proceedings of the 32nd IEEE International Symposium on Real-Time Systems (RTSS)*. Vienna, Austria: IEEE Computer Society, Dec. 2011, pp. 67–77. ISBN: 978-0-7695-4591-2.

[KBHL+11] S. Kobbe, L. Bauer, J. Henkel, D. Lohman, and W. Schröder-Preikschat. “DistRM: Distributed Resource Management for On-Chip Many-Core Systems”. In: *Proceedings of the IEEE International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*. Taipei, Taiwan, Oct. 9–14, 2011, pp. 119–128.

[OSKB+11] B. Oechslein, J. Schedel, J. Kleinöder, L. Bauer, J. Henkel, D. Lohmann, and W. Schröder-Preikschat. “OctoPOS: A Parallel Operating System for Invasive Computing”. In: *Proceedings of the International Workshop on Systems for Future Multi-Core Architectures (SFMA)*. Ed. by R. McIlroy, J. Sventek, T. Harris, and T. Roscoe. Vol. USB Proceedings. Sixth International ACM/EuroSys European Conference on Computer Systems (EuroSys). EuroSys. Salzburg, Austria, Apr. 2011, pp. 9–14.

[SSP11] P. Stellwag and W. Schröder-Preikschat. “Challenges in Real-Time Synchronization”. In: *Proceedings of the 3rd USENIX Workshop on Hot Topics in Parallelism (HotPar)*. Ed. by M. McCool and M. Rosenblum. Berkeley, CA, USA: USENIX Association, May 2011.

[TLSSP11] R. Tartler, D. Lohmann, J. C. R. Sincero, and W. Schröder-Preikschat. “Feature Consistency in Compile-Time Configurable System Software”. In: *Proceedings of the Sixth International ACM/EuroSys European Conference on Computer Systems*

(EuroSys). Ed. by C. Kirsch and G. Heiser. Salzburg, Austria: ACM Press, Apr. 2011, pp. 47–60.

[THHSL+11] J. Teich, J. Henkel, A. Herkersdorf, D. Schmitt-Landsiedel, W. Schröder-Preikschat, and G. Snelting. “Invasive Computing: An Overview”. In: *Multiprocessor System-on-Chip – Hardware Design and Tool Integration*. Ed. by M. Hübner and J. Becker. Springer, Berlin, Heidelberg, 2011, pp. 241–268.

C2: Simulation of Invasive Applications and Invasive Architectures

Frank Hannig, Michael Gerndt, Andreas Herkersdorf

Vahid Lari, Marcel Meyer, Sascha Roloff, Aurang Zaib

Project C2 investigates novel simulation methodologies that enable the validation and variants' exploration of all essential features of invasive computing in order to guide research directions at an early project stage. Project C2 has two major research fields: (a) Behavioural simulation of invasive resource-aware programs and (b) Performance evaluation of individual architectures and an integrated simulation methodology to co-simulate different types of invasive architectures. In order to handle the complexity and diversity of the considered architectures as well as different invasive programming, resource management and invasion strategies, new methods for the modularisation and orthogonalisation of these exploration concerns are developed. This project has a fundamental role within the TCRC by providing evaluation facilities and by assisting in the optimisation of the concepts of invasive computing across all project areas, especially, without the need to have full hardware or software implementations available.

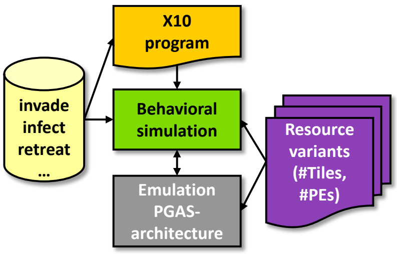


Figure 4.21: Overview of the functional simulation approach.

Behavioural Simulation

The main idea of our behavioural simulation methodology is to provide the main commands of invasion (invade, infect, etc.) early to the application programmer and to offer the possibility to simulate resource-awareness for different invasive architectures. This shall allow the early validation of invasive programming concepts as well as the investigation of a broad range of different invasive hardware platforms on a functional layer. An introduction to the resource-aware programming concepts and the functional simulator was published in cooperation with Project A1 in [HRST+11]. A general introduction to the invasive computing paradigm is given in [THHSL+11].

In the last year, we have developed a functional simulation environment completely based on the programming language X10, which provides a functionally correct interpretation of the resource-aware programming constructs provided by the framework of invasive programming. It also delivers important timing information about the parallel execution of several applications running on an invasive architecture, taking into account the computational properties of the different types of processing elements. Due to the fact that a cycle-accurate simulation of heterogeneous future MPSoCs including hundreds or thousands of cores would be much too slow, we have developed a novel high-level simulation approach, which tackles the complexity and the heterogeneity of such systems and enables the early investigation of resource-aware programming concepts. In Figure 4.21, a general overview of the functional simulation approach is depicted. Several invasive applications are written in X10. The simulation framework itself is also realised in X10 using a library-based approach. The applications are simulated on a modelled heterogeneous tiled architecture, which provides a rudimentary abstraction of an invasive architecture. Means are provided to describe such architectures in X10 in terms of number and topology of tiles, including their contained processing elements as well as their local memory sizes. The main idea is to model the state and the utilisation of resources by concurrently simulated processes. These are implemented in X10 by lightweight threads called activities and thus implemented in the same way as application *i*-lets. Each processing element may be individually customised by defining its type (e.g., RISC, *i*-Core or TCPA), type-dependent properties (cycles per instruction, cache type, clock frequency, scratchpad size, etc.) and monitors for certain time-variant states (e.g., load, temperature, faultiness). Subsequently, the resulting X10 program is compiled and executed on the host PC and simulates the parallel execution of several invasive applications running on the modelled target architecture. This simulation of the interplay between invasive program behaviour and the resulting states of the underlying processing resources such as their temperature, load or faultiness in dependence of their state of invasion is one of the key features of this functional simulation concept. Finally in [RHT12], we will present a new timing model that allows also an *approximate time* simulation of applications running on cores with different computational properties without requiring detailed simulation models of the target processors. Instead, we use a hybrid approach based on performance counters and analytical models. This method is much faster than a cycle-accurate simulation and thus allows the investigation of invasive program behaviour of hundreds of applications running on a heterogeneous invasive architecture.

Outlook

The next steps of research are to formulate and integrate also timing models of complex architectures such as TCPAs (Project B2) in order to obtain adequate speedup values for the execution of *i*-lets also here. Moreover, an integration of a network model for the calculation of network latencies between communicating *i*-lets on different tiles needs to be achieved. Finally, the timing simulation is currently realised in a sequential way. So, another next step is to parallelise the simulation by making use of current multi-core computers in order to speed up the simulation of invasive applications on heterogeneous targets.

Integrated Simulation Methodology

The second main objective is a uniform simulation methodology, which is based on common methods to model invasive applications, concerted mechanisms to provide resource-awareness. In addition, it will allow to integrate different architecture simulators across all hardware architectures covered in the TCRC. In this year, we have evolved the interfaces for the common simulation framework, the Integrated Invasive Simulation Framework (IISF) for investigating invasive resource-aware architectures. Here, we have followed a novel and unique layered approach that separates the simulation of the invasive constructs from non-invasive parts in an invasive application. The invasive behaviour of the application is captured in a control flow graph which is represented accordingly in the simulation input file, the Abstract Intermediate Representation (AIR). Abstract Invasive Machine (AIM) is the component which sends non-invasive parts to simulate on underlying simulation

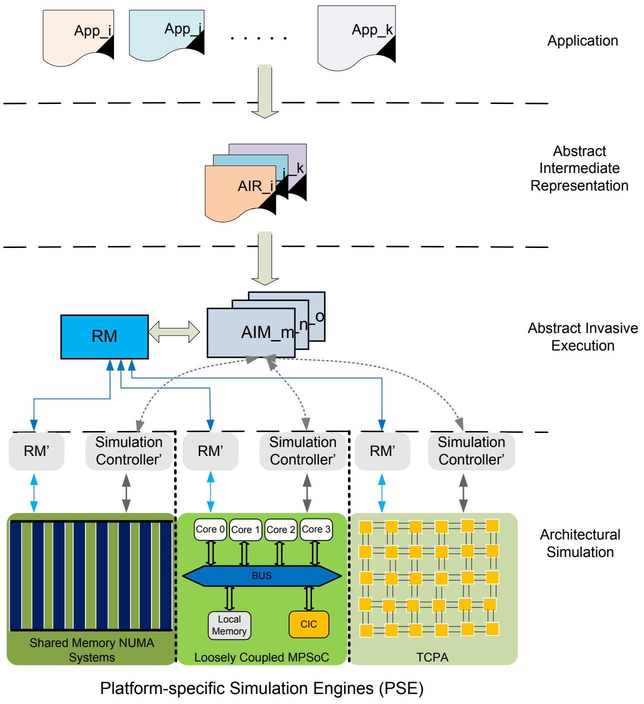


Figure 4.22: Integrated Invasive Simulation Framework (IISF).

engines and executes invasive parts by itself with help of Resource Manager (RM). Figure 4.22 describes this layered simulation approach for the simulation of invasive architectures. In the scope of IISF, the important concerns for co-simulation of heterogeneous architectures like global timing and synchronisation between different architecture simulators would be addressed in 2012.

The architecture simulator for loosely-coupled MPSoC (LCMPSoC) targeted for intra-tile investigations was adapted to the proposed layered simulation approach in the current year. This simulator allows simulation of applications keeping in view their invasion requirements. The actual execution on on underlying MPSoC architecture is abstracted through traces. Also in the scope of the architecture simulators, a cycle-accurate simulator for invasive TCPA architectures (depicted in the figure at bottom right) was developed in C++ in close cooperation with Project B2. This simulator is used for exploring design options and invasion strategies. Therefore, it is also capable of modelling both

FSM-based and programmable invasion controllers at the functional and architectural abstraction levels. It was extensively used to develop and validate many investigated invasion strategies, to study the signalling concepts published in [LHT11; LNHT11] as well as to investigate different hierarchical power management techniques for TCPAs [LBMH+11].

In 2012, the simulation model will be expanded to include the *i*NoC in collaboration with Project B5. Furthermore, the environment shall be extended to investigate and integrate invasive resource management strategies and monitoring concepts.

Publications

[HRST+11] F. Hannig, S. Roloff, G. Snelting, J. Teich, and A. Zwinkau. "Resource-Aware Programming and Simulation of MPSoC Architectures through Extension of X10". In: *Proceedings of the 14th International Workshop on Software and Compilers for Embedded Systems (SCOPES)*. St. Goar, Germany: ACM Press, June 27–28, 2011, pp. 48–55. ISBN: 978-1-4503-0763-5. DOI: `10.1145/1988932.1988941`.

[LBMH+11] V. Lari, S. Boppu, S. Muddasani, F. Hannig, and J. Teich. "Hierarchical Power Management for Adaptive Tightly-Coupled Processor Arrays". Talk, International Workshop on Adaptive Power Management with Machine Intelligence at International Conference on Computer-Aided Design (ICCAD), San Jose, CA, USA. Nov. 10, 2011.

[LHT11] V. Lari, F. Hannig, and J. Teich. "Distributed Resource Reservation in Massively Parallel Processor Arrays". In: *Proceedings of the International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*. Anchorage, AK, USA: IEEE Computer Society, May 16–17, 2011, pp. 313–316. ISBN: 978-0-7695-4385-7. DOI: `10.1109/IPDPS.2011.157`.

[LNHT11] V. Lari, A. Narovlyanskyy, F. Hannig, and J. Teich. "Decentralized Dynamic Resource Management Support for Massively Parallel Processor Arrays". In: *Proceedings of the 22nd IEEE International Conference on Application-specific Systems, Architectures, and Processors (ASAP)*. Santa Monica, CA, USA: IEEE Computer Society, Sept. 2011, pp. 87–94. ISBN: 978-1-4577-1291-3. DOI: `10.1109/ASAP.2011.6043240`.

[RHT12] S. Roloff, F. Hannig, and J. Teich. "Approximate Time Functional Simulation of Resource-Aware Programming Concepts for Heterogeneous MPSoCs". In: *Proceedings of the 17th Asia and South Pacific Design Automation Conference (ASP-DAC)*. Sydney, Australia, Jan. 30–Feb. 2, 2012, pp. 187–192.

[THHSL+11] J. Teich, J. Henkel, A. Herkersdorf, D. Schmitt-Landsiedel, W. Schröder-Preikschat, and G. Snelting. “Invasive Computing: An Overview”. In: *Multiprocessor System-on-Chip – Hardware Design and Tool Integration*. Ed. by M. Hübner and J. Becker. Springer, Berlin, Heidelberg, 2011, pp. 241–268.

C3: Compilation and Code Generation for Invasive Programs

Gregor Snelting, Jürgen Teich

Matthias Braun, Sebastian Buchwald, Frank Hannig, Georgia Kouveli, Manuel Mohr, Alexandru Tanase

Project C3 investigates compilation techniques for invasive architectures. A compiler for the concrete X10-based language defined in Project A1 is being developed. Efficient code generation for invasive constructs is essential. Back ends are developed based on the FIRM infrastructure that generate code for SPARC architectures and TCPAs targeting loop-level parallelism. Moreover, we investigate symbolic mapping techniques which will allow to generate invasive multi-processor programs that have the capability to correctly synchronise their computations and communications at run time on an array of processors, based on the result of invasion.

Architectural Overview

In this project, we consider compilation and code generation as well as program transformation and optimisation techniques for non-regular (procedural) as well as task-level and regularly-structured (e.g., loop-level) code. An overview of the compiler framework is shown in Figure 4.23. For the concrete language and its interfaces as defined in Project A1, a compiler needs to be developed. The back end will be based on the existing FIRM infrastructure for code optimisation.

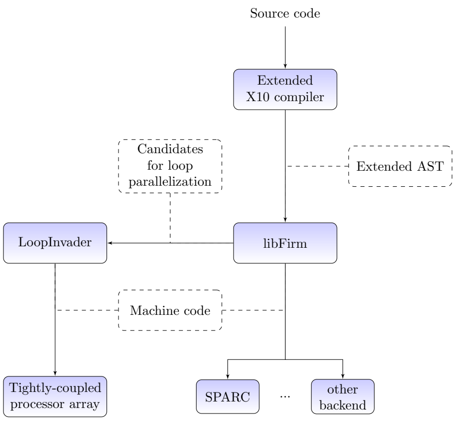


Figure 4.23: Compiler framework for Invasive Computing.

FIRM provides static single assignment (SSA) form as a basis for program analysis and optimisation. The compiler must eventually support both loosely- and tightly-coupled as well as heterogeneous and homogeneous invasive multi-processor target architectures.

The compiler will be based on the existing X10 compiler, but with front end extensions for invasive constructs, and a new or modified back end for SPARC architectures, as well as optimisations for efficient utilisation of invasive hardware and operating system support, respectively. In order to exploit invasive computing concepts [THHSL+11] at the level of loop programs, symbolic techniques are necessary that describe sets of processor mappings, schedules and synchronisations of loop computations in dependence on the number N of processors, which will only be known at run time. Here, the main challenge is to define such parametrised schedules and mappings together with their proper mathematical foundations, as well as to derive compact case-dependent processor and time mappings. A compiler for invasive loop nests must not only generate functional code, but also control code. In addition, there will be a high potential for several optimisations, such as determination of optimal invasion parameters, generation of the control code for splitting registers, copying of data/program code, inter-processor link configuration and memory-access scheduling based on the run-time parameter N. It is intended to provide a prototype code generator for loosely-coupled RISC-type processors as well as for tightly-coupled processor arrays (TCPAs), supporting the acceleration of nested loop programs. The output of a code generator shall thus be assembly-level code for architectures such as investigated in Project B2 and Project B3. The generated codes shall be used as input for the simulation tools developed in Project C2 as well as for real hardware once the central FPGA demonstrator (Project Z2) is ready to run first code examples.

Symbolic Loop Parallelization (Loop Invader)

In this part of the project, we focus on compiler transformations for the adaptive parallel execution of loop programs on processor arrays such as tightly-coupled processor arrays (TCPAs). A simplified drawing of a TCPA with 24 processor elements (PEs) is sketched in Figure 4.24. Here, the different rectangular areas denote three applications running simultaneously on the array. Whereas static mapping and loop parallelisation techniques for coarse-grained reconfigurable and TCPA architectures are well studied [Han10; VRMD+10], we proposed and formalised for the first time symbolic tiling as an automatic program

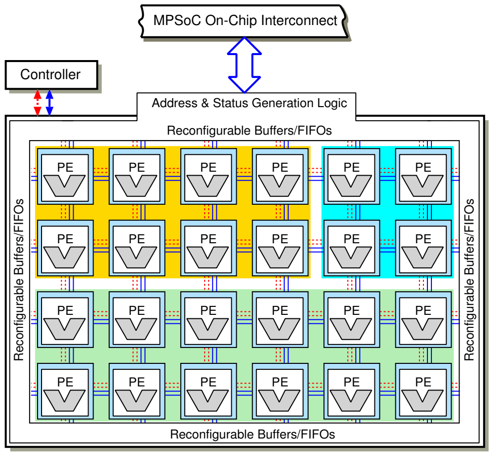
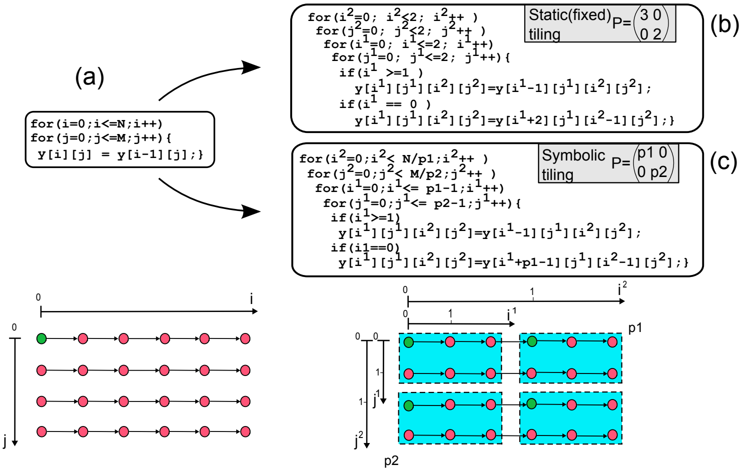

MPSoC On-Chip Interconnect
Controller
Address & Status Generation Logic
Reconfigurable Buffers/FIFOs
Reconfigurable Buffers/FIFOs
Reconfigurable Buffers/FIFOs
Reconfigurable Buffers/FIFOs
PE

Figure 4.24: Tightly-coupled processor array (TCPA)

transformation for symbolic parallelisation of nested loop programs with uniform data dependencies. This step is essentially important for invasive programming on MPSoCs, because here the number of processors, which determines the choice of tiling for parallelisation, is not known until run time. For executing loop programs on such architectures, tiling is needed as a compiler transformation to split iterations of a loop to the set of available processing elements. For illustration, consider the nested loop program in Figure 4.25(a) and its iteration space visualised for $N = 6$ and $M = 4$. Each nodes represents an iteration of the loop program, that is, the execution of the loop body for iteration (i, j). Data dependencies between different iterations are depicted by the directed edges. Tiling increases the depth of the loop nest, in our example from a 2-deep nest to a 4-deep nest. The size of a tile can be represented by a so-called tiling matrix P.

Tiling the example into 3×2 tiles can be represented by the matrix P, resulting in the statically tiled code represented in Figure 4.25(b). After tiling, the innermost loop iterates over the iterations contained in a tile and the outer loop iterates over the origins of the tiles. When the number of available processors in the array is not known at compile-time, different possibilities might be considered: The first one is to store a program configuration for each possible array size and to select the appropriate one at run time. For this approach, obviously, the number of different configurations and thus the amount of necessary instruction memory can explode easily. Another possibility might be just-in-time compilation. However, a compiler framework could consume easily dozens to hundreds of megabytes of memory. In addition, parallelisation and mapping of a loop program is a time consuming process. Therefore, this approach is usually not viable for embedded architectures. Due to the aforementioned arguments, the need for *symbolic* loop tiling arises. Here, a tiling transformation is specified by a parametric tiling matrix P. For example, tiling maps the loop to a $n \times m$ processor array, with $n = N/p_1$, $m = M/p_2$, where m and n are unknown at compile-time. An example of a symbolically tiled C loop and the corresponding symbolic tiling matrix P are shown in Figure 4.25(c). The program to be parallelised is specified in a subset of the language C with annotated static control parts, which can be represented in the polyhedral model. Currently, we are also investigating the issue of symbolic scheduling that is needed for scheduling our symbolically tiled code. Here, there are several remaining issues, for instance that the number of statically generated cases of latency-optimal schedules for the symbolically tiled code can unfortunately be n!. Although being exponential, the overhead is typically not too high for loops with usually less than $n = 4$ dimensions and small loop kernels[3]. For a small number of dimensions this will

[3]Also note that not all generated scheduled loop codes must fit simultaneously into the program memory of a processor, but could be loaded at run time instead.


(a)
(b)
(c)
Static(fixed) tiling P= (3 0 / 0 2)
Symbolic tiling P= (p1 0 / 0 p2)
0
i
j
i^1
i^2
j^1
j^2
1
p1
p2

```
for(i=0;i<=N;i++)
for(j=0;j<=M;j++){
 y[i][j] = y[i-1][j];}
```

```
for(i^2=0; i^2<2; i^2++ )
 for(j^2=0; j^2<2; j^2++ )
  for(i^1=0; i^1<=2; i^1++)
   for(j^1=0; j^1<=2; j^1++){
    if(i^1 >=1 )
      y[i^1][j^1][i^2][j^2]=y[i^1-1][j^1][i^2][j^2];
    if(i^1 == 0 )
      y[i^1][j^1][i^2][j^2]=y[i^1+2][j^1][i^2-1][j^2];}
```

```
for(i^2=0;i^2< N/p1;i^2++ )
 for(j^2=0;j^2< M/p2;j^2++ )
  for(i^1=0;i^1<= p1-1;i^1++)
   for(j^1=0;j^1<= p2-1;j^1++){
    if(i^1>=1)
     y[i^1][j^1][i^2][j^2]=y[i^1-1][j^1][i^2][j^2];
    if(i1==0)
     y[i^1][j^1][i^2][j^2]=y[i^1+p1-1][j^1][i^2-1][j^2];}
```

Figure 4.25: Tiling of a loop program (a), static tiling (b), symbolic tiling (c) for $p_1 = 3$ and $p_2 = 2$.

still keep the size of code smaller in comparison to dynamic (JIT) compilation, for which the dynamic compiler might need a very large amount of space. The design of symbolic scheduling methods is part of our ongoing research.

Code Generation for TCPAs and Resource Management for Other Targets

In order to symbolically map loop programs onto parts of a TCPA an appropriate symbolic code generation back end is also needed, which generates code that can be used with slight modifications at run time for different numbers of available PEs. In close collaboration with Project B2, we studied first symbolic approaches for run-time reconfiguration of TCPAs [BHTP11]. This work is an important basis for a code generator for the TCPA targets designed in Project B2. We plan a release of the code generator for the next year as well as the integration in the overall compiler infrastructure (libFirm).

Furthermore, we have adapted the invasive computing paradigm for other tiled architectures, namely Tilera's TILEPro64 and Intel's Single-Chip Cloud Computer, which have 64 and 48 processors, respectively. Here, we investigated the resource-aware programming concepts and have designed new libraries, which provide support for the basic invasive language constructs. We studied the overheads of invasion on these architectures and looked into the trade-offs of centralised versus distributed approaches of resource management [KHLT11; MTKB+11].

In addition, Richard Membarth and others have developed a new approach for resource management of many-core processors (graphics processing units, GPUs) in the field of medical imaging [MLHT+12]. In the approach, real-time constraints, dependencies between tasks, multiple as well as heterogeneous GPUs are supported.

Invasive X10 Front End

We use the existing X10 front end which is available as open-source software. This allows us to reuse the existing syntax and semantic checking code as well as the program representation. We develop ways to transform the X10-AST into the libFIRM intermediate representation. This is necessary to generate code for the TCPA hardware, the *i*-Core instruction set extensions and to ensure an optimal implementation of the invasive constructs. We also expect faster compile times and improved debug information by using a direct code generation approach in contrast to generating C++ as the original X10 compiler. As we are not running on a standard Posix/MPI platform we are also developing a new run-time library which supports the *i*RTSS and our custom hardware.

So far we have implemented a Java wrapper for libFIRM (jFirm) so we can integrate with the existing X10 compiler. We can compile the sequential subset of X10, including modern language features such as object orientation with classes and interfaces, generic code and closures. We specified operating system interfaces in collaboration with Project C1. Work on the parallel constructs has begun. The implementation is in schedule and we expect to have a complete sequential and partially invasive version ready by the end of 2012.

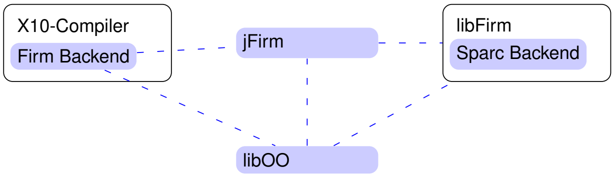


Figure 4.26: Compiler Components (newly developed ones marked in blue)

SPARC Back End

The general purpose cores found in an invasive system have a SPARC instruction set optionally with *i*-Core extensions. There will be variants with and without floating-point support.

To this end a new back end has been developed using the existing infrastructure in libFIRM. This involved creating code-selection strategies, and handling the SPARC calling convention which employs register windows. Register allocation and scheduling is performed with the generic infrastructure. We further developed peephole optimisations and special code generation phases to fill delay slots and respect the stack alignment requirements of the application binary interface. To handle software floating-point a new pass which replaces arithmetic operations with calls into an emulating library has been created. We are in the process of extending our register allocator to support aliased floating-point registers as found in the SPARC architecture. The back end has matured to a point where efficient code is generated and the SPEC2000 benchmark suite is handled. We are ahead of schedule giving us time to further tune the back end and explore instruction set extensions (see next section).

Register Permutations

The collaboration with Project B1 allows us to explore extended instruction sets: Our previous work [HGG06; BH09; BZB11] shows that register allocation for programs in SSA-forms leads to an optimal assignment of registers. Translating out of SSA form however requires parallel copy constructs. These are traditionally implemented by sequences of mov and xor instructions for copying and exchanging values. In practise minimising these instructions is an NP-hard problem (for all known register allocation strategies). We tackle this problem with instruction set extensions that allow performing multiple exchanges within a single cycle. This should improve the run-time of the generated code and allows the usage of simple and fast copy-coalescing techniques.

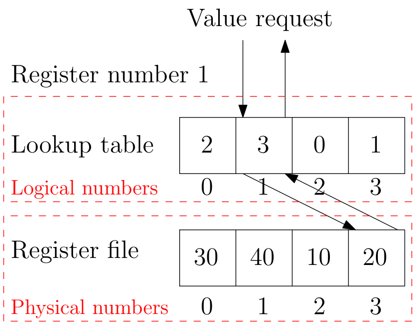


Figure 4.27: *i*-Core physical to logical register mapping

The hardware extensions are available for the *i*-Core now and we developed compiler support for them. We are now in the process of evaluating these instructions with bigger benchmarks and are tuning the instruction encodings for common classes found in the benchmarks.

Publications

[BHTP11] S. Boppu, F. Hannig, J. Teich, and R. Perez-Andrade. “Towards Symbolic Run-Time Reconfiguration in Tightly-Coupled Processor Arrays”. In: *Proceedings of the International Conference on Reconfigurable Computing and FPGAs (ReConFig)*. Cancun, Mexico: IEEE Computer Society, Nov. 30–Dec. 2, 2011, pp. 392–397. ISBN: 978-1-4577-1734-5. DOI: 10.1109/ReConFig.2011.91.

[BH09] M. Braun and S. Hack. “Register Spilling and Live-Range Splitting for SSA-Form Programs”. In: *Proceedings of the International Conference on Compiler Construction (CC)*. Springer, Mar. 2009, pp. 174–189. DOI: `10.1007/978-3-642-00722-4_13`.

[BZB11] S. Buchwald, A. Zwinkau, and T. Bersch. “SSA-Based Register Allocation with PBQP”. In: *Proceedings of the International Conference on Compiler Construction (CC)*. Ed. by J. Knoop. Vol. 6601. Lecture Notes In Computer Science (LNCS). Springer, 2011, pp. 42–61. DOI: `10.1007/978-3-642-19861-8_4`.

[HGG06] S. Hack, D. Grund, and G. Goos. “Register Allocation for Programs in SSA-Form”. In: *Proceedings of the International Conference on Compiler Construction (CC)*. Ed. by A. Zeller and A. Mycroft. Vol. 3923. Lecture Notes In Computer Science (LNCS). Springer, Mar. 2006, pp. 247–262. DOI: `10.1007/11688839_20`.

[Han10] F. Hannig. “Retargetable Mapping of Loop Programs on Coarse-grained Reconfigurable Arrays”. Talk, International Conference on Hardware-Software Codesign and System Synthesis (CODES+ISSS), Scottsdale, AZ, USA. Oct. 26, 2010.

[KHLT11] G. Kouveli, F. Hannig, J. Lupp, and J. Teich. “Towards Resource-Aware Programming on Intel’s Single-Chip Cloud Computer Processor”. In: *3rd Many-core Applications Research Community (MARC) Symposium*. Vol. 7598. KIT Scientific Reports. Ettlingen, Germany: KIT Scientific Publishing, July 5–6, 2011, pp. 111–114. ISBN: 978-3-86644-717-2.

[MTKB+11] P. Marwedel, J. Teich, G. Kouveli, I. Bacivarov, L. Thiele, S. Ha, C. Lee, Q. Xu, and L. Huang. “Mapping of Applications to MPSoCs”. In: *Proceedings of the IEEE International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*. Taipei, Taiwan, Oct. 9–14, 2011, pp. 109–118.

[MLHT+12] R. Membarth, J. Lupp, F. Hannig, J. Teich, M. Körner, and W. Eckert. “Dynamic Task-Scheduling and Resource Management for GPU Accelerators in Medical Imaging”. In: *Proceedings of the 24th International Conference on Architecture of Computing Systems (ARCS)*. Munich, Germany, Feb. 28–Mar. 2, 2012.

[THHSL+11] J. Teich, J. Henkel, A. Herkersdorf, D. Schmitt-Landsiedel, W. Schröder-Preikschat, and G. Snelting. “Invasive Computing: An Overview”. In: *Multiprocessor System-on-Chip – Hardware Design and Tool Integration*. Ed. by M. Hübner and J. Becker. Springer, Berlin, Heidelberg, 2011, pp. 241–268.

[VRMD+10] T. Vander Aa, P. Raghavan, S. Mahlke, B. De Sutter, A. Shrivastava, and F. Hannig. “Compilation Techniques for CGRAs: Exploring All Parallelization Approaches”. In: *Proceedings of the International Conference on Hardware-Software Codesign and System Synthesis (CODES+ISSS)*. Scottsdale, AZ, USA: ACM, Oct. 24–29, 2010, pp. 185–186. ISBN: 978-1-60558-905-3. DOI: `10.1145/1878961.1878995`.

D1: Invasive Software–Hardware Architectures for Robotics

Rüdiger Dillmann, Tamim Asfour, Walter Stechele

Manfred Kröhnert, Johny Paul

In Project D1 we focus on exploring the specific benefits and restrictions of invasive architectures in challenging real-time embedded systems and in particular in humanoid robotics. We focus on implementing a cognitive robot control architecture with its different processing hierarchies, both on invasive TCPA and RISC-based MPSoC. The goal is to explore techniques of self-organization to efficiently allocate available resources for the timely varying requirements of robotic applications. Such a resource-aware computing methodology will lead to better load balancing and efficient utilization of resources compared to traditional resource allocation at compile time. To demonstrate the above aspects we focus on algorithms used on the humanoid robot ARMAR-III which depend heavily on the current task and range from stereo vision, object recognition and obstacle detection to higher level functionality such as object grasping, motion planning or autonomous navigation. During the operation of ARMAR-III the underlying computing architecture faces varying load conditions over time. Depending on the task there are different algorithms either running in combination or sequentially each having a distinct execution time and frequency.

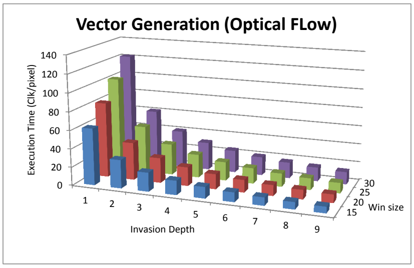
During the first year of the project we analyzed the disparity map and optical flow computation used for robot navigation and the object recognition algorithm (based on Harris features and SIFT descriptors) which helps the robot to recognize various objects around it. The progress of the work is described in detail in the following sections.

Optical Flow on TCPA

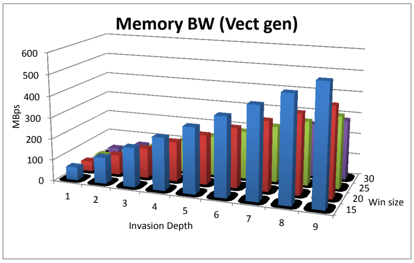
Here we analyzed the census transformation based optical flow implementation where the flow vector pattern can be used for robot navigation or obstacles detection. The three main stages of the algorithm (image smoothening, census transformation and signature generation) were restructured in order to benefit from invasive computing. The application can now self-explore resource availability in the neighborhood, using the resource exploration controller (invasion controller) integrated within each processing element (PE) on the TCPA. The availability of 100s of PEs, along with the dedicated interconnect mechanism between neighboring PEs makes TCPA an ideal platform to implement low level pixel processing applications like optical flow. The algorithm is based on sliding 2D windows and the implementation on TCPA can ensure reduction in execution time, as more resources (PEs) are acquired. Figure 4.28a shows various implementations of the signature generation stage of Optical Flow. Each bar in the bar-graph indicates a possible implementation and it can be seen that the execution time varies based on the PEs acquired. The execution time is represented in clocks per pixel, while the number of PEs used for that configuration is the product of window-size and invasion-depth. The memory bandwidth requirements are expected to scale based on the execution time as shown in Figure 4.28b(for an ASIC implementation running @ 1GHz). More about how to ensure sufficient bandwidth (bandwidth reservation over the iNoC etc.) is a topic of further research. The other two stages of the optical flow work in a similar fashion [PSKA+12]. The complete algorithm runs on the TCPA simulator (Project C2) with static resource allocation as the current version of the TCPA simulator does not support dynamic resource allocation. The application will be implemented in the PAULA programming language (Project C3) once the compiler is ready. Other strategies for optical flow computation will be explored in the coming years using the PAULA language. We also plan to implement more applications like Harris Corner detection, required for object recognition, on the TCPA.

Object Recognition on MPSoC

The possible benefits of implementing Object Recognition algorithm on loosely coupled MPSoC was explored. A methodology which helps the algorithm to decide when to invade/retreat was developed. The application can request extra processing elements if the features to be matched is above a certain threshold. The feature matching can for example be performed on two RISC CPUs simultaneously each performing a kd-tree based search in one half-region of the frame. The partial results can be combined later. Another approach would be to acquire processing elements based on the objects stored in the database and the objects currently detected in the input frame. Most often, in a



(a) Execution time



(b) Memory bandwidth requirements

Figure 4.28: Optical flow on TCPA

real-time implementation, an exhaustive search is not feasible in every frame, as there is a high probability for the previously detected object to exist in the current frame and a low probability for finding a new object in every new frame. Hence the algorithm can be optimized so that an exhaustive search is performed only once in every second, looking only for an already existing object in the rest of the frames. The C++ algorithm from IVT (Integrating Vision Toolkit) was ported to run on a single LEON3 core. More analysis to support multi-threading and support for real-invasion will be explored in the coming years. We plan to come up with a prototype model of object recognition algorithm running on the OctoPOS (Project C1) at the same time using the iNoC (Project B5) and CIC features (Project B3). In the initial stage, the application is expected to perform simple invade/retreat operations on the demonstrator platform provided by Project Z2.

Disparity Map on MPSoC

In order to be able to perform navigation and motion planning tasks the robot needs to be aware of its environment. Especially the distance of surrounding objects is most important in such a scenario. The Disparity Map algorithm provides one possible approach to retrieve this kind of information from stereo camera images. The output of the algorithm is a grayscale image where the gray value of a pixel corresponds to its distance from the cameras.

At first the Disparity Map algorithm from IVT was ported to X10 together with the necessary image loading code. Basic testing was performed during implementation to gather knowledge about the X10 programming language which is going to be used throughout the project.

Once available, the implementation was adapted to the invadeX10 framework provided by Project A1 which enabled the usage of invade/infect/retreat primitives to exploit parallelism. Resulting behaviors were then evaluated using the functional simulator provided by Project C2. These simulation results of the Disparity Map algorithm are presented in [PSKA+12]. It could be shown that the application benefits from the multi-core approach as the execution time decreases with an increasing amount of useable cores. However, these results need to be verified on the demonstrator platform once a matching design is synthesized.

Additionally several methods of passing image data to the application have been investigated enhancing the possibilities of testing vision algorithms on different platforms. Up to now the images can be compiled into the binary to allow testing on platforms without filesystems. In case of filesystems being available it is possible to read image sequences from a directory. Current work focuses on receiving and sending images over TCP/IP sockets which enables accessing camera images from the robot head, even in simulation.

In the near future we will implement a scenario where live images are captured from the robot head and afterwards processed by different vision algorithms. The scenario will initially be evaluated using the simulator. Once the compiler of invasive X10 applications and a prototype of the invasive architecture are available we will use those to evaluate the algorithms on real hardware.

Publications

[PSKA+12] J. Paul, W. Stechele, M. Kröhnert, T. Asfour, and R. Dillmann. "Invasive Computing for Robotic Vision". In: *Proceedings of the 17th Asia and South Pacific Design Automation Conference (ASP-DAC)*. Sydney, Australia, Jan. 30–Feb. 2, 2012, pp. 207–212.

D3: Multilevel Approaches and Adaptivity in Scientific Computing

Hans-Joachim Bungartz, Michael Gerndt

Michael Bader, Andreas Hollmann, Tobias Neckel, Martin Schreiber, Josef Weidendorfer, Tobias Weinzierl

In this project we examine how invasive applications can be realized on standard HPC systems based on iOMP. We also provide numerical algorithms for invasion-enabled hardware by providing scientific algorithms to other projects.

After exploring important constraints for HPC applications we developed an API for invasive OpenMP which is similar to the X10 implementation [HRST+11]. The resource management in iOMP is implemented in a client/server based way.

A generic framework was developed for the tsunami simulation, which should demonstrate the benefits of invasive computing. A basic shallow water simulation was implemented that supports a full-adaptive simulation and uses a split/merge oriented approach for parallelization.

Demonstrator Platform: X10 Applications

For the activities on the demonstrator platform (Project Z2), for the design of an invasion-enabled X10 programming language, and to identify requirements to the hardware and operating systems, D3 provides typical algorithms and algorithmic patterns from scientific computing. In the first year, we provided the following algorithms as X10 implementations:

- An adaptive numerical integration of arbitrary 2D functions on a quad tree structure was implemented with the size of the hierarchical surplus being the stopping criterion for recursion. The required quad tree traversal represents the fundamental algorithmic scheme for all kinds of simulations on recursively adaptive meshes. Since invasive commands are generated down to very lightweight tasks, this algorithm will indicate the overhead of a single invasive command as well scheduling problems for this typically strongly-unbalanced problem.

- A multigrid algorithm based on NAS Parallel Benchmark 3.0 was ported from X10 1.7 to X10 2.2. Due to the high complexity of this code and the fact that it implements a special class for 3D data distribution, it was not possible to utilize the invasive extension of X10 in a reasonable amount of time without rewriting big portions of the source code. This application shows that there are requirements to extend the current invadeX10 implementation to work more transparently. It would be beneficial to have methods to extend and shrink the number of places without having to rewrite parts of the already existing source code. Such an approach would include an implicit redistribution of data in the case of changing number of places. The work on a simplified multigrid using invasive concepts is in progress. It will mimic the behavior of a multigrid with its changing phases of parallelism. The resource requests are coarse grained and occur due to the V-cycle in a cyclic order.

- Our third algorithm is a block-recursive matrix-matrix multiplication. Since the underlying hardware does not offer cache coherency across tiles, a distributed-memory matrix-matrix multiplication based on a space filling curve (SFC) [Bad08] was implemented in X10. Using a software-managed cache to asynchronously preload blocks, one of the major problems of high latencies can be avoided. The workload as well as the data are distributed using the SFC. Special requirements in collaboration with other projects were identified – such as the allocation of local memory on tile, the asynchronous loading of sub-blocks of the arrays and the reusage of claims. This algorithm was also selected for the walk-through steps to clarify the interfaces between other projects (see Section "Collaboration within the SFB").

  All implemented algorithms are steadily updated to the newest invadeX10 API provided by Project A1. Also scalability and efficiency studies on the simulator are expected to give us more insight for further enhancements of the algorithms. Once our algorithms are able to run on the simulator, further performance optimizations in cooperation with other projects can be driven.

iOMP

Recent trends in the architecture of supercomputers as well as in algorithms used in applications running on those systems require a more flexible partitioning. Modern applications are more and more dynamic and consist of phases of more and less parallelism.

The idea of iOMP [BBGH+11] is to bring together the varying demand of parallelism with the extensive parallelism on hardware level via invasive computing. Adaptive applications do not statically allocate the maximum number of resources they can use, but allocate and free cores according to the available parallelism.

A first implementation of iOMP was developed to investigate this new programming approach. The iOMP implementation consists of two parts: the resource manager and the iOMP library. The iOMP resource management approach is implemented as a client/server architecture. There is only one resource manager running on a shared memory system, which has a global overview of all resources.

The iOMP library is implemented in C++ using an object oriented approach that is similar to the invasive X10 implementation. The major difference is that an iOMP program will work only with a single claim. This claim consists of all the PEs currently allocated for the program. Initially it consists of only a single PE but can grow and shrink under the control of the program and the system resource management. At least one PE has to remain in the claim until program termination.

The demands on invasive programming for HPC architectures are identified and specified as constraints. These constraints are: PEQuantity, to control the number of PEs; Scatter and Compact, to control the placing of PEs for higher bandwidth or last level cache sharing; AND and OR, to express intersection and alternatives of constraints.

iOMP applications contact the resource manager via the invade and retreat methods. In the current implementation, the resource manager simply keeps a list of free PEs and returns on request as many free PEs as possible to an application. PEs are assigned to applications for exclusive access, via pinning of threads to the PEs in the claim. The communication between applications and the resource manager is implemented using Linux message queues for interprocess communication (IPC).

The current version of iOMP does not support optimization of locality, that is, NUMA awareness. Resources, that is, PEs, are given to an application for exclusive usage, however, depending on the processor architecture PEs might share the last level cache, the bandwidth to the memory controller and also the interconnection bandwidth. To use resources efficiently it is crucial to minimize remote memory accesses and to minimize cache thrashing effects. In the next year we focus the support of NUMA aware invasion strategies, consider to add nested claims and to keep track of already claimed PEs for better data locality.

A language specification, describing the available constraints and the interaction between the library and the server component, was published on the project's website [HG11].

Tsunami Simulation

To implement and evaluate invasive algorithms and strategies for the demonstrator application of Project D3, a framework was created to run simulations on 2D dynamically adaptive conforming triangular meshes. This framework was used to implement the 2D shallow-water equations (as standard model for tsunami simulation) as well as a discretization with the first-order discontinuous Galerkin scheme. Refinement and coarsening is triggered via a gradient-based error indicator. The adaptive grid is described via a binary refinement-tree structure, such that grid processing relies on (parallel) traversals of this structure.

Our parallel implementation follows a split-and-merge paradigm: partitions are created as subtrees that can be split and merged to stay within a certain range of number of grid cells (Fig. 4.29).

To address the problem of load balancing, we create far more sub-partitions than processors are available on the system. In this way, the system can be flooded with tasks that are then dynamically scheduled to the number of currently available cores. With the existing iOMP implementation we are able to decrease/increase the number of used CPUs for each timestep during a running simulation with the number of CPUs depending on the overall workload.

To allow the immediate execution of arbitrary actions for many processors it is necessary to create those tasks as fast as possible. Instead of storing all sub-partitions in a list, all sub-partitions are stored in a binary tree to start the execution of those actions in parallel. The execution of specific actions is then triggered by a recursion over this tree creating a task within each node.

Another problem arises when executing methods with different signatures (stack setup, triggering sub-partition traversals, water surface setup, communication methods, etc.) in parallel by using the above mentioned binary tree with sub-partition. New C++11 features (e.g., lambda functions) were used to solve this programmability issues.

Besides the OpenMP and iOMP implementation, we also implemented Intel's Threading Building Blocks (TBB) for further scheduling tests on shared memory systems. In the current tests, TBB creates a slight overhead compared to OpenMP's tasking concept but offers more flexible ways to schedule sub-partitions in a probably better way which is part of our ongoing research.

Among other general performance improvements, a code generator was used to avoid excessive branch miss-predictions by unrolling discretized parameters. These branching instructions are used whenever the decision has to be taken to/from which stack the communication data has to be pushed/pulled. However, this creates another kind of challenge of executing a specific recursion method for a given set of method parameters. In the current implementation, this was handled by using a lookup-table for the initial basic domain triangulation offering also lookup-indices for both child recursion methods which is necessary to get a constant run time for the split operation.

An interactive visualization was also implemented using OpenGL which gets relevant for sub-real-time simulation data analysis.

Further information and videos are available at `http://www5.in.tum.de/sierpi/`.

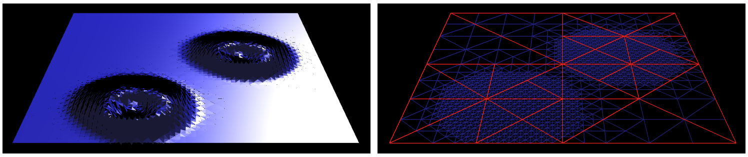


Figure 4.29: Adaptive Shallow Water Simulation, Left handed image - two waves propagating through the water, right handed image: adaptive SFC ordered mesh with sub-partitions in red.

Collaboration within the SFB

First of all we provided algorithms to the compiler project C3 to extend and improve the invasive command space. For example a reduce operation was included in the infect command to generally speed up the function integration implemented in the quad tree traversal. Also other X10 language features like polling mechanisms and message handling were discussed and evaluated from an HPC perspective to support asynchronous loading of blocks.

The matrix-matrix multiplication algorithm was chosen for a walk-through in which the hardware interfaces should be clarified in the next few months to setup a running FPGA system on the simulator. Also this led to intensive collaborations with other projects:

- B3, B5, C1, D1: We provided a simplified and abstract version of the currently existing matrix-matrix multiplication to present an abstract view on the algorithm for hardware developers and to concretize the very basic requirements of the hardware as well as software interfaces.

- B3: Contrary to invasive constraints which are mandatory side-constraints for the creation of the claim, hints were also included in the simplified code in collaboration with B3, which are expected to improve the overall run time by providing more information to the scheduler.

- B3, B5, C1, C3: Also demands on the operating system – for instance, the asynchronous block loading – from the perspective as an application developer were discussed with members of these projects to settle the interactions between the chips, operating system, NoCs, etc.

- D1, C3: In addition to the matrix-matrix multiplication selected for the walk-through, further algorithms, such as those from Project D1, are also involved if we see overlapping areas. Those should be handled in advance to avoid problems due to specialized interfaces created for matrix-matrix multiplication.

Besides the matrix-matrix multiplication, we also evaluate ways to speed up matrix-sub-block multiplications with the help of iCore features together with Project B1.

For linear algebra on HPC systems, we collaborate with Project A3 to enhance the scheduling by evaluating task dependencies with the help of directed acyclic graphs.

iMPI on Intel's SCC

The Single-chip Cloud Computer (SCC) from Intel Labs is an experimental CPU that integrates 48 cores. As its name suggests, it is a distributed memory (DM) system on a chip. To support the invasive programming model the MPI implementation MPICH2 was optimized and extended [CG11]. The result is a library that provides resource awareness through extensions to MPI, while maintaining compatibility with a large number of standard MPI applications.

Three operations were added to the MPI layer: `MPI_Comm_invade()`, `MPI_Comm_infect()` and `MPI_Comm_retreat()`. Additionally, the Invasive Process Manager (IPM) was implemented and provides functionality for resource awareness, topology information and power management. The infect operation is based on spawning new processes on remote nodes.

For MPI applications that only require a short amount of compute time, it was shown that the new process manager provided a significant reduction in the total run time. For the infect operation, it was shown that the initialization procedure of the MPI library takes a significant amount of time. It was also shown that the invade and retreat operations are considerably faster than infect.

Future work for the Invasive Library includes the addition of more resources to be handled. Currently, the cores and the memory available to it are invaded simultaneously; a separation can be made and the interface should allow for their invasion separately. In addition, the dynamic voltage and frequency scaling functionality of the SCC should be exploited.

Peano - Task-based parallelization of patches

The Peano Framework [Wei09], developed by Tobias Weinzierl and others, is an environment for dynamical adaptive simulations on spacetree grids, with its main applications currently in computational fluid dynamics[BGLM+11]. In the last year, the parallel implementation was enhanced towards a hybrid approach that combines MPI for distributed-memory and Intel TBB for shared-memory parallelism.

A new kind of parallelism was developed in the Peano Framework which allows an adaptive setting of the number of threads used to process coarse-granular patches for adaptive simulations. This creates a new way of processing single patches in parallel. Depending on the patch-size, more or less CPUs have to be utilized to avoid task scheduling overheads in order to optimize the overall runtime and efficiency.

Publications

[Bad08] M. Bader. *Exploiting the Locality Properties of Peano Curves for Parallel Matrix Multiplication*. Las Palmas, Aug. 2008.

[BBGH+11] M. Bader, H.-J. Bungartz, M. Gerndt, A. Hollmann, and J. Weidendorfer. “Invasive Programming as a Concept for HPC”. In: *Proceedings of the 10h IASTED International Conference on Parallel and Distributed Computing and Networks 2011 (PDCN)*. Feb. 2011.

[BGLM+11] H.-J. Bungartz, B. Gatzhammer, M. Lieb, M. Mehl, and T. Neckel. “Towards Multi-Phase Flow Simulations in the PDE Framework Peano”. In: *Computational Mechanics* 48.3 (2011), pp. 365–376. URL: `http://www5.in.tum.de/pub/int/compumech_mehletal_2011.pdf`.

[CG11] I. A. Comprés Ureña and M. Gerndt. “Improved RCKMPI’s SCCMPB Channel: Scaling and Dynamic Processes Support”. 4th MARC Symposium. Dec. 2011.

[HRST+11] F. Hannig, S. Roloff, G. Snelting, J. Teich, and A. Zwinkau. “Resource-Aware Programming and Simulation of MPSoC Architectures through Extension of X10”. In: *Proceedings of the 14th International Workshop on Software and Compilers for Embedded Systems (SCOPES)*. St. Goar, Germany: ACM Press, June 27–28, 2011, pp. 48–55. ISBN: 978-1-4503-0763-5. DOI: `10.1145/1988932.1988941`.

[HG11] A. Hollmann and M. Gerndt. “iOMP Language Specification 1.0”. Internal Report. Dec. 2011.

[Wei09] T. Weinzierl. “A Framework for Parallel PDE Solvers on Multiscale Adaptive Cartesian Grids”. Dissertation. München: Institut für Informatik, Technische Universität München, 2009. URL: `http://www.dr.hut-verlag.de/978-3-86853-146-6.html`.

Z: Central Services

Jürgen Teich, Jürgen Kleinöder, Katja Lohmann

The central activities and services in InvasIC are coordinated and organised by Project Z. These activities and services are subdivided into two parts:

The first part is administrative support, organisation of meetings (internal project meetings, PhD student retreats) and assistance for visits of guest researchers and for researchers traveling abroad. Technical support and tools for communication and collaboration are provided as well as support and organization of central publications. Last but not least, financial administration and bookkeeping is one of the central services.

The second part is public relations. Contacts with important research sites were established as well as an international Industrial and Scientific Board. Scientific ideas and results were discussed at various workshops and conferences.

For detailed information on the general idea and organisation of InvasIC as well as on the progress made in the different projects, the InvasIC-website `http://www.invasic.de` was created.

A detailed listing of the scientific meetings and events organised and conducted by Project Z is provided in Part III of this report.

Z2: Validation and Demonstrator

Jürgen Becker, Frank Hannig, Thomas Wild

Srinivas Boppu, Stephanie Friederich, Ralf König, David May, Shravan Muddasani

Evaluating new architectural ideas is a very expensive task. Especially, in the case of a complete computing architecture, as in the case of invasive computing, where interacting software and hardware components are developed, a suitable demonstration environment has to be developed. The goal of Project Z2 is to build a demonstrator for validating the fundamental aspects of invasive computing such as improved resource utilisation and speed-up of applications on the invasive architectures. The work involves coordination, the provision of a necessary hardware infrastructure, integration of the different projects' contributions, that is, hardware and software components, and finally, to build a common FPGA-based demonstrator.

The Synopsys *CHIPit* rapid prototyping system (Figure 4.30) has been chosen as the demonstrator platform. *CHIPit* features a programmable interconnect architecture for automated design implementation and

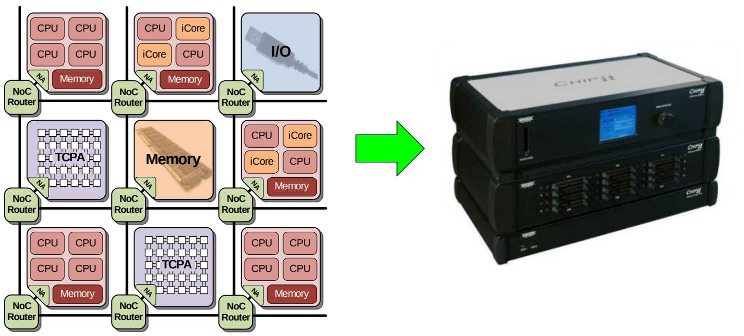


Figure 4.30: *InvasIC Demonstrator*: a concept validation demonstrator with 3x3 tiled array consisting of LEON cores, TCPA, Memory, *i*-Core interconnected through NoC prototyped on *CHIPit* system

enabling emulation-like capabilities. After evaluating different prototyping systems, the choice fell on *CHIPit* systems as they are optimised for transaction-based verification, providing verification and validation throughout the whole SoC and ASIC project cycle. They are based on the latest FPGA technologies and complemented by an integrated set of tools, including comprehensive debug capabilities. The modular *CHIPit* systems are claimed to be useful in diverse verification modes, giving designers improved productivity and flexibility to verify chip implementation and system functionality, thereby significantly reducing overall verification time.

During the last year, we worked on getting familiarised with the *CHIPit* system. We developed knowledge of the system's hardware capabilities, peripherals, extension boards and the associated EDA tools for implementation, debugging and verification of design and software. We attended training sessions from Synopsys on the usage of the *CHIPit* system. In cooperation with Project B2 and Project B3, we successfully prototyped two first designs: a LEON3 based MPSoC and TCPA (2×2 array), which were both demonstrated in the InvasIC "Summer of Code" meeting. This meeting was initiated by Project Z2 in order to obtain a common understanding for all projects, concerning the start-up of an initial demonstrator architecture. There, requirements considering hardware and software have been identified with respect to the needs of the D*x* application projects and an initial demonstrator setup, called *InvasIC Demonstrator*, has been defined. As shown in Figure 4.30, it consists of a 3×3 array of tiles, either RISC compute tiles (standard LEON3, *i*-Core), TCPA compute tiles, memory tiles and one I/O tile.

The LEON3-based MPSoC prototype designs involve the integration of SSRAM, UART and Ethernet extension boards to the *CHIPit* system.

Further on, we will finalise our work to provide a memory controller to access external DDR2 memory. We will also provide a mechanism to load binary executable on this memory and control its execution. Therefore, we are working on an interface based on SCE-MI, a universal interface of the *CHIPit* system, as shown in Figure 4.31. First results are very promising, as

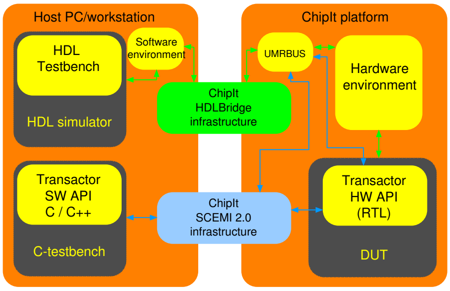


Figure 4.31: *CHIPit* Co-Simulation/Emulation environment

this interface offers a much higher bandwidth and flexibility than other commonly used interfaces like JTAG or UART.

In order to identify possible invasive architecture implementation alternatives on the prototyping platform, we analysed the resource consumption of major hardware components. These results and the description of our prototyping methodology have been published in a paper of the ASP-DAC 2012 together with Project B5 [BFHK+12].

As a first application to be run on this platform for testing the interplay of the involved hardware and software compiler components, a matrix multiplication algorithm has been chosen as its complexity is manageable and which helps to identify and solve open questions. This "start-up" demonstrator platform including the matrix multiplication application is targeted for second quarter of 2012.

In order to clarify the interfaces between interacting components of the platform already during conceptual phase the description of an abstract "walk-through" of the matrix multiplication has been started on a project wiki. The "walk-through" will capture the interaction among hardware and software instances on a high level of abstraction and thus help to come to a common view of all relevant interfaces between components provided by the involved projects. In this way, potential interoperability problems should be identified already in the design phase and be avoided in the integration phase. Currently, the "walk-through" is on the way to be finalised.

The next steps towards the realisation of the "start-up" demonstrator will be to update and concretise the roadmap in a common meeting with all projects. There, a schedule for all contributions will be specified. In the sequel, Project Z2 will supervise and enforce this schedule and co-ordinate the integration of of the provided components into the demonstrator. Regular meetings with other projects will be held to keep track of the defined milestones.

Publications

[BFHK+12] J. Becker, S. Friederich, J. Heisswolf, R. Koenig, and D. May. "Hardware Prototyping of Novel Invasive Multicore Architectures". In: *Proceedings of the 17th Asia and South Pacific Design Automation Conference (ASP-DAC)*. Sydney, Australia, Jan. 30–Feb. 2, 2012, pp. 201–206.

5 Working Groups

WG1: Working Group Architecture

Coordinator: Andreas Herkersdorf

The main goal of the Working Group Architecture is to promote integration and co-operation among all architecture projects (project area B) as well as representing architectural aspects and perspectives towards other invasive computing domains (project areas A, C and D), which more or less all have (to a varying degree) the need to interact with B projects. This also implies to resolve possible problems or incompatibilities, either among B projects or between architecture projects and cooperating non-B projects.

The major objective in 2011 was to consolidate the definition of a tiled invasive architecture platform which allows a flexible composition of optimised, application-specific invasive architecture instantiations. A common understanding between processor hardware related projects with the run-time system and agent layer (Project C1), the simulation infrastructure (Project C2), the compiler support (Project C3), language definition (Project A1) and applications (Project D1, Project D3) is necessary for a successful evaluation of the InvasIC concepts in a first demonstrator in 2012 as well as for the planned ASIC evaluation in the next InvasIC program phase.

Tiled Invasive Architecture

The overall tiled architecture platform is depicted in Figure 5.1. Hardware resources are partitioned into tiles which are connected by an invasive network-on-chip (*i*NoC). There are four types of tiles: i) invasive TCPAs compute tiles perform stream-based computations on a massively parallel array of low-complexity processing elements, ii) RISC compute tiles perform general-purpose computation on openly available, possibly invasive enhanced standard SPARC V8 cores, iii) I/O tiles serve as interfaces towards external peripherals (e.g., video, IP networking, serial port, debugging) and iv) memory tiles that either provide access to external DDR memory or comprise on-chip SRAM memory.

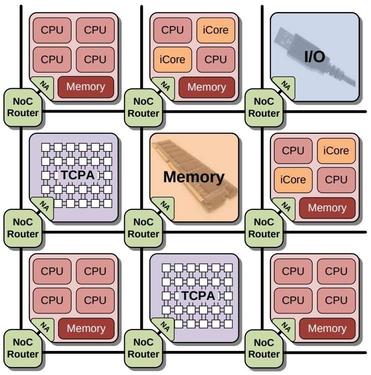


Figure 5.1: Tiled InvasIC Architecture

Memory Hierarchy

The three InvasIC levels of addressable memory are defined as follows:

- *Scratchpad* – each core has its private scratchpad memory which allows for low-latency accesses managed by the application programmer. It is non-cached. Due to its small size and latency requirements it will be implemented using the BlockRAM on the demonstrator prototype (Project Z2). The ASIC realisation of an invasive many-core processor will consist of SRAM-type memory.

- *Tile Local Memory (TLM)* – each tile contains a TLM that is shared among all cores on that tile. Regions of the TLM may be invaded and accesses to the TLM are generally cached (although certain areas may be configured as being non-cacheable at run time). Cores from remote tiles can access the TLM using Message Passing Services through the on-chip network. However, cache coherency will only be guaranteed for cores within one tile (or X10 place, if the X10 place is bounded by the borders of a compute tile).

- *Partitioned On-Chip and External Memory* – this type of memory can be accessed through message passing services over the *i*NoC towards either the on-chip SRAM tiles or via a DDR controller

START	END	Type	Remarks
SCRATCHPAD_START	SCRATCHPAD_END	Scratchpad	Local variables, stack, frequently used math library
TLM_START	TLM_END	Tile Local Memory	Local, global, shared code/data within the tile
OCM_START	OCM_END	On-chip Memory	On-chip SRAM memory tiles
EXTMEM_START	EXTMEM_END	External Memory	Global, shared, code/-data across tiles
PERIPHERAL_START	PERIPHERAL_END	Peripherals	UART, Timers, GPIO, VGA
DEBUG_START	DEBUG_END	Debug Support Unit	JTAG debugger interface

Table 5.1: Sample memory map of a tiled invasive many-core

towards off-chip SDRAM modules. Note, although these forms of memories can be accessed from all compute tiles, the address space is partitioned among different X10 places (one or several cores of a compute tile). However, in InvasIC phase 1, there is no hardware mechanism provisioned to supervise the validity (non-overlap in address space access from different places) of partitioned memory accesses.

The only non-addressable type of memory is the:

- *Core Cache* – each core has an instruction- and data-cache. Again, coherency is guaranteed among cores within one tile only. This is realised by the write-through cache update policy and AHB snooping. Inter-tile coherency – if required – needs to be handled by software.

A sample memory map which corresponds to the above specification is shown in Table 5.1. Note, B4 monitoring registers, general control registers and network buffers as part of the network adapter will as well be mapped in the address space but are not reflected in Table 5.1.

RISC Tile Configuration

RISC compute tiles consist of four types of resources: i) computation cores, ii) TLM, iii) C*i*C, and iv) network adapter and are connected via a 32-bit AMBA High-performance Bus (AHB). An example for the configuration of such a compute tile is shown in Figure 5.2.

In general, compute tiles may contain a variable number of cores. The current working assumption for the demonstrator is that a RISC compute tile has 3 native LEON SPARC V8 cores, and a fourth core which is either an *i*-Core or another LEON core. Cores may be furnished with a floating-point unit (FPU) although area constraints with respect to the demonstrator will limit the number of FPUs. The exact number will be decided upon in cooperation with Project Z2 and the application projects (project area D) when the first common demonstrator platform will have reached its implementation phase. All cores comprise an instruction cache and data cache. Their sizes will be investigated by means of the first version of the common demonstrator setup.

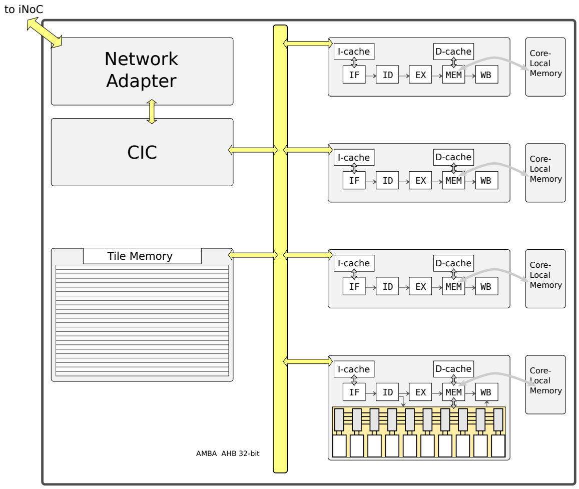


Figure 5.2: RISC Tile with 3 LEON cores and 1 *i*-Core

A compute tile will include TLM that is implemented in two technologies for the first common demonstrator scenario (coordinated by Project Z2):

- BlockRAM-based, with a size of up to several 100 KBytes, with a low latency, and
- SRAM-based, with a size of up to several MBytes, but with a higher latency.

WG1

This provides a high-bandwidth between the *i*-Core and the BlockRAM-based TLM allowing its reconfigurable fabric to perform with high efficiency without causing contention when other tiles on the core access the TLM. The BlockRAM-based TLM may be accessed by any other core on the tile as well as through an arbiter resolving simultaneous accesses to the *i*-Core fabric and other cores. Both TLM blocks form a contiguous address range, allowing software to be written independently of the underlying implementation (i.e., the complete TLM could be implemented in one technology).

In order to determine a computation resource for an i-let, the iRTSS first selects the tile on which the i-let will run. The C*i*C (Project B3) of this tile then assigns the i-let to a specific core. The C*i*C is also responsible for gathering monitor data from the cores, preprocessing it, and providing hardware support for the virtual power gating (ViPG in Project B3). The C*i*C also serves as the interface between the AHB and the Network Adapter (which connects the tile to the *i*NoC) of the tile.

In Summary

The intensive exchange and collaboration among project area B as well as between the B and non-B projects settled the definition and specification of central aspects of a tiled many-core platform which is adequate for demonstrating the benefits of the invasive computing paradigm. This statement does not preclude minor adjustments and enhancements to the InvasIC hardware architecture when necessary.

Meetings and Talks

Jan. 19, 2011 Meeting WG Architectures, Karlsruhe

Jun. 22, 2011 Talk from Tim Mattson, Intel USA, in Karlsruhe on “Charting a course to our many-core future: HW, SW and the parallel programming problem”

Aug. 1, 2011 “Summer of Code” in Karlsruhe

Oct. 19, 2011 C1-B3-B5-D3 Meeting at TU München to detail C*i*C-*i*RTSS interaction

In general, bilateral meetings among different projects, and meetings within one projects (but different sites) are not listed.

WG2: Working Group System Software

Coordinator: Wolfgang Schröder-Preikschat

Working group AKSS (abbr. ger. "Arbeitskreis Systemsoftware") is involved in the entirety of all programs which control function and operation of the invasive computing system on behalf of invasive-parallel application processes. It provides a forum for hardware and (application/system) software developers to jointly discuss topics related to these programs from the perspective of the individual research areas A, B, C, D and Z. In addition, AKSS bundles requests and requirements related to system software as stated by the projects, produces proposals for solutions of problems aroused in the outer field of hardware, compiler and application software, and communicates achievements, recommendations and obligations back to the projects. In the reporting year, the following meetings took place:

14.12.2010 FAU Erlangen, host team Schröder-Preikschat[1]

13.05.2011 TU Munich, host team Herkersdorf

19.09.2011 FAU Erlangen, host team Schröder-Preikschat

Key aspects of activity of the working group so far was in the definition of common terms, consideration of the application programming interface (API), specification of the memory hierarchy including their programming implications, creation of a plot plan of the (partitioned) global address space and its meaning, and discussion of the so far identified constraints of invasion. Materials in preparation for the working group meetings as well as for the documentation of their results are maintained at `https://invasic.informatik.uni-erlangen.de/intern/wiki/ak_systemsoftware`. Some of the results generated by the working group that have a particularly broad relevance for the overall project are briefly summarised in the following paragraphs.

[1]As the annual report of 2011 is the first one released, the (kick-off) meeting of the working group held in the preceding year is mentioned here for the sake of completeness.

Common Terms A piece of program subjected to parallel processing according to the paradigm of invasive computing is referred to as an "invasive-let": in short *i*-let.[2] Depending on the level of abstraction considered, different *i*-let entities and associated properties are distinguished:

candidate (a) prospect out of a family of algorithms for the same problem to be solved, (b) potential cause of a specific operating mode of the (parallel) processor as to be enforced by *i*RTSS and (c) possibly represented and maintained as a separate source module.

instance (a) medium of activity of an invasive-parallel program, (b) specification of a virtual processor for it and (c) possibly represented and maintained as a separate object module.

incarnation (a) characteristic of the mode of operation to be realised by *i*RTSS, (b) ground anchor for the resources virtually needed for making progress in parallel processing and (c) possibly represented and maintained as a separate load module.

execution (a) actual disposition of a portion of an invasive-parallel program running on a real processor, (b) effective unit of processing implemented in soft-, firm-, or hardware (c) associated with a dedicated memory image.

Given these notions of *i*-let and taking an operating system's point of view, candidates and instances are user-level entities while incarnations and executions are system-level entities. At system level, two more terms have been established which manifest in corresponding *i*RTSS abstractions:

claim designates a particular set of hardware resources made available to an invading process on demand and according to selected constraints.

team designates a particular set of *i*-let entities (i.e., incarnations) associated with a specific claim.

These two abstractions aid application-level processes in the description of (static/dynamic) resource demands, the indication of the operating mode of the computing machine and the modelling of a certain run-time behaviour of the constituting *i*-lets.

[2]This conception goes back to the notion of a "servlet", which is a (Java) application program snippet target for execution within a web server.

Application Programming Interface Two sorts of APIs will be supported by *i*RTSS: an X10-oriented on the one hand and a design for C/C++ on the other hand. The latter appears to be the default API as *i*RTSS will be predominantly implemented in C/C++. The former version is due to the case that constraints for invasion are described (only) by means of a class hierarchy expressed in X10, which necessitates run-time conversion into the C/C++ language domain. As these constraints control the process of resource allocation provided by the agent system of *i*RTSS, their anyway needed interpretation and filtration will be carried out by a system-level component implemented in X10. This component takes care for turning functional as well as non-functional requirements as expressed by the constraints into directives to the operating units of *i*RTSS and, thereby, also bridges the language barrier.

Memory Hierarchy A multi-level memory system is assumed as indicated in Figure 5.3. Overall, it provides for a (physically) partially distributed organisation in terms of the core- and tile-local units. Except for core-local memory (*log* in logical terms of *i*RTSS abstractions and *scratchpad* in physical terms of the hardware), all other memory building blocks are globally addressable within a common *partitioned global address space* (see also Figure 5.4). Upon read or write access, data originating from the lower three levels of the memory hierarchy will be cached close to the core having issued that very access.[3] In particular this also holds for data from remote tile-local memory of the same hierarchy level. Writing policy of cached data will be write-through.

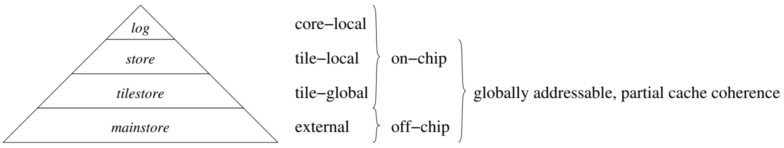


Figure 5.3: Memory organisation. Capacity and latency increases from top to bottom. The top-most level (*log*) is private to a core, whereas all other levels are public to any core. In addition, the upper two levels constitute decentralised and the lower two ones centralised memory. The levels below *log*-level are subject to caching. Cache coherence is provided only for cores of the same coherence domain (i.e., cores of the same tile).

At hardware level, the system assumes *partial cache coherence*. On the one hand, core-local memory will generally be excluded from the

WG2

[3]Data originating from core-local memory will not be cached, as, from the hardware's point of view, the core-local memory roughly compares to addressable cache memory.

sphere of the cache coherence protocols carried out in hardware. On the other hand, hardware-maintained cache coherence will be effective only for those cores which reside within the very same tile and, thus, belong to the same *coherence domain*. This defines a limiting factor when process migration comes into play, be it for load balancing or fault tolerance: Pulling a single process out of its tile-local *team of processes* and relocating it to a different scope also implies the change of a coherence domain. In such a case, the complete process team will have to be migrated in order to maintain the coherence domain for all related processes. If needed, more comprehensive memory semantics will have to be provided by dedicated software means of *i*RTSS, the X10 run-time system, or even the actual application program.

Global Address Space The different memory areas constituting the memory hierarchy are combined in a single partitioned global address space, as shown in Figure 5.4. In this concept there are two noticeable aspects:

1. Every single tile-local store is assigned to a unique (physical) address range within the global address space. This makes tile-local store globally addressable from any core.
2. Additionally, a tile-local store is also assigned to a (logical) address range within the global address space. This address range is the same for each of the tile-local stores. Motivation behind this second way is support for process migration.

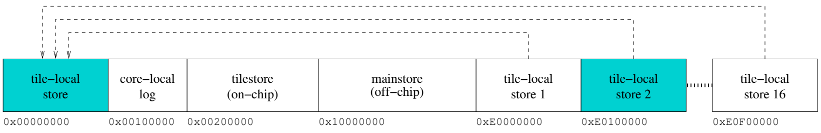


Figure 5.4: Partitioned global address space. The (physical) start addresses assigned to each of the partitions are exemplary and given for a better ease of understanding, only. Highlighted is the double mapping of the address range of tile-local store 2. All cores of this tile have their tile-local memory starting at logical address `0x00000000` and physical address `0xE0100000`. By means of their physical address ranges are all tile-local memory partitions accessible by all cores in the system.

Double mapping of the address range of a tile-local store can be done either hard-wired or by means of a programmable memory management unit (MMU). In the latter case, *i*RTSS will have to take care of the mapping from the logical address range of a tile-local store to its physical complement upon process creation, on the one hand, and shifting of these mappings upon process switches, on the other hand. How this double mapping shall be put into practise is still for further study. In any case, hardware support (namely either hard-wiring or MMU) is required to support this approach.

Constraints of Invasion The paradigm of invasive computing relies on *resource-aware programming* of the underlying computing machinery. Thereto, an application process requests from *i*RTSS the (physical or virtual) resources needed for making progress according to a specific algorithm or solution procedure. This request is enriched by means of constraints to indicate the functional and non-functional properties that should be fulfilled by the system for a certain phase of parallel processing. The outcome of such a request will be a *claim* of resources that *i*RTSS can offer at a given point in time. The properties of the claim set out by *i*RTSS for the respective application process may differ in functional as well as non-functional terms compared to the properties specified in the request. In this context, resource awareness means that, by reflecting the claim properties, the application process adapts itself to the actual resource allocation decision made by *i*RTSS. This may add up to select a different *i*-let candidate or put together the appropriate *i*-let instance in order to match the capabilities and potentials of the claim provided.

At the time being, the constraints are described (1) in X10 and (2) by means of a class hierarchy, whereby the latter then will be "interpreted" by the *i*RTSS agent system. A constraint may be of a scalar or composite data type, it may also be a performance curve, for example, indicating a whole range of hints to the operating units of *i*RTSS (i.e., the agents) responsible for resource allocation. The discussion on these constraints started. Open issues relate to a classification scheme, the mapping of constraints onto operating modes of *i*RTSS and the hardware, and the form of description given to a C/C++ API.

WG3: Working Group Language and Applications

Coordinator: Gregor Snelting

Goals of the Working Group

Resource-aware and invasive programming is not possible without language support. The language must support fundamental invasive operations (invade infect, retreat). It must provide interfaces for dynamic resource parameters and system state. It must also exploit existing technology for parallel programming. In particular, the language must support distributed, heterogeneous memory architectures. The language must be exercised and validated on real algorithms and problems. Expressiveness and usability must be evaluated for different types of potential target architectures, ranging from MPSoC architectures to HPC machines.

The working group "Language" coordinates all research activities with respect to language development, compilation, application, and validation. It starts out from a fundamental design decision: In 2010 it was decided to base the invasive computing language on X10. X10 is the only available language and compiler, which supports heterogeneous, distributed address spaces. It was further decided to implement invasive constructs and commands not as additional syntax (as described in the original proposal), but in form of a framework and library classes.

Details of the language design, as well as examples and rationale, are described in the report of Project A1. The design and status of the InvasIC Compiler is described in the report of Project C3.

Related Projects

The core projects contributing to the working group are A1 and C3. Project A1 defines and validates the language and the framework, respectively. This includes reference examples and case studies for invasive programming. Project C3 develops the compiler; it generates code for SPARC processors and will use specific optimisations for invasive constructs, based on the libFirm code optimisation framework.

There are important interfaces to the simulator (Project C2) and to the operating system (Project C1). The simulator allows to execute invasive programs on traditional hardware for purposes of study and evaluation. In fact the simulator can execute programs written in the invade X10 framework. The interface to the *i*RTSS includes support for resource-aware programming (in particular functions for monitoring the hardware state, including core temperature and availability), as well as support for fundamental invasive constructs such as invade, infect, and retreat. Applications in Project D1 and Project D3 are expected to use the invasive X10 framework, thus they are integrated into the language design and validation process. Even the architectural projects in area B need coordination with the working group, because fundamental questions concerning, for instance, the memory model or invasion-specific instructions affect the design of language, compiler, and run-time system; but cannot be solved by Project A1, Project C3 and Project C1 alone.

The working group is developing reference examples for invasive programming (see below). Eventually, these examples must run on the demonstrator platform (Project Z2).

Developing the invasive X10 framework

In autumn 2010, the working group was confronted with the task to "market" the language concept in the whole project, and to coordinate all language implementation and validation efforts. It was decided to use a bottom-up approach with iterated and refined feedback loops, which enabled all projects to participate in the design and validation of the language and the framework, respectively. In fact, only the participation of *all* projects in language design and programming examples will generate a *common understanding of invasive programming*; which in turn is indispensable for the success of the whole research centre.

Thus a call for pseudo-codes was published, where all projects where encouraged to submit invasive algorithms or programs; these programs or pseudocodes were to represent the project's understanding of invasive programming. In addition, an X10 Tutorial took place in Erlangen, 12.11.2010. This successful tutorial was delivered by Prof. von Praun, a former co-developer of X10. It attracted 38 attendees from all projects.

All projects submitted invasive pseudocodes, which were discussed in a Language Workshop at KIT, 10.12.2010. Open questions included: invade vs. X10 "at"; memory model; "reinfect" vs. "reinvade"; "assort";

use of load curves for resource-aware programming; invasion of memory/communication resources; and others. The discussion results where incorporated into the first version of the framework in X10, which includes the invasive command space. The framework, including a description and design rationale, was released in spring 2011.

In the next iteration, all projects were asked to submit their former pseudocode as X10 code using the new invasive framework. Indeed, Projects A1, B1, B3, B5, C2, C3, D1, D3 submitted X10 programs. The A1, C2, D1, D3 programs were able to be run within the simulator (Project C2). All X10 programs were discussed at a workshop at KIT, 4.7.2011. This discussion further refined the definition of framework and invasive command space. An interface to iRTTS is currently being defined for resource-aware programming. Concerning the memory model and memory consistency, it was decided that on the language level, there will only be distributed memory, while on the hardware level, global memory is a possible option. It will be the compilers job to exploit this option if possible.

Reference examples

At the above-mentioned workshop, it was decided to investigate three reference examples in more detail. These examples – matrix multiplication, a streaming application, and a disparity map – are to be implemented manually through *all* levels of the invasive soft- and hardware chains, in order to validate the overall invasive design and integration. We consider the reference examples to be indispensable not only for validation of the language and framework, but also for a successful integration of the whole research centre; as well as essential for the evaluation by the DFG in early 2014. Thus eventually, the reference examples will have to run on the demonstrator platform (Project Z2).

Outlook

There are not enough resources to tackle all three reference examples at the same time. Thus, the plan for 2012 is to first concentrate on the matrix multiplication example, and use the experiences for the other reference examples. Project Z2 will coordinate the matrix example; the Working Group Languages will coordinate the others. A workshop in spring 2012 will collect the experiences and prepare the complete implementation of other reference examples (not necessarily the ones mentioned above). These must not only exploit the framework and hardware results, but must also demonstrate resource-aware programming, dynamic load balancing, fault tolerance, and other invasive features.

```
val constraints = new AND();
constraints.add(new TypeConstraint(PEType.RISC));
constraints.add(new PEQuantity(MAX_CORES));
val claim = Claim.invade(constraints);

// initialize workload
val globalWorkload = globalInit(); // generate random noise
val localWorkloadMap = X10ArrayWrapper.distributeArray
                       (globalWorkload.region, claim);

// initialize result data structures
val globalResult = new Array[int](MAX_VALUES);
val localResultMap = X10ArrayWrapper.duplicateArray
                     (globalResult.region, claim);

// i-let definition
val code = (id:IncarnationID) => {
        val localWorkload = localWorkloadMap(id.ordinal)()()
                            as Array[int](2){rect==true};
        val localResult = localResultMap(id.ordinal)()()
                            as Array[int](1){rect==true};
        // histogram calculation
        for([i,j] in localWorkload)
                localResult(localWorkload(i,j))++;
};

claim.infect(code);

// collect the results
for(key in localResultMap.keySet()) {
        val localResult = localResultMap(key)()()
                          as Array[int](1){rect==true};
        for([i] in globalResult)
                globalResult(i) += localResult(i);
}

claim.retreat();
```

Figure 5.5: An invasive X10 program to calculate a histogram, exploiting the invasive framework developed by Project A1.

III

Events and Activities

Summary

The central activities and services in InvasIC are coordinated and conducted by Project Z.

In the following sections we summarise the major events and activities between July 2010 and December 2011. These events include Internal Meetings (Section 6), the DRR 2011 (Section 7) and Trainings and Tutorials (Section 8). Last but not least, we present the current constitution of the Industrial and Scientific Board in Section 9 and conclude with further scientific activities in Section 10.

Figure 5.6: At the annual meeting in Lauterbad 2011

6 Internal Meetings

Collaboration between the researchers of the three sites Karlsruhe, München and Erlangen is essential for the success of the TCRC 89 - InvasIC. Within the 18 months of the existence of InvasIC, researchers met at the following opportunities:

Event	Date	
InvasIC Kickoff Meeting	June 28. 2010, Erlangen	Elections of the board of the TCRC
Annual Meeting 2010	Sep. 15/16. 2010, Karlsruhe	The first annual meeting after the start of the project was used for an initial synchronization of all projects, a report of the work done so far and discussions about further direction.
Workshop Language Group	Dec. 10. 2010, Karlsruhe	In the language group workshop at Karlsruhe the invasive language constructs were discussed.
Workshop Systemsoftware Group	Dec. 14. 2010, Erlangen	Open questions from the language group workshop were discussed, application- and hardware developers addressed their needs to an invasive system software.
Workshop Architecture Group	Jan. 19. 2011, Karlsruhe	At the meeting of all architecture projects topics concerning the hardware architecture were discussed.
Semi-annual meeting 2011	Feb. 13-15. 2011, Munich	The semi-annual meeting was used for a brief review of scientific research from all projects within the past half year.
Workshop Systemsoftware Group	May 13. 2011, Munich	In the systemsoftware working group workshop in Munich researchers from different projects discussed the InvasIC memory hierarchy.
Workshop Language Group	July 4. 2011, Karlsruhe	With more than 30 participants, the first invasive language extension of X10 v0.1 was introduced by the Snelting group
InvasIC “Summer of Code”	Aug. 1/2. 2011, Karlsruhe	At the first InvasIC Summer of Code workshop, 27 researchers from all project areas came together and discussed common scenarios for the validation and demonstration of Invasive Computing
Workshop Systemsoftware Group	Sep. 19. 2011, Erlangen	The focus of the discussion at the systemsoftware working group workshop was on memory hierarchy and claim constraints.

Annual Meeting 2011	Oct. 3/4. 2011, Lauterbad	51 InvasIC scientists met in Lauterbad (Black Forest) to review and discuss the progress made in the last year
Doctoral Researcher Retreat	Oct. 5-7. 2011, Lauterbad	The first Doctoral Researchers' Retreat (DRR) was held as a continuation of the annual meeting.

Figure 6.1: Discussions at the Summer of Code 2011 in Karlsruhe

Figure 6.2: Semi-annual meeting 2011 in Munich

7 DRR 2011 in Lauterbad

To support PhD students with their work, a yearly PhD student retreat is organized.

The first SFB/TR 89 “Invasive Computing" Doctoral Researchers’ Retreat (DRR) was held as a continuation of the annual meeting from Oct. 5–7, 2011. The event was organized by five *Nachwuchswissenschaftler* Daniel Lohmann, Frank Hannig, Josef Weidendorfer, Lars Bauer, Michael Hübner, who participated together with 35 doctoral researchers.

All participants considered the retreat as very productive and helpful for their daily work in the SFB/TR. As a consequence, the participants suggested to implement a semi-annual schedule for the DRR; two doctoral researchers from each location volunteered to organize the next retreat.

Goals

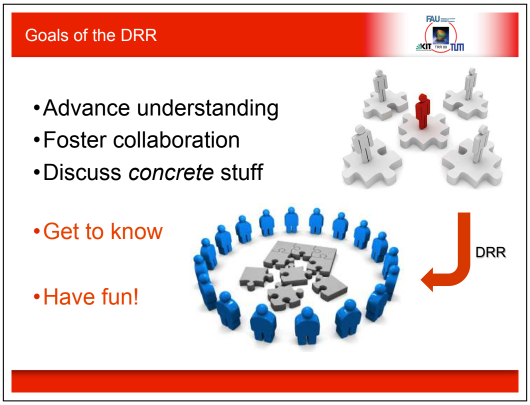
Figure 7.1 lists the primary goals of the first DRR, which can be summarized as *community building*: These goals were motivated by our observation that the doctoral researchers do not yet know and trust each other well enough to animate Invasive Computing by low-barrier, direct, every-day interaction with their colleagues from different projects and locations.

In this realm the motto of the first DRR was *“Get Together and Get to Know*". It deliberately embeds a *social* and a *factual* facet of interaction; both of which were to be improved during the DRR in order to foster cross-project collaboration.

Concept

The resulting concept accounted for these goals by integrating elements of experience education and social-skills training with factual technical/scientific work. The basic idea was to use factual work not only as a first-class goal in its own realm, but also as a “vehicle" to foster social interaction and development of trust by letting the participants work and interact with each other in a field they feel secure about.

The major instrument was *group work assignments* to be carried out in small (and frequently changing) groups of 2–5 participants. The groups were selected by *random* to overcome the human preference for interaction with people one already knows. However, for all group work selections there was an exception rule: If a group of doctoral


Goals of the DRR

- Advance understanding
- Foster collaboration
- Discuss *concrete* stuff
- Get to know
- Have fun!

Figure 7.1: Goals of the 2011 DRR

researchers from at least three different projects organized themselves to “get something done", these participants were exempted from the random selection process.

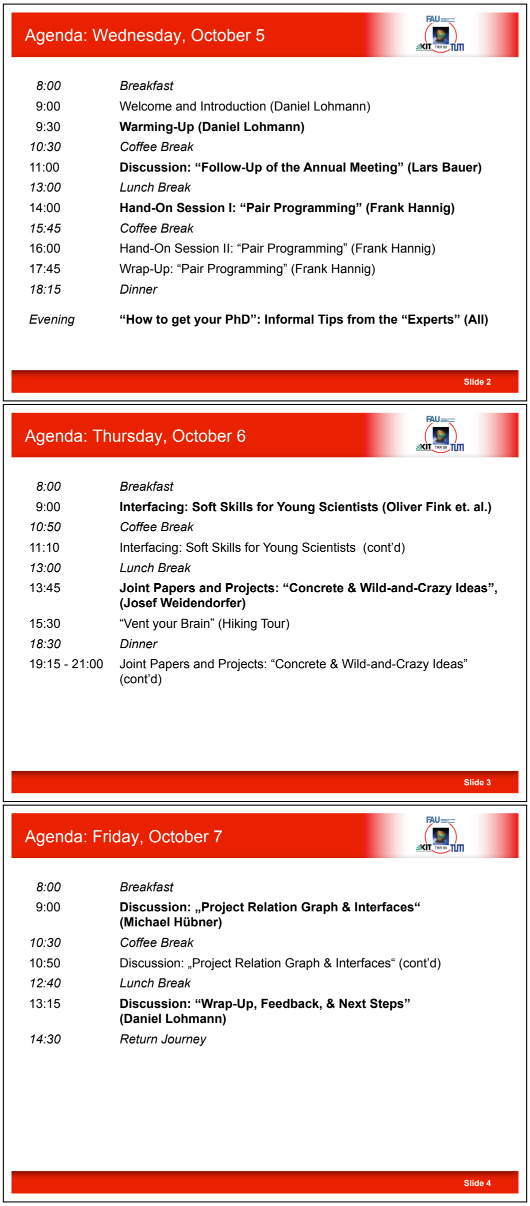
Program

Figure 7.2 shows the resulting program for the 2011 DRR. The major elements were:

Warming-Up. A short “breaking the ice" session to open the participants for interaction with their colleagues and to get to know each other by six rounds of “speed dating". The process was as follows:

1. Wander through the room, look for the guy you know *least*.
2. Get to know within five minutes.
3. When the bell rings, proceed with Step 1.

Follow-Up of the Annual Meeting. A discussion round to clarify and discuss open questions from the annual meeting. Participants were randomly distributed into groups of five and given 10 minutes to write down questions. These questions were then collected, grouped and discussed with the audience. The *Nachwuchswissenschaftler* moderated the process, but avoided to answer the questions in favor of letting the group itself figure them out. Thereby, several doctoral researchers could establish *themselves* as experts for their respective field and project.



Agenda: Wednesday, October 5

8:00	*Breakfast*
9:00	Welcome and Introduction (Daniel Lohmann)
9:30	**Warming-Up (Daniel Lohmann)**
10:30	*Coffee Break*
11:00	**Discussion: “Follow-Up of the Annual Meeting” (Lars Bauer)**
13:00	*Lunch Break*
14:00	**Hand-On Session I: “Pair Programming” (Frank Hannig)**
15:45	*Coffee Break*
16:00	Hand-On Session II: “Pair Programming” (Frank Hannig)
17:45	Wrap-Up: “Pair Programming” (Frank Hannig)
18:15	*Dinner*
Evening	**“How to get your PhD”: Informal Tips from the “Experts” (All)**

Slide 2

Agenda: Thursday, October 6

8:00	*Breakfast*
9:00	**Interfacing: Soft Skills for Young Scientists (Oliver Fink et. al.)**
10:50	*Coffee Break*
11:10	Interfacing: Soft Skills for Young Scientists (cont’d)
13:00	*Lunch Break*
13:45	**Joint Papers and Projects: “Concrete & Wild-and-Crazy Ideas”, (Josef Weidendorfer)**
15:30	“Vent your Brain” (Hiking Tour)
18:30	*Dinner*
19:15 - 21:00	Joint Papers and Projects: “Concrete & Wild-and-Crazy Ideas” (cont’d)

Slide 3

Agenda: Friday, October 7

8:00	*Breakfast*
9:00	**Discussion: „Project Relation Graph & Interfaces“ (Michael Hübner)**
10:30	*Coffee Break*
10:50	Discussion: „Project Relation Graph & Interfaces“ (cont’d)
12:40	*Lunch Break*
13:15	**Discussion: “Wrap-Up, Feedback, & Next Steps” (Daniel Lohmann)**
14:30	*Return Journey*

Slide 4

Figure 7.2: Agenda of the 2011 DRR

Pair Programming. An engineering-oriented group-work session. Randomly selected pairs of doctoral researchers worked together according to the rules of Pair Programming on their project-related topics: One participant as the “driver" and one as the “observer"; after two hours the roles were switched. Thereby, the participants got a deeper insight and understanding in how and what their colleagues from (possibly very distant) projects are actually doing and could provide each other with a “fresh view" on the topic and its challenges.

Soft-Skill Training. An experience-education seminar about the essential elements of team work and team building, held by professional trainers.

Joint Papers and Projects: Concrete & Wild-and-Crazy Ideas. A research-oriented group-work session. Participants were randomly selected into teams of three and asked to come up with a “concrete" or “wild and crazy" idea for a joint paper. In every group there were two participants who discussed a potential idea and poured this into a paper abstract, whereas the third had to play the role of the *advocatus diaboli* to challenge the concept.

Project Relation Graph. A whole-group discussion round to consolidate the findings about the relations and interfaces between the projects examined during the DRR. Results were written down as a project relation graph.

Wrap-Up, Feedback & Next Steps. A short “touchdown" session to wrap up and gather feedback: Participants were randomly distributed into groups of five and given 15 minutes and three sheets of paper to discuss and write down what (1) they liked, (2) they disliked, and (3) they missed on the DRR. These points were then collected and discussed with the audience to generate structured input for the organizers of the next DRR. Additional critique and ideas were collected in a following open feedback round.

Results

For all sessions a short description of the factual results can be found in the SFB/TR-internal wiki[1].

[1] `https://invasic.informatik.uni-erlangen.de/intern/wiki/mitgliederversammlungen/drr2011/`

Overall, the first Invasive Computing DRR was a clear success. The doctoral researchers experienced it as *very* productive for their work within the SFB/TR and especially pointed out that it helps to discuss their topics “on equal terms", that is, with other doctoral researchers. They suggested to have a DRR not only once, but twice a year with the next DRR as addendum to the semi-annual meeting in February 2012.

Next Steps

The next DRR will be organized by the doctoral researchers themselves: Two volunteers from each location are currently organizing the DRR in Febuary 2012 in Obertrubach. The volunteers to organize the event were quickly found (names are listed in the wiki). Hence, the *Nachwuchswissenschaftler* will step back, however will still be available as mentors.

Figure 7.3: Soft Skills for Young Scientists

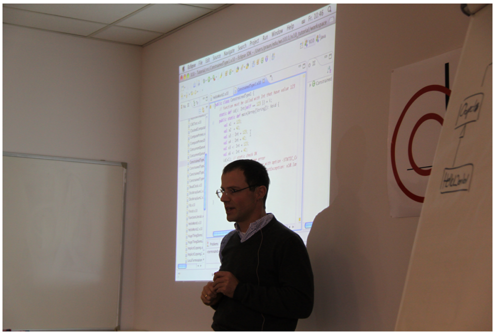


Figure 7.4: X10 Tutorial given by Prof. Christoph von Praun

Figure 7.5: Participants of the Gender-Training during role-playing

Figure 7.6: Dagstuhl Seminar co-organized bei Prof. Becker and Prof. Teich

8 Trainings and Tutorials

Workshops and trainings were organized under the coordination of Project Z, to give InvasIC-members the opportunity to to strengthen their soft-skills, train their key qualifications and improve their knowledge on InvasIC-specific topics. In workshops co-organized with the local graduate schools, especially the PhD students trained their skills in presenting their scientific work in english talks and publications. In special Gender-Trainings all InvasIC-members were sensitised to applicate gender mainstreaming principles in daily work.

Event	Date	
Dagstuhl seminar No. 10281	July 11-16, 2010	Dagstuhl seminar on "Dynamically Reconfigurable Architectures" was co- organized by Prof. Jürgen Becker and Prof. Jürgen Teich
Seminar Variability-induced challenges beyond 22nm	July 26-27, 2010 Munich	Speaker: Dr. Sani Nassif (IBM)
X10 Tutorial	Nov. 12, 2010 Erlangen	Speaker: Prof. Christoph von Praun (Georg-Simon-Ohm-Hochschule, Nuremberg)
Gender Training I	Dec. 15, 2010 Munich	The trainers Marion Bredebusch and Martin Conrath sensitised the participants of the seminar on pitfalls in the communication between man and woman
Wittenberg Colloquium	May 22-24, 2011	Prof. Teich co-organises an extraordinary interdisciplinary doctoral researcher colloquium on trends in multi-core architecture research
Seminar Academic Writing	July 21/22 Erlangen	The seminar of Prof. Erika von Rautenfeld was was adressed especially to the junior scientists of InvasIC
Seminar Academic Talks	Aug. 15/16 2011 Erlangen	The seminar of Joseph Ganahl was was adressed especially to the junior scientists of InvasIC
Gender-Training I	Dec. 12, 2011 Erlangen	Repetition of the training from Dec. 2010 for the remaining InvasIC members

9 Industrial and Scientific Board

For the promotion of our ideas to the industrial community and for the discussion with peer colleagues world-wide, we decided to establish the InvasIC Industrial and Scientific Board. Suitable researchers and industrial experts are currently selected and invited to form the board. Until now, 6 experts from four institutions have become members of the board:

IBM

Dr. Peter Roth (IBM Böblingen)

Dr. Patricia Sagmeister (IBM Rüschlikon)

Intel

Hans-Christian Hoppe (Intel Director of ExaCluster Lab Jülich, Intel Director of Visual Computing Institute Saarbrücken)

Elmar Maas (Intel Braunschweig)

University of Edinburgh

Prof. Dr. Michael O'Boyle
(Director Institute for Computing Systems Architecture)

Georg-Simon-Ohm Hochschule Nürnberg

Prof. Dr. Christoph von Praun
(Faculty Member and Associate Department Chair)

10 InvasIC Activities

To promote the ideas and results of InvasIC and discuss them with leading experts from industry and academia, InvasIC-members invited guest speakers to the "InvasIC Seminar", gave talks at important research sites ("Invited Talks") and gave or organised workshops ("Workshops and Conferences) on the topics of invasive computing. The "InvasIC Seminar" is a series of talks given alternately at one of the three sites. A live-stream of the respective talk is transmitted via AdobeConnect to the other sites. For further information we refer to our website `http://www.invasic.de`.

Figure 10.1: Prof. Neil Bergmann giving a talk at the InvasIC Seminar

Figure 10.2: Prof. Sybille Hellebrand together with Prof. Jürgen Teich

InvasIC Seminar

Time and Place	Title	Speaker
Erlangen, July 15. 2010	DSystemJ — A GALS Language for Dynamic Distributed Systems	Dr. Avinash Malik (INRIA, Grenoble)
Erlangen, Nov. 3. 2010	Designs and design methods for embedded security.	Prof. Ingrid Verbawhede (K.U. Leuven)
Munich, Nov. 23. 2010	Optimising for a multi-core when you have to share.	Prof. Michael O'Boyle (University of Edinburgh)
Erlangen, Nov. 26. 2010	System-level MPSoC Design with Daedalus.	Hristo Nikolov, Ph.D. (Leiden Institute of Advanced Computer Science)
Erlangen, Jan. 31. 2011	Wie zuverlässig sind robuste Systeme wirklich?	Prof. Sybille Hellebrand (Universität Paderborn)
Erlangen, Feb. 18. 2011	Adaptive Verbindungsnetzwerke für Parallelrechner: Vom SAN (System Area Network) zum NoC (Network on Chip).	Prof. Erik Maehle (Universität zu Lübeck)
Erlangen, Mar. 3. 2011	Das Polyedermodell zur automatischen Schleifenparallelisierung	Prof. Christian Lengauer (Universität Passau)
München, Mar. 30. 2011	Insieme - an optimization system for OpenMP, MPI and OpenCL programs.	Dr. Hans Moritsch (Universität Innsbruck)
Erlangen, Mar. 30. 2011	Das Polyedermodell vor dem Durchbruch?	Dr. Armin Größlinger (Universität Passau)
Erlangen, July 6. 2011	Reconfigurable Computing Research at University of Queensland.	Prof. Neil Bergmann (University of Queensland)
Erlangen, July 11. 2011	Profile-Directed Semi-Automatic Parallelisation.	Prof. Bjoern Franke (University of Edinburgh)
Karlsruhe, July 19. 2011	What does it take to write efficient proximity algorithms (for robotics, graphics and CAD)	Prof. Kim Young (Ewha Womans University)
Erlangen, Aug. 26. 2011	Physics-inspired Management of Complex Adaptive Network Structures.	Dr. Ingo Scholtes (Universität Trier)
Erlangen, Sep. 14. 2011	Learning to see and understand.	PD Dr. Rolf Würtz (Ruhr-Universität Bochum)
Munich, Nov. 18. 2011	Parallelization of the Computation of a SPAI Preconditioner.	Dr. Gilles Fourestey (Swiss National Supercomputing Centre, Lugano)

Invited Talks

Date and Place	Title	Speaker
Aug. 6. 2010, National University of Singapore (NUS)	Invasive Computing - A Novel Paradigm for Parallel Computing	Prof. Jürgen Teich (FAU)
Aug. 8. 2010, The University of Sydney, Australia	Invasive Computing - An Overview	Prof. Jürgen Teich (FAU)
Oct. 15. 2010, IBM Böblingen, GI-Fachgruppe Betriebssysteme	Systemsoftware im Zeitalter mehrkerniger Prozessoren	Prof. Wolfgang Schröder-Preikschat
June 16. 2011, Hasso-Plattner-Institut, Potsdam, Future Trends in SOC 2011	System Software in the Many-Core Era	Prof. Wolfgang Schröder-Preikschat
July 22. 2011, Par Lab, UC Berkeley, California, USA	Invasive Parallel Computing - An Introduction	Prof. Jürgen Teich (FAU)
July 25. 2011, Stanford University (USA), Department of Electrical Engineering and Department of Computer Science	Invasive Parallel Computing - An Introduction	Prof. Jürgen Teich (FAU) invited by Prof. Subhasish Mitra
Sep. 9. 2011, Universität zu Lübeck, Colloquium Computer Science	Invasive Parallel Computing - An Introduction	Prof. Jürgen Teich (FAU) invited by Professor Erik Maehle
Sep. 15. 2011, University of Erlangen-Nuremberg, 12th Colloquium of the DFG Priority Programme 1183 "Organic Computing"	Invasive Parallel Computing - An Introduction	Prof. Jürgen Teich (FAU), Keynote

Figure 10.3: Prof. Teich at Par Lab, UC Berkeley

Workshops and Conferences

Date and Place	Title	Speaker
Sep. 9. 2010, Autrans, France (Artist Network of Excellence on Embedded System Design Summer School Europe 2010)	Tutorial Invasive Computing - Basic Concepts and Foreseen Benefits	Prof. Jürgen Teich (FAU)
Oct. 26. 2010, Scottsdale, AZ, USA (International Conference on Hardware-Software Codesign and System Synthesis (CODES+ISSS))	Retargetable Mapping of Loop Programs on Coarse-grained Reconfigurable Arrays	Dr. Frank Hannig (FAU)
Oct. 28. 2010, Scottsdale, AZ, USA (Workshop on Compiler-Assisted System-On-Chip Assembly (CASA))	Invasive Computing	Prof. Jürgen Teich (FAU)
Oct. 28. 2010, Scottsdale, AZ, USA (Workshop on Compiler-Assisted System-On-Chip Assembly (CASA))	Communication Synthesis of Loop Accelerator Pipelines	Dr. Frank Hannig (FAU)
Nov. 15. 2011, Erlangen, Germany (Tutorial at Multicore@Siemens 2011)	Frameworks for Multi-core Architectures and GPU Accelerators: A Comprehensive Evaluation using 2D/3D Image Registration	Richard Membarth (FAU) and Dr. Wieland Eckert (Siemens)

11 Publications

[AGSLW11] N. P. Aryan, G. Georgakos, D. Schmitt-Landsiedel, and M. Wirnshofer. “Comparison of In-situ Delay Monitors for Use in Adaptive Voltage Scaling”. In: *Kleinheubacher Tagung 2011*. to appear in Volume 10 (2012) in Advances in Radio Science (ARS) Journal. 2011.

[Bad08] M. Bader. *Exploiting the Locality Properties of Peano Curves for Parallel Matrix Multiplication*. Las Palmas, Aug. 2008.

[BBGH+11] M. Bader, H.-J. Bungartz, M. Gerndt, A. Hollmann, and J. Weidendorfer. “Invasive Programming as a Concept for HPC”. In: *Proceedings of the 10h IASTED International Conference on Parallel and Distributed Computing and Networks 2011 (PDCN)*. Feb. 2011.

[BFHK+12] J. Becker, S. Friederich, J. Heisswolf, R. Koenig, and D. May. “Hardware Prototyping of Novel Invasive Multicore Architectures”. In: *Proceedings of the 17th Asia and South Pacific Design Automation Conference (ASP-DAC)*. Sydney, Australia, Jan. 30–Feb. 2, 2012, pp. 201–206.

[BHTP11] S. Boppu, F. Hannig, J. Teich, and R. Perez-Andrade. “Towards Symbolic Run-Time Reconfiguration in Tightly-Coupled Processor Arrays”. In: *Proceedings of the International Conference on Reconfigurable Computing and FPGAs (ReConFig)*. Cancun, Mexico: IEEE Computer Society, Nov. 30–Dec. 2, 2011, pp. 392–397. ISBN: 978-1-4577-1734-5. DOI: 10.1109/ReConFig.2011.91.

[BH09] M. Braun and S. Hack. “Register Spilling and Live-Range Splitting for SSA-Form Programs”. In: *Proceedings of the International Conference on Compiler Construction (CC)*. Springer, Mar. 2009, pp. 174–189. DOI: 10.1007/978-3-642-00722-4_13.

[BZB11] S. Buchwald, A. Zwinkau, and T. Bersch. “SSA-Based Register Allocation with PBQP”. In: *Proceedings of the International Conference on Compiler Construction (CC)*. Ed. by J. Knoop. Vol. 6601. Lecture Notes In Computer Science (LNCS). Springer, 2011, pp. 42–61. DOI: 10.1007/978-3-642-19861-8_4.

[BGLM+11] H.-J. Bungartz, B. Gatzhammer, M. Lieb, M. Mehl, and T. Neckel. “Towards Multi-Phase Flow Simulations in the PDE Framework Peano”. In: *Computational Mechanics* 48.3 (2011), pp. 365–376. URL: `http://www5.in.tum.de/pub/int/compumech_mehletal_2011.pdf`.

[CLS11] N. Chen, B. Li, and U. Schlichtmann. “Timing Modeling of Flipflops Considering Aging Effects”. In: *International Workshop on Power and Timing Modeling, Optimization and Simulation (PATMOS)*. Vol. 6951. Lecture Notes in Computer Science (LNCS). Sept. 2011, pp. 63–72.

[CG11] I. A. Comprés Ureña and M. Gerndt. “Improved RCKMPI’s SCCMPB Channel: Scaling and Dynamic Processes Support”. 4th MARC Symposium. Dec. 2011.

[FHGB+11] P. Figuli, M. Hübner, R. Girardey, F. Bapp, T. Bruckschlögl, F. Thoma, J. Henkel, and J. Becker. “A heterogeneous SoC Architecture with embedded virtual FPGA Cores and runtime Core Fusion”. In: *NASA/ESA 6th Conference on Adaptive Hardware and Systems (AHS)*. San Diego, CA, USA, June 2011.

[GBH12] A. Grudnitsky, L. Bauer, and J. Henkel. “Partial Online-Synthesis for Mixed-Grained Reconfigurable Architectures”. In: *IEEE/ACM 15th Design Automation and Test in Europe Conference (DATE)*. to appear. Dresden, Germany, Mar. 2012.

[HGG06] S. Hack, D. Grund, and G. Goos. “Register Allocation for Programs in SSA-Form”. In: *Proceedings of the International Conference on Compiler Construction (CC)*. Ed. by A. Zeller and A. Mycroft. Vol. 3923. Lecture Notes In Computer Science (LNCS). Springer, Mar. 2006, pp. 247–262. DOI: `10.1007/11688839_20`.

[Han10] F. Hannig. “Retargetable Mapping of Loop Programs on Coarse-grained Reconfigurable Arrays”. Talk, International Conference on Hardware-Software Codesign and System Synthesis (CODES+ISSS), Scottsdale, AZ, USA. Oct. 26, 2010.

[HRST+11] F. Hannig, S. Roloff, G. Snelting, J. Teich, and A. Zwinkau. “Resource-Aware Programming and Simulation of MPSoC Architectures through Extension of X10”. In: *Proceedings of the 14th International Work-*

shop on Software and Compilers for Embedded Systems (SCOPES). St. Goar, Germany: ACM Press, June 27–28, 2011, pp. 48–55. ISBN: 978-1-4503-0763-5. DOI: 10.1145/1988932.1988941.

[HBHG11] J. Henkel, L. Bauer, M. Hübner, and A. Grudnitsky. “i-Core: A run-time adaptive processor for embedded multi-core systems”. In: *Proceedings of the International Conference on Engineering of Reconfigurable Systems and Algorithms (ERSA)*. invited paper. Las Vegas, NV, USA, July 2011.

[HHBW+12] J. Henkel, A. Herkersdorf, L. Bauer, T. Wild, M. Hübner, R. Pujari, A. Grudnitsky, J. Heisswolf, A. Zaib, B. Vogel, V. Lari, and S. Kobbe. “Invasive Manycore Architectures”. In: *Proceedings of the 17th Asia and South Pacific Design Automation Conference (ASP-DAC)*. Jan. 30–Feb. 2, 2012, pp. 193–200.

[HLSP11] W. Hofer, D. Lohmann, and W. Schröder-Preikschat. “Sleepy Sloth: Threads as Interrupts as Threads”. In: *Proceedings of the 32nd IEEE International Symposium on Real-Time Systems (RTSS)*. Vienna, Austria: IEEE Computer Society, Dec. 2011, pp. 67–77. ISBN: 978-0-7695-4591-2.

[HG11] A. Hollmann and M. Gerndt. “iOMP Language Specification 1.0”. Internal Report. Dec. 2011.

[HFGS+11] M. Hübner, P. Figuli, R. Girardey, D. Soudris, K. Siozios, and J. Becker. “A Heterogeneous Multicore System on Chip with Run-Time Reconfigurable Virtual FPGA Architecture”. In: *Proceedings of the International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*. Anchorage, AK, USA, May 16–17, 2011.

[HTGB+11] M. Hübner, C. Tradowsky, D. Göhringer, L. Braun, F. Thoma, J. Henkel, and J. Becker. “Dynamic Processor Reconfiguration”. In: *Proceedings of the International Conference on Reconfigurable Computing and FPGAs (ReConFig)*. Cancun, Mexico: IEEE Computer Society, Nov. 30–Dec. 2, 2011.

[KGSH+11] D. Kissler, D. Gran, Z. Salcic, F. Hannig, and J. Teich. “Scalable Many-Domain Power Gating in Coarse-grained Reconfigurable Processor Arrays”. In: *IEEE Embedded Systems Letters* 3.2 (June 2011), pp. 58–61. ISSN: 1943-0663. DOI: 10.1109/LES.2011.2124438.

[KJS12] C. Knoth, H. Jedda, and U. Schlichtmann. “Current Source Modeling for Power and Timing Analysis at Different Supply Voltages”. In: *Design Automation and Test in Europe (DATE)*. To appear. Mar. 2012.

[KUKS11] C. Knoth, C. Uphoff, S. Kiesel, and U. Schlichtmann. “SWAT: Simulator for Waveform-Accurate Timing including Parameter Variations and Transistor Aging”. In: *International Workshop on Power and Timing Modeling, Optimization and Simulation (PATMOS)*. Vol. 6951. Lecture Notes in Computer Science (LNCS). Sept. 2011, pp. 193–203.

[KBHL+11] S. Kobbe, L. Bauer, J. Henkel, D. Lohman, and W. Schröder-Preikschat. “DistRM: Distributed Resource Management for On-Chip Many-Core Systems”. In: *Proceedings of the IEEE International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*. Taipei, Taiwan, Oct. 9–14, 2011, pp. 119–128.

[KHLT11] G. Kouveli, F. Hannig, J. Lupp, and J. Teich. “Towards Resource-Aware Programming on Intel’s Single-Chip Cloud Computer Processor”. In: *3rd Many-core Applications Research Community (MARC) Symposium*. Vol. 7598. KIT Scientific Reports. Ettlingen, Germany: KIT Scientific Publishing, July 5–6, 2011, pp. 111–114. ISBN: 978-3-86644-717-2.

[LBMH+11] V. Lari, S. Boppu, S. Muddasani, F. Hannig, and J. Teich. “Hierarchical Power Management for Adaptive Tightly-Coupled Processor Arrays”. Talk, International Workshop on Adaptive Power Management with Machine Intelligence at International Conference on Computer-Aided Design (ICCAD), San Jose, CA, USA. Nov. 10, 2011.

[LHT11] V. Lari, F. Hannig, and J. Teich. "Distributed Resource Reservation in Massively Parallel Processor Arrays". In: *Proceedings of the International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*. Anchorage, AK, USA: IEEE Computer Society, May 16–17, 2011, pp. 313–316. ISBN: 978-0-7695-4385-7. DOI: 10.1109/IPDPS.2011.157.

[LNHT11] V. Lari, A. Narovlyanskyy, F. Hannig, and J. Teich. "Decentralized Dynamic Resource Management Support for Massively Parallel Processor Arrays". In: *Proceedings of the 22nd IEEE International Conference on Application-specific Systems, Architectures, and Processors (ASAP)*. Santa Monica, CA, USA: IEEE Computer Society, Sept. 2011, pp. 87–94. ISBN: 978-1-4577-1291-3. DOI: 10.1109/ASAP.2011.6043240.

[LBS11] D. Lorenz, M. Barke, and U. Schlichtmann. *Finding Possible Critical Paths for On-line Monitoring Of Aging in Integrated Circuits*. Technical Report. Technische Universität München, Dec. 2011.

[MTKB+11] P. Marwedel, J. Teich, G. Kouveli, I. Bacivarov, L. Thiele, S. Ha, C. Lee, Q. Xu, and L. Huang. "Mapping of Applications to MPSoCs". In: *Proceedings of the IEEE International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*. Taipei, Taiwan, Oct. 9–14, 2011, pp. 109–118.

[MLHT+12] R. Membarth, J. Lupp, F. Hannig, J. Teich, M. Körner, and W. Eckert. "Dynamic Task-Scheduling and Resource Management for GPU Accelerators in Medical Imaging". In: *Proceedings of the 24th International Conference on Architecture of Computing Systems (ARCS)*. Munich, Germany, Feb. 28–Mar. 2, 2012.

[OSKB+11] B. Oechslein, J. Schedel, J. Kleinöder, L. Bauer, J. Henkel, D. Lohmann, and W. Schröder-Preikschat. "OctoPOS: A Parallel Operating System for Invasive Computing". In: *Proceedings of the International Workshop on Systems for Future Multi-Core Architectures (SFMA)*. Ed. by R. McIlroy, J. Sventek, T. Harris, and T. Roscoe. Vol. USB Proceedings. Sixth International ACM/EuroSys European Conference on Computer Systems (EuroSys). EuroSys. Salzburg, Austria, Apr. 2011, pp. 9–14.

[PSKA+12] J. Paul, W. Stechele, M. Kröhnert, T. Asfour, and R. Dillmann. “Invasive Computing for Robotic Vision”. In: *Proceedings of the 17th Asia and South Pacific Design Automation Conference (ASP-DAC)*. Sydney, Australia, Jan. 30–Feb. 2, 2012, pp. 207–212.

[PWHV+11] R. K. Pujari, T. Wild, A. Herkersdorf, B. Vogel, and J. Henkel. “Hardware Assisted Thread Assignment for RISC based MPSoCs in Invasive Computing”. In: *Proceedings of the 13th International Symposium on Integrated Circuits (ISIC)*. Singapore, Dec. 2011.

[RHT12] S. Roloff, F. Hannig, and J. Teich. “Approximate Time Functional Simulation of Resource-Aware Programming Concepts for Heterogeneous MPSoCs”. In: *Proceedings of the 17th Asia and South Pacific Design Automation Conference (ASP-DAC)*. Sydney, Australia, Jan. 30–Feb. 2, 2012, pp. 187–192.

[SS11] P. Sanders and J. Speck. “Efficient Parallel Scheduling of Malleable Tasks”. In: *International Parallel and Distributed Processing Symposium (IPDPS)*. Anchorage, AL, USA: IEEE Computer Society, 2011, pp. 1156–1166. DOI: 10.1109/IPDPS.2011.110.

[SSP11] P. Stellwag and W. Schröder-Preikschat. “Challenges in Real-Time Synchronization”. In: *Proceedings of the 3rd USENIX Workshop on Hot Topics in Parallelism (HotPar)*. Ed. by M. McCool and M. Rosenblum. Berkeley, CA, USA: USENIX Association, May 2011.

[TLSSP11] R. Tartler, D. Lohmann, J. C. R. Sincero, and W. Schröder-Preikschat. “Feature Consistency in Compile-Time Configurable System Software”. In: *Proceedings of the Sixth International ACM/EuroSys European Conference on Computer Systems (EuroSys)*. Ed. by C. Kirsch and G. Heiser. Salzburg, Austria: ACM Press, Apr. 2011, pp. 47–60.

[Tei10] J. Teich. “Invasive Computing – Basic Concepts and Foreseen Benefits”. Artist Network of Excellence on Embedded System Design Summer School Europe 2010, Autrans, France, Invited Tutorial. Sept. 7, 2010.

[Tei11a] J. Teich. “Invasive Parallel Computing – An Introduction”. Par Lab and AMP Lab Seminar Talk, UC Berkeley, CA, USA. July 22, 2011.

[Tei11b] J. Teich. “Programming Invasively Parallel – An Introduction”. Pervasive Parallelism Laboratory (PPL) Seminar Talk, Stanford University, CA, USA. July 25, 2011.

[THHSL+11] J. Teich, J. Henkel, A. Herkersdorf, D. Schmitt-Landsiedel, W. Schröder-Preikschat, and G. Snelting. “Invasive Computing: An Overview”. In: *Multiprocessor System-on-Chip – Hardware Design and Tool Integration*. Ed. by M. Hübner and J. Becker. Springer, Berlin, Heidelberg, 2011, pp. 241–268.

[VRMD+10] T. Vander Aa, P. Raghavan, S. Mahlke, B. De Sutter, A. Shrivastava, and F. Hannig. “Compilation Techniques for CGRAs: Exploring All Parallelization Approaches”. In: *Proceedings of the International Conference on Hardware-Software Codesign and System Synthesis (CODES+ISSS)*. Scottsdale, AZ, USA: ACM, Oct. 24–29, 2010, pp. 185–186. ISBN: 978-1-60558-905-3. DOI: 10.1145/1878961.1878995.

[WWT11] A. Weichslgartner, S. Wildermann, and J. Teich. “Dynamic Decentralized Mapping of Tree-Structured Applications on NoC Architectures”. In: *Proceedings of the Fifth ACM/IEEE International Symposium on Networks-on-Chip (NOCS)*. Pittsburgh, PA, USA, May 1–4, 2011, pp. 201–208.

[Wei09] T. Weinzierl. “A Framework for Parallel PDE Solvers on Multiscale Adaptive Cartesian Grids”. Dissertation. München: Institut für Informatik, Technische Universität München, 2009. URL: http://www.dr.hut-verlag.de/978-3-86853-146-6.html.

[WHGSL11a] M. Wirnshofer, L. Heiss, G. Georgakos, and D. Schmitt-Landsiedel. “A Variation-Aware Adaptive Voltage Scaling Technique Based on In-Situ Delay Monitoring”. In: *IEEE 14th International Symposium on Design and Diagnostics of Electronic Circuits & Systems*. 2011, pp. 261–266.

[WHGSL11b] M. Wirnshofer, L. Heiss, G. Georgakos, and D. Schmitt-Landsiedel. “An Energy-Efficient Supply Voltage Scheme using In-Situ Pre-Error Detection for on-the-fly Adaptation to PVT Variations”. In: *International Symposium on Integrated Circuits*. 2011.