Sonderforschungsbereich/Transregio 89
Transregional Collaborative Research Center 89

Invasive Computing
Annual Report 2012

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 2012
January 2012 – December 2012

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 2012.

The main idea of InvasIC is to develop and investigate a completely novel paradigm for designing and programming future parallel computing systems. To support its major 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.

A highlight in 2012 was the Annual Meeting in Kloster Irsee, Germany. 55 scientists met to review and discuss the progress within the last 12 months. Besides short talks in which projects and the three working groups presented the state of the art of their work, the meeting program also comprised a poster session. A special highlight was the demonstrator day on the second day of the Annual Meeting. Here, 6 members of the "InvasIC Industrial and Scientific Board" attended to evaluate the ideas and progress of the presented projects in real and first time hardware and software demonstrations.

Also, the SFB/TRR InvasIC was very active in organising workshops, seminars and special sessions on the topic of invasive computing throughout the year.

We would like to thank all members of the SFB/TRR InvasIC and all our partners from industry and academia for the fruitful collaborations and inspiring discussions. The substantial progress made within the last year was only possible by the motivated and committed work of the members of InvasIC.

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

The goal of the Transregional Collaborative Research Centre InvasIC is to implement invasive programming through all layers of abstraction, starting from language design, supportive OS and middleware layers to new architectural processor, interconnect and memory structures providing the necessary extensions and isolations to support predictable and resource-aware programming on multicores. 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 Centre aggregating the best researchers from three excellent sites in Germany do 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
- Prof. Dr.-Ing. Tamim Asfour

Institut für Technik der Informationsverarbeitung

- Prof. Dr.-Ing. Jürgen Becker

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 main aspects of invasive computing, the TRR 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, Prof. 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, Prof. 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 Zwinkau, Andreas Weichslgartner, Stefan Wildermann

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 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.

Overhead Analysis

When considering benefits of invasive programming, 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 experimental results for several available many-core ar-

chitectures such as Intel's 48-core Single-Chip Cloud Computer (SCC), the Tilera's 64-core TILEPro64, as well as TCPAs (cooperation with Project B2) and in single RISC tiles (cooperation with Project C1), with a shared memory and a minimal implementation of the invasive core primitives. The result of these measurements, the derived formulas for the RISC tile and an introduction to invasive computing was published in [TWOS12]. It could be shown that for the minimal RISC tile implementation, the invasive overheads are reasonably small and behave almost linearly to the number of cores. For the SCC and TILEPro64, the overheads are orders of magnitudes higher because of the crossing of operating system borders and the costly thread/process creation. To consider complex communication structures, Project A1 also worked together with Project B5 and Project C2 on an integration of a network model into the functional simulator [RHT12a].

Game-Theoretic Analysis of Distributed Core Allocation

The invasive constructs invade and retreat are used locally in applications to steer their resource utilisation. However, this local usage influences the system globally. Thus, it is important to anticipate how the local use of invade and retreat influences the overall system. In [WZT13], we proposed and applied game theory for analysing the influence of different local usage schemes on the global goal of maximising the average speedup of multiple concurrently active applications. Particularly, we studied the convergence time, communication overhead, and the optimality of several core allocations schemes.

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 many 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. After the iterative group process for language design last year, we had a working base system [Zwi12] for

invasive programming called InvadeX10, which is based on the X10 programming language from IBM. For further changes we adopted a document-based approach, inspired by RFCs (Request for Comments), PEPs (Python Enhancement Proposal), JSRs (Java Specification Request), and others. This allowed for small interest groups to develop new features independently and in parallel. Only for a final consensus the whole project was involved.

From a range of currently 22 language change proposals (LCPs), so far seven have reached consensus and were implemented into our invasive programming framework. Namely:

Asynchronous Block Prefetching For efficient matrix multiplication algorithms, blocks of data must be cached on tile-local memory. Those blocks should be prefetched in parallel to the multiplication. This can be done efficiently by the *i*NoC, which acts similar to a DMA controller on a desktop computer.

iNoC Constraints The *i*NoC can reserve throughput and latency for specific applications. These resources can be allocated via constraints. The application as a whole benefits, without additional effort for the programmer. To give Quality-of-Service (QoS) guarantees is an essential part of Project B5 research and will be also demonstrated.

Scalability Graphs / Performance Curves An invasive application requests its resource requirements via a constraint system. For example, if an application asks for 8–12 cores, it either gets any number of cores in this range or an exception is thrown. To evaluate the benefit of resources within that range, an application must express to which extent it benefits from the requested resources. Performance/scalability curves provide this information to the agent system.

Claim.update An agent system (Project C1) continuously prepares a resource optimisation (shrink/extend/modify) using hints from the application. We propose a reinvade() method, so applications can allow the optimisation prepared by its agent. The claim still fulfils the constraints.

PE Retreat Retreat single specific processing elements.

Change Claim Constraints The agent system prepares and optimises resource allocation using the given constraints. A claim's constraint can be updated to adapt to resource changes. To support this agent system API, the framework needs to be extended.

ThisPlace Constraint Currently, it is impossible to request processors (PEs) that must be part of the same tile as the requesting PE. As all PEs within a place have fast access to shared memory, this constraint is useful to avoid expensive inter-place communication.

Manual

For a more formal specification and better documentation, a manual was written. It provides an introduction to invasive computing for a programmer, who wants to use the InvadeX10 framework. It also lists all constraints and features of the framework with their semantics.

Publications

[RHT12a] 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. ISBN: 978-1-4673-0770-3. DOI: 10.1109/ASPDAC.2012.6164943.

[Tei12] J. Teich. “Hardware/Software Co-Design: The Past, Present, and Predicting the Future”. In: *Proceedings of the IEEE* 100.Centennial-Issue (May 2012), pp. 1411–1430. DOI: 10.1109/JPROC.2011.2182009.

[TWOS12] J. Teich, A. Weichslgartner, B. Oechslein, and W. Schröder-Preikschat. “Invasive Computing - Concepts and Overheads”. In: *Forum on Specification & Design Languages (FDL)*. Vienna, Austria, Sept. 18–20, 2012, pp. 193–200. ISBN: 978-2-9530504-5-5.

[WZT13] S. Wildermann, T. Ziermann, and J. Teich. “Game-Theoretic Analysis of Decentralized Core Allocation Schemes on Many-core Systems”. In: *Proceedings of Design, Automation and Test in Europe Conference (DATE)*. Mar. 18–22, 2013.

[Zwi12] A. Zwinkau. *Resource Awareness for Efficiency in High-Level Programming Languages*. Tech. rep. 12. Karlsruhe Institute of Technology, 2012. URL: http://pp.info.uni-karlsruhe.de/uploads/publikationen/zwinkau12high.pdf.

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 invasive computing systems.

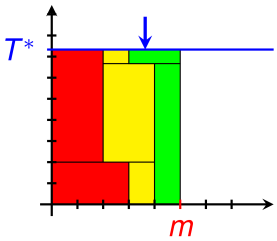


Figure 4.1: Energy Efficient Malleable Schedule

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. As announced last year we included energy efficiency into our solution of the scheduling problem of malleable tasks (see [SS12]). In the energy efficient setting the tasks are malleable and additionally the speed of execution of the

cores can be changed. All tasks have to be processed in the same time interval. The goal was 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 super-linearly 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 plan to solve this problem also for the case when each job has a different interval during which it has to be processed. As we are considering energy efficiency as important for computer architectures in the future, we work further on energy efficient scheduling.

Together with Felix Brandt and Markus Völker, Jochen Speck took part in the Google ROADEF-Challenge (http://challenge.roadef.org/2012/en/index.php). The participation was quite successful as the second price in the junior category (no people with a PhD allowed) was won. The subject of the challenge was to improve solutions for big complex scheduling problems arising in the computing centres of Google. The challenge needed a different view on scheduling problems, because here the model was very complex, an optimal solution was impossible to find and it was impossible to theoretically cover the subject. On the other hand no performance guarantees were needed and with 5 minutes the time to compute a solution was quite long. Hence heuristics were used which improved the solution step by step until the given time was over. With this challenge and our recent theoretical results we cover the whole bandwidth from thorough analytic analysis of simple models to heuristic solutions for complex models which is needed for scheduling of invasive computing systems.

As proposed last year we finished the development of the malleable sorter which is a flexible sorting algorithm that can handle changes in the number of assigned processors very well. As we can see from the Figure 4.2 the improvements when a task called Loadtask runs in parallel are significant compared to the merge sort from MCSTL (here called STLMS). Also the performance of Loadtask shows an improvement. We still work on the X10 implementation. During that work we cooperated with Project C1 and Project C3 on the OS to application interface, which is very important for a malleable application.

A further important area is the scheduling of task-DAGs (DAG = directed acyclic graph). Many applications can be modelled as small tasks with dependencies. Especially numerical applications from Proj-

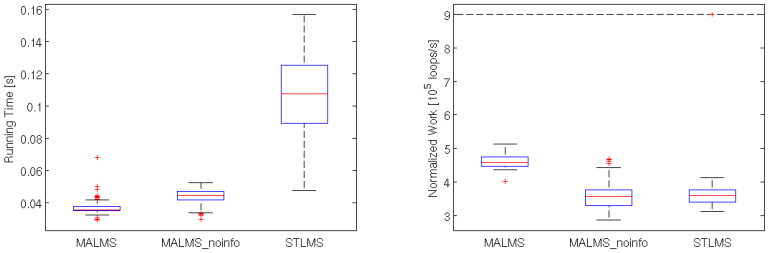


Figure 4.2: Results with parallel running Loadtask for sorting $n = 10^6$ integers, 2ms time slots and $k = 100$ workpackages. The second picture shows the performance of Loadtask.

ect D3 fit into this model. We started our investigations about DAG scheduling in the area of dense linear algebra algorithms. During that work we noticed that there is no scalable and work efficient matrix inversion algorithm. The Newton inversion method is not work efficient as the amount of work done is a logarithmic factor larger than in other algorithms. On the other hand the Newton method is very scalable. The inversion method of Strassen has a critical path linear in the size of the matrix, thus larger machines can not speed up this method very much if the matrix is not very big. On the other hand the method of Strassen is work efficient. We managed to combine these two algorithms and now have work efficient algorithm which also scales very well.

We also cooperate with the agent system (see Project C1) in order to build a theoretical model of the agent behaviour.

In the future we plan to do more work in the area of dense linear algebra. We plan to develop a malleable matrix multiplication and we will also continue our work on DAG scheduling. As the project builds more and more parts of the invasive system our models will get more detailed and we might be able to create scheduling algorithms for these more detailed models.

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.

[SS12] P. Sanders and J. Speck. “Energy Efficient Frequency Scaling and Scheduling for Malleable Tasks”. In: *Euro-Par 2012 Parallel Processing*. Vol. 7484. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2012, pp. 167–178. ISBN: 978-3-642-32819-0. DOI: 10.1007/978-3-642-32820-6_18. URL: http://dx.doi.org/10.1007/978-3-642-32820-6_18.

B1: Adaptive Application-Specific Invasive Microarchitecture

Jörg Henkel, Jürgen Becker, Lars Bauer, Michael Hübner

Artjom Grudnitsky, Carsten Tradowsky

The goal of Project B1 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 4.3, 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.

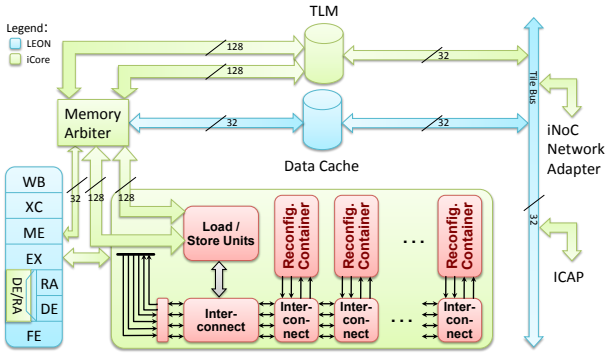


Figure 4.3: *i*-Core Architecture

Adaptive μArch

In [HGTH12], we introduce the novel methodology to adapt the microarchitecture of a processor during run-time. The goal is to tailor

the internal architecture to the requirements of an application and the data to be processed. In Project B1, we use the architecture description language (ADL) Language for Instruction-Set Architectures (LISA) for modelling the invasive microarchitecture.

LISA enables the developer to create behavioural as well as cycle-accurate models of processors and their instruction- set architecture (ISA). These models cover the needs of the design space exploration of hardware / software co-design and co-simulation. Thus, it is possible to close the gap between hardware description languages (HDL) on the one hand and the needs of compilers and ISA simulators on the other hand. The hardware / software co-design takes place in different levels of abstraction. We published our approach of modelling the *i*-Core in [TTHB12a]. The Best Work-in-Progress Paper Award was won with this publication.

LISA provides a description of the pipeline on the operational level which enables the modelling of complex pipeline mechanisms. To further evaluate the LISA approach the LImbiC (LISA model based ideal Cortex) is modelled. Out of this model all the tools, like the necessary tools as well as HDL code is generated. These generated tools, like the compiler and simulator, are only used for development and evaluation. In the end, the results are discussed with Project C2 and Project C3 to make use of their developments. The LImbiC is then extended by instructions to support image processing. With this extension of the hardware, the execution time of a Canny-Filter, in comparison to software execution, is reduced by 98 %. In addition to this, the instruction set of LImbiC can be reduced to the instructions, which are needed for the filter's execution. This adaptions for an optimised application-specific instruction-set processor (ASIP) for image processing.

The variation of *i*-Core's pipeline depth should increase performance and decrease power consumption depending on the specific work-load. The merge of adjacent pipeline stages is achieved by switching adjacent pipeline registers transparently. This requires additional instructions to be added to the existing instruction set. This instruction has to be inserted additionally into the existing program code by the compiler or the application developer for example using inline assembly. These additions to the instruction set have been made available to the invasive applications, operating system and compiler projects. As presented in [TTHB12b], it is important to highlight that the concept of pipeline reconfiguration has—in contrast to a branch prediction—impact on control flow branches such as jumps and calls. Thus, the potential of this concept on the one hand is to save one cycle per control flow branch.

On the other hand, valuable energy on the basis of the transparency of the pipeline registers can be saved and thus lower dynamic power consumption is anticipated. Furthermore, we implemented the adaptive pipeline onto a standalone Leon3 system, which was prototyped on a Xilinx XUPV5. We made it available to the application Project D1. Each adaptive microarchitecture feature will be brought into the invasive hardware on its own. After successful tests on the XUP system, *i*-Core will be integrated including more adaptive features onto the CHIPit system for the final demonstration.

While integrating the adaptive pipeline into Leon3, the question on where to place which part of the hardware on the FPGA came up. For further analysis of the heat distribution of the FPGA, we developed an on-chip measurement system for reconfigurable hardware, which can be placed close to the hardware itself throughout the reconfigurable hardware, in order to monitor its heat distribution. Based on the results from the experiments, a trend is identified. As a result, the effect of temperature variations are measurable. This enables the clarification of a temperature gradient across the reconfigurable hardware. These interesting results were published at the ReConFig conference [TCDH+12].

Another very interesting part of the microarchitecture is the cache. The concept of an adaptive cache for realisation on an FPGA platform is developed. This concept enables the adaptation of the cache parameters during run-time without the need for reconfiguration of the FPGA. Different properties of caches, like for example the line size, the associativity, the cache size or the write policy have an direct impact on the performance and on the efficiency of the cache itself. Until today, these parameters were fixed. This means that the configuration of the cache needs to be decided at design time according to a chosen trade-off. This trade-off might be not beneficial in some application scenarios. Together with Project D3, we identified the need for adaptive cache size, especially when running HPC scenarios, like the tsunami scenario. Depending on the application, it might be better for cache performance, e. g., minimising the miss rate and by that bus accesses, to change some parameter setting of the cache. Thus, the possibilities of an adaptive run-time adaptive cache are evaluated in order to have a run-time parametric cache. An adaptive concept based on a cache control register is currently developed to make the cache parameter accessible and change them during run-time via a register update. The resulting implementation uses more logic compared to a static design. In real world, however, the Leon3 is running at a low clock frequency, such that an overhead in area and speed can be used to implement more

adaptive features in the following year.

We will now discuss the reconfigurable fabric and Instruction Set Extension (ISE).

Reconfigurable Fabric

B1

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. A talk providing an overview of the reconfigurable fabric of the *i*-Core was given in [Hen12].

Performance-Aware Task Scheduling

Loading of accelerators, required for efficient execution of SIs, takes a considerable amount of time. While loading is an asynchronous operation, and thus concurrent to program execution, SIs have to be emulated in software until the required accelerators have finished loading. This *reconfiguration latency* can be efficiently hidden through a task scheduling approach that is aware of the underlying reconfigurable architecture. We have explored this approach in [BGSH12]. Tasks request accelerators, and their requests are subsequently fulfilled by loading accelerators onto the fabric. We have introduced *performance levels* for tasks, which are defined as $\frac{\#\text{ fulfilled accelerator loading requests}}{\#\text{ accelerator loading requests}}$ for a task. Our proposed scheduling approach, *PATS*, targets a set of periodic soft deadline tasks and aims to reduce the system tardiness, i. e., the sum of all times that the tasks finished late. The general approach is to prefer tasks with a high performance level, weighted by the remaining time to their deadline. Tasks with a low performance level wait while their performance level increases. A comparison of our PATS scheduler to existing scheduling approaches is shown in Figure 4.4.

Prototype Implementation

We have implemented the *i*-Core with the reconfigurable fabric on a XUP V5 board, which uses the same type of FPGA as the CHIPit demonstrator, which should reduce the migration effort to the CHIPit platform. In particular, we have we have developed a dedicated interconnect between the reconfigurable fabric and the tile-local memory, an IP core for the reconfiguration port, which allows loading of accelerators in an asynchronous manner and the bus-attached memory for the μ-programs that control Special Instruction execution on the fabric. A floorplan

of this implementation is shown in Figure 4.5. In addition to this standalone implementation, we have implemented an *i*-Core along with a standard Leon in a shared memory configuration, which can be viewed as a minimal tile configuration, in preparation to the integration of the *i*-Core into the common demonstration platform.

A demonstrator of the *i*-Core capabilities using a video encoder demo, was presented to the InvasIC industrial board (Figure 4.6) and at the demo night at ReConFig conference 2012.

Collaborations

We have developed two versions of the register file permutator as part of our cooperation with Project C3 - one optimised for speed, and one for low area overhead - and integrated them as optional components into the *i*-Core. Further details on the compiler support of this feature is discussed in the report of Project C3.

We have developed a behavioural and timing accurate *i*-Core simulator and provided it to Project C2 for integration into their overall InvasIC simulator. The simulator can also be used to model standard Leon cores.

In collaboration with the other architecture projects and Project Z2 a partitioning of the CHIPit demonstrator has been defined, with *i*-Core tiles placed onto their own FPGAs due to their higher memory requirements.

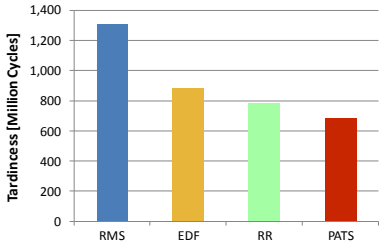


Figure 4.4: System tardiness of the PATS Scheduler compared to existing schedulers averaged over different task set configurations

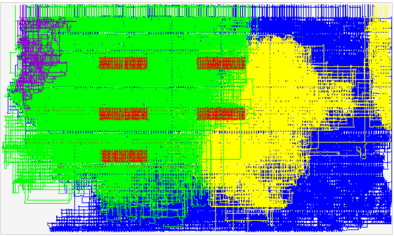


Figure 4.5: Floorplan of the *i*-Core prototype

Figure 4.6: *i*-Core prototype running a video encoding application, presented to the industrial board

Publications

[BGSH12] L. Bauer, A. Grudnitsky, M. Shafique, and J. Henkel. “PATS: a Performance Aware Task Scheduler for Runtime Reconfigurable Processors”. In: *20th Annual International IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM)*. Toronto, Canada, May 2012.

[Hen12] J. Henkel. “i-Core: Adaptive Computing for Multi-core Architectures”. Embedded System Design from MultiMedia to Cloud, Hong Kong, Invited Talk. May 18, 2012.

[HGTH12] M. Hübner, D. Göhringer, C. Tradowsky, and J. Henkel. “Adaptive Processor Architecture”. In: *12th International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS XII)*. July 2012.

[TCDH+12] C. Tradowsky, E. Cordero, T. Deuser, M. Hübner, and J. Becker Jürgen. “Determination of On-Chip Temperature Gradients on Reconfigurable Hardware”. In: *Proceedings of the International Conference on Reconfigurable Computing and FPGAs (ReConFig)*. Cancun, Mexico: IEEE Computer Society, Dec. 5–7, 2012.

[TTHB12a] C. Tradowsky, F. Thoma, M. Hübner, and J. Becker. “LISPARC: Using an architecture description language approach for modelling an adaptive processor microarchitecture (Best Work-in-Progress (WiP) Paper Award)”. In: *7th IEEE International Symposium on Industrial Embedded Systems (SIES’12)*. June 2012.

[TTHB12b] C. Tradowsky, F. Thoma, M. Hübner, and J. Becker. “On Dynamic Run-Time Processor Pipeline Reconfiguration”. In: *Proceedings of the International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*. May 2012.

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 for *tightly-coupled processor arrays* (*TCPAs*, as shown in Figure 4.7) are investigated including the development of proper hardware controller circuitry.

Architectural Overview

As part of the overall heterogeneous tiled invasive computing architecture [HHBW+12], TCPAs may be used as accelerators for computationally intensive tasks. That is, such processor arrays are designed to support many application-specific or domain-specific algorithms in image and signal processing domains. A simplified drawing of a TCPA with 24 processor elements (PEs) is sketched in Figure 4.7. The different rectangular areas show three applications running simultaneously on the array tile.

In the first two years of research, we have proved that an invasion of an array of TCPA processing elements can be accomplished at an overhead of only two clock cycles per processing element by exploiting regularity and by developing dedicated invasion control hardware structures, called *invasion controllers* or short *i*Ctrl (see Figure 4.7). Thanks to this hardware support, the overhead for resource management could be significantly reduced by two orders of magnitude compared to a centralised software-based approach. We proposed and evaluated two variants of invasion controllers, namely *FSM-based* and *programmable invasion controllers*. For these, we developed and evaluated various 1-D and 2-D invasion strategies with respect to invasion time, area, power and flexibility. Invasion signals (e. g., INV, CLAIM and RET) are propagated in the same way as data from one processing element (PE) to another neighbouring PE in a TCPA in order to (a) invade the PEs, (b) build a claim structure that includes information about the invaded

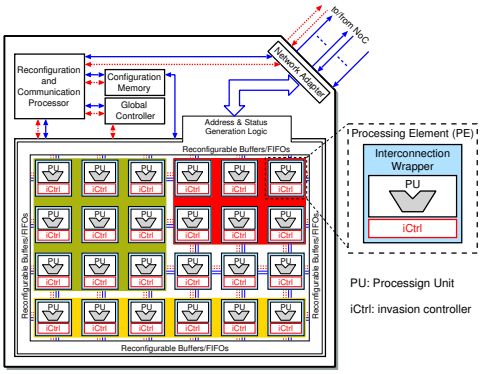


Figure 4.7: Invasive Tightly-Coupled Processor Array (TCPA). Each processing unit (PU) is augmented by an invasion controller (*i*Ctrl). Invade and retreat requests are propagated locally PE by PE through the mesh-connected array architecture.

PEs, and finally, (c) free the captured PEs in a distributed manner again.

Invasive Computing as an Enabler for Power Management

Resource-aware computing shows its importance when targeting many-core architectures consisting of tens to thousands of processing elements. Such a great number of computational resources allows to support very high levels of parallelism but on the other side may also cause a high power consumption. One traditional way to decrease the overall power dissipation on a chip is to decrease the amount of static power by powering-off unused resources. In the context of invasive computing, we now may exploit invasion requests to wake up processors and retreat requests to shut down the processors as well to save power. These invasion and retreat requests may be initiated by each application, thus, the architecture itself adopts to the application requirements in terms of power needs. During the invasion phase, two different kinds of power domains are considered: *processing unit power domains* and *invasion controller power domains*. These domains are controlled hierarchically, based on the system utilisation, which is in turn controlled by the invasion controllers (see Figure 4.8). When a PE receives an INV signal, its invasion controller is first powered on, and then when the invasion is confirmed by a CLAIM signal, the processing unit is turned on (making the PE ready to start application execution). Finally, by receiving a RET signal, both components are turned off again.

Power gating of individual invasion controllers may reduce the power consumption of the MPSoC but at the cost of timing overhead of power switching delays. Therefore, in [LMBH+12a] the effects of grouping multiple invasion controllers in the same power domain were studied. Such grouping mechanisms may reduce the hardware cost for power gating yet, by sacrificing the granularity of power gating capabilities. The finer the granularity for the power control, the more power we might save. In contrast, grouping more invasion controllers together will reduce the timing overhead that is needed for power switching during both, the invasion and the retreat phases. Figure 4.8 shows different example architectures for grouping the invasion controllers. Experimental results show that up to 70 % of system energy consumption may be saved for selected applications and different resource utilisations. In addition, we developed a mathematical energy consumption model being dependent on the size of the invasion controller power domains for TCPAs [LMBH+12b]. Notably, the estimation error of the presented models is less than 3.6 % when compared to the simulation results.

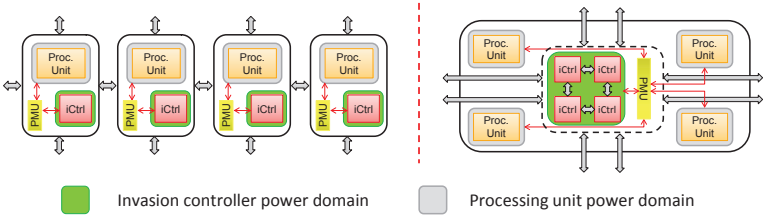


Figure 4.8: Different designs for grouping invasion controllers into one power domain. Left side: invasion controller power domains controlling the power state of a single invasion controller, right side: an invasion controller power domain controlling the power state of four invasion controllers belonging to four processing elements.

Invasive TCPA Video Demo

Combining the parallelism at instruction level (VLIW architecture) and at loop level (multiple PEs working concurrently) is a major strength of TCPAs, which, as aforementioned, makes them suitable as an accelerator for a multitude of computationally intensive applications like image or video processing. Here, we applied the invasive computing paradigm to video applications that are pre-processed on TCPAs. A visual eye-catcher for the benefits of invasive computing is here that the applications get the ability to adapt the quality of their output image depending on the number of available resources. Together with Project Z2, we have

implemented a first invasive TCPA prototype on FPGA basis, which was demonstrated at the Invasive Computing annual meeting in October 2012 as well as at the demo night of DASIP 2012 [MBHK+12]. For the demo, a TCPA array of size 5×5, a datapath width of 16 bits was generated, where each VLIW-PE was configured with one addition/-subtraction unit, one multiplier, one unit for logical operation and one shifter. The exact description of the prototype architecture and its implementation on the CHIPit FPGA platform is given in the report section of Project Z2.

Video Applications: The first targeted applications on the invasive TCPA prototype were several real-time[1] 1-D and 2-D image filters (e. g., FIR filtering, 2-D convolutions, edge and feature detection) on a streaming input video. Such image filters have many applications in imaging and video processing. The range of applications vary from very precise medical imaging systems to low precision industrial imaging and consumer video applications.

An example of a 2-D convolution is specified in Eq. (4.1). Here, the convolved output pixel at location (m, n) for a given window size of $w_h \times w_v$ is computed as follows:

$$y(m,n) = \sum_{i=\lfloor -w_h/2 \rfloor}^{\lfloor w_h/2 \rfloor} \sum_{j=\lfloor -w_v/2 \rfloor}^{\lfloor w_v/2 \rfloor} h(i,j) \cdot x(m-i, n-j) \qquad (4.1)$$

In Eq. (4.1), x represents the input pixel stream and h represents the convolution window. Depending on the number of invaded PEs, the convolution window size can be varied, which results in a different quality of the output images as shown in Figure 4.9.

Outlook

Our current work and future activities include the mapping of computer vision algorithms like *optical flow* and *Harris corner edge detection* such as studied in Project D1. In order to ease the algorithm mapping, we are also in the course of developing a compiler back end for the TCPA code generation in cooperation with Project C3, and investigate and design finally required architectural components like a global controller, which can reduce control overheads in the program execution on the TCPA. Currently, only streaming applications are supported. In order to allow for partitioned algorithms, a further task is the development

[1] Processing of a video stream with a resolution of 1024 $\times$ 768 pixels and 60 frames per second.

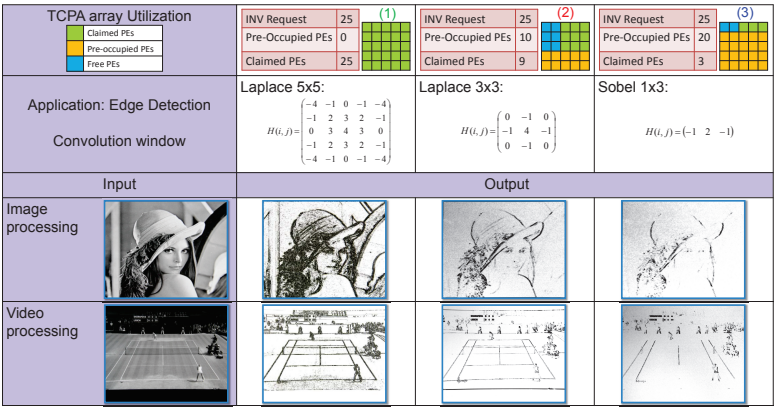


Figure 4.9: Three application scenarios for an invasive edge-detection algorithm on TCPAs, a 1×3 Soebel filter suitable to be mapped on 3 PEs, a 3×3 Laplace filter suitable to be mapped on 9 PEs and 5×5 Laplace filter on 25 PEs.

of address generators to support data access from/to buffers with in order or 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). Moreover, our 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

[Han12] F. Hannig. “Invasive Tightly-Coupled Processor Arrays”. Talk, 1st International Workshop on Domain-Specific Multicore Computing (DSMC) at International Conference on Computer-Aided Design (ICCAD), San Jose, CA, USA. Nov. 8, 2012.

[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. ISBN: 978-1-4673-0770-3. DOI: 10.1109/ASPDAC.2012.6164944.

[LMBH+12a] V. Lari, S. Muddasani, S. Boppu, F. Hannig, and J. Teich. “Design of Low Power On-Chip Processor Arrays”. In: *Proceedings of the 23rd IEEE International Conference on Application-specific Systems, Architectures, and Processors (ASAP)*. Delft,

The Netherlands: IEEE Computer Society, July 9–11, 2012, pp. 165–168. ISBN: 978-0-7695-4768-8. DOI: 10.1109/ASAP.2012.10.

[LMBH+12b] V. Lari, S. Muddasani, S. Boppu, F. Hannig, M. Schmid, and J. Teich. “Hierarchical Power Management for Adaptive Tightly-Coupled Processor Arrays”. In: *ACM Transactions on Design Automation of Electronic Systems (TODAES), accepted for publication* 18.1 (Dec. 2012).

[MBHK+12] S. Muddasani, S. Boppu, F. Hannig, B. Kuzmin, V. Lari, and J. Teich. “A Prototype of an Invasive Tightly-Coupled Processor Array”. In: *Proceedings of the Conference on Design and Architectures for Signal and Image Processing (DASIP)*. Karlsruhe, Germany: IEEE, Oct. 23–25, 2012, pp. 393–394. ISBN: 978-1-4673-2089-4.

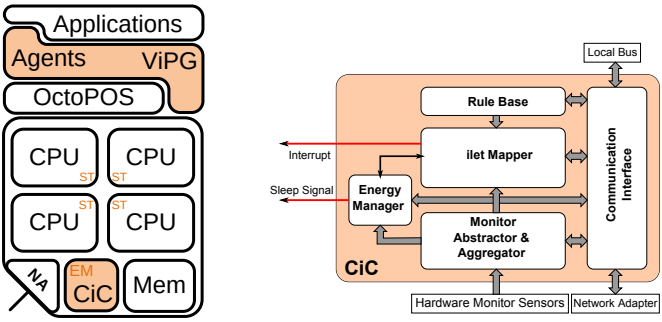
B3: Invasive Loosely-Coupled MPSoCs

Andreas Herkersdorf, Jörg Henkel

Anton Ivanov, Ravi Kumar Pujari, Muhammad Shafique, Benjamin Vogel, Thomas Wild

B3

Performance-optimised and energy-efficient invasive computing on loosely-coupled RISC processors is the major goal of this project. Functionalities with computational overhead within the *i*-let assignment process are 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-specific knowledge provided by the invasive programming 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 4.10: Envisaged architecture

Figure 4.10a 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 and hints. The C*i*Cs aggregate monitor data (e. g., performance counters or temperature values) provided by Project B4.

C*i*C

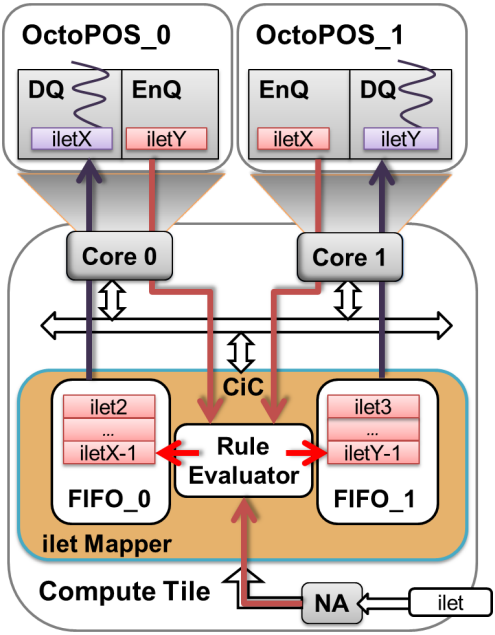


Figure 4.11: Infection of *i*-let among cores within a compute tile. New *i*-lets are buffered on arrival in *i*-let Mapper's FIFOs.

In order to let the invasive 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.

During this year, many enhancements to the initial C*i*C design are incorporated based on the review and discussions with the OctoPOS/*i*RTSS development team (Project C1). In particular, to avoid frequent interruptions to the run-time system software (OctoPOS) and to offload the

infection routines, *i*-let FIFOs within the *i*-let-mapper (Figure 4.11) are developed as part of the C*i*C_v1. With this C*i*C_v1, infection of *i*-lets among cores within a tile is possible. The *i*-let-mapper controller logic is designed to avoid race conditions that might occur due to non-atomic access over the bus while enqueuing or dequeuing *i*-lets. Also, to save power on empty FIFOs, the C*i*C drivers are enhanced to power down the cores in conjunction with the ViPG policies. The complete synchronisation and interaction mechanism with the SW layers is co-developed with the OctoPOS/*i*RTSS development team.

To enable resource awareness in the system, a number of hardware sensors/monitors are being developed. These sensors/monitor data aid the C*i*C's decision logic in selecting the cores for *i*-let assignments during infection stage. At any given point in time, these monitors can have arbitrary values based on the current/past run-time behaviour of applications. As a consequence, (a) it is really difficult for an application programmer to express the exact values for these dynamically changing sensor/monitor data that are suitable for an application to run and, also (b) for the Agent System, the search space would be prohibitively large to find a core matching all the hardware monitoring conditions as expected by an application.

To aid the applications to express their desired hardware monitoring conditions, we proposed to classify applications into a smaller / finite set of different classes (Table 4.1). Any application can express its hardware needs by specifying the class it belongs to, as part of the hints/constraints passed to the invade call.

Sensors/ Monitors	Application Class					
	Compute Intensive	Bandwidth Limited	Low Latency	Low Power	Safety Critical	General Purpose
Temperature	1	1	3	5	5	2
Aging	2	1	2	3	5	1
CPU load	5	1	4	4	3	2
CPU power	2	1	2	5	4	4
Cache miss rate	3	5	4	2	1	1
C*i*C fifo level	5	1	5	2	1	1
i-let dispatch rate	5	1	5	1	1	1
Bus load	-	-	-	-	-	-
NoC link load	-	-	-	-	-	-

Table 4.1: Weights for different sensor values for different application classes. A higher weight implies a higher relevance of the monitor data for an application.

For example,

- A programmer can specify which parts of the application are **Compute Intensive** and needs more computing power. The C*i*C's selection logic would give CPU load, C*i*C FIFO level and *i*-let dispatch rate higher importance than temperature or ageing data while selecting a core.
- Similarly, for a control application to be able to react quickly on updated sensor data, it needs **Low Latency**, thereby requiring a minimal FIFO fill level and fast *i*-let dispatch rate from the C*i*C's FIFOs.

Thus, based on the application class information, the C*i*C computes cost/benefit values for each core using normalised weights as mentioned in Table 4.1 and selects the core for infection which provides maximum gain. The *i*-let assignment policy can be set by configuring these weights via the OctoPOS/*i*RTSS driver interface. If desired or needed, the policy can be fine-tuned even during the run-time of an application by updating the weights.

To adhere to the promised guarantees made by the Agent System to the application, further work is under progress to enhance the rule based evaluator within the C*i*C to perform *i*-let assignments taking into account the claim information of the application.

ViPG

The classification of applications and the abstracted monitor values provided by the C*i*Cs are used to drive the ViPG's dynamic power management policies. The integration of ViPG in the overall system is depicted in Figure 4.12.

From the hardware perspective, an **E**nergy **M**anagement (EM) block is integrated into the C*i*C which drives the power down signals for the Leon3 cores. To demonstrate ViPG policies in the central CHIPit demonstrator platform (Project Z2), we enhanced the Leon3 processor with clock gating capabilities. We extended applications from the *parsec* benchmark suite to express its core requirements to an in-house Linux kernel interface. When the application invades additional cores, these cores are woken up, when cores are retreated, the cores are clock gated immediately. We used a Xilinx Virtex5 board (similar to the FPGAs in the central CHIPit demonstrator platform (Project Z2)) with built-in power meters to directly present the impact of the activation and deactivation of the cores.

The driver to perform clock gating of the Leon3's is already integrated into the C*i*C's FIFO dispatching logic. The information of an empty FIFO can be transmitted to the SW layers including the ViPG to integrate into the policy decisions. As current technology prevents to perform power gating in the central CHIPit demonstrator platform (Project Z2), we are currently examining how to mimic the temporal and the energy overhead of power gating. The deactivation and activation of a power gated core has a considerably higher overhead as the (one cycle) clock gating. We therefore are investigating the power gating overhead of the Leon3 in an ASIC synthesis to reintegrate it into the EM-block.

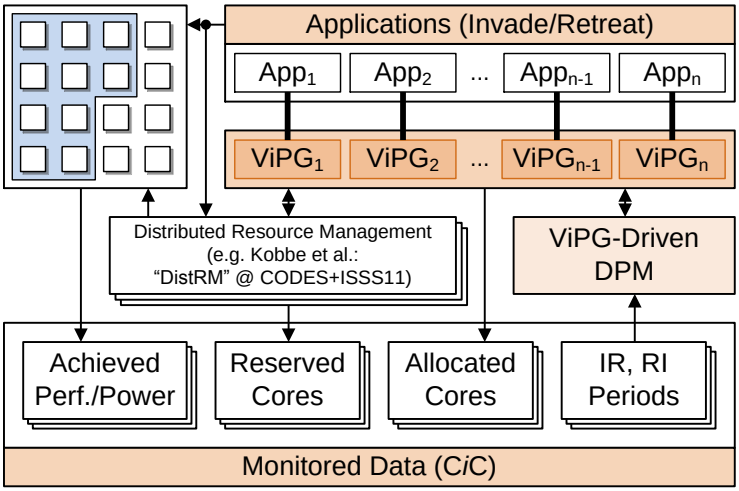


Figure 4.12: Integration of ViPG in the invasive system.

Besides our effort in the integration, a self-adaptive, hybrid DPM policy has been built this year and will be presented in [SVH13]. Invasive applications adapt their core requirements depending on their application-specific knowledge. This knowledge is expressed through the *constraints* and *hints* in the invasive language (see Project A1). In such a scenario, invasive applications *invade* and *retreat* cores in quick succession. The *invade* and *retreat* periods of applications are dependent on the application-specific knowledge and might occur uncoordinated. It may happen that an application retreats its dispensable cores and later tries to invade them again when the cores are already invaded by another application. These unavailable cores would lead to a performance degradation and thus to a loss of energy efficiency. We compensated this energy loss by a hybrid prediction approach of the *invade/retreat*

periods. An overview of the flow in the ViPG managers is depicted in Figure 4.13.

Based on our analysis, we concluded that it might be, from an energy perspective, better to *temporarily* reserve dispensable cores, i. e., to virtually power gate them. These virtually power gated cores can be put in the appropriate sleep state as the point in time they are needed again is accurately predicted. Moreover, the wakeup overhead can be hidden from the applications. We analysed the behaviour of several applications and classified them into different classes to be used in our hybrid prediction. System feedback capabilities compensate for competing applications as the energy efficiency improvement of one application might lead to performance loss of another application in uncoordinated systems.

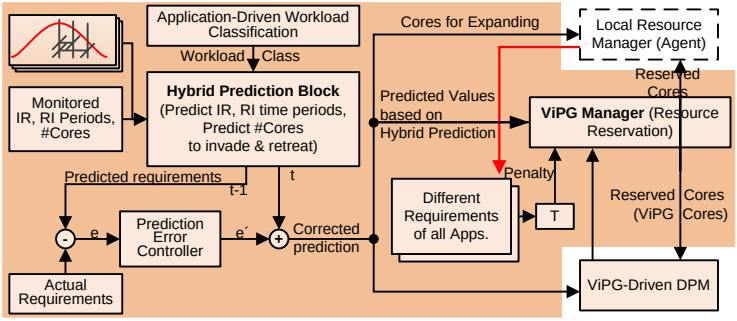


Figure 4.13: Detailed flow of the ViPG managers.

Summarising, we save the energy overhead to deallocate and reallocate cores through the agent system (Project C1), and decrease performance loss due to uncoordinated *invade/retreat* patterns. To evaluate our approach, we enabled applications from the *parsec* benchmark suite to behave in an invasive fashion. As an example, a H.264/AVC encoder (x264) invades additional cores when the throughput constraint (e. g., in fps) is not met. Accordingly, an application retreats cores, if the throughput constraint is reached. For these applications, ED^2P-savings in the range of $15\% - 40\%$ were achieved compared to recent related work (cmp. Figure 4.14).

We extended our power simulation infrastructure to determine the achievable energy savings. A cycle accurate performance simulator (gem5) is combined with an ITRS-based power model (McPAT) to determine power/performance trade offs. Moreover, a power state machine based on current processor architectures has been integrated to take

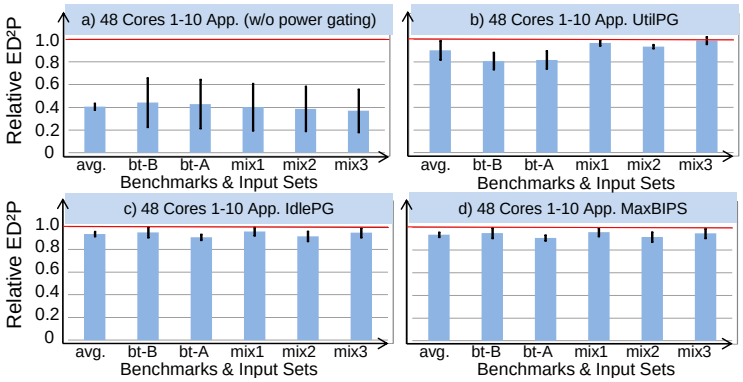


Figure 4.14: Relative ED^2P savings of the ViPG-based DPM.

the power gating overhead into account. Our simulation infrastructure is depicted in Figure 4.15. We are currently examining other performance simulators and behavioural simulations. An integration in the Project C2-temporal behavioural simulator is planned.

We further demonstrate the potential and benefits of leveraging the application-specific knowledge for dynamic power management with the help of a case study, a complex multiview video encoding application, where processing of different views compete for the compute and memory resources [SZSA+13]. The algorithm properties and the input data properties are jointly accounted to reach an application-driven DPM. In the multiview video encoding application, the different views are competing for the available memory resources. By reordering the processing of the different views at application level, we increased the potential to achieve energy efficiency improvements.

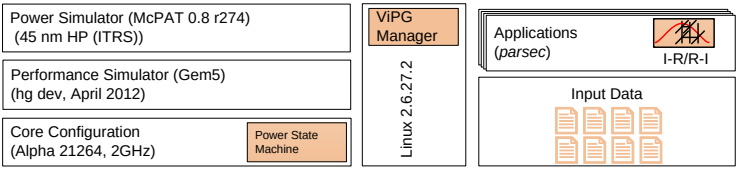


Figure 4.15: The ViPG simulation infrastructure.

We participated in the extension of the invasive language (LCPs), especially in the integration of application hints in cooperation with Project A1, Project C1, and Project D3. The cooperation with the agent system (Project C1) led to interfaces to include energy-efficiency

enhancements in the agent system decisions. Hints expressed by invasive applications allow the ViPG to decide then to virtually power gate retreated cores instead of retreating them to the agent system. These virtually power gated cores can be physically powered down in an appropriate sleep state to be quickly revealed to a reinvading application. The physical power down is performed by the aforementioned **e**nergy **m**anagement block in the C*i*C.

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. ISBN: 978-1-4673-0770-3. DOI: 10.1109/ASPDAC.2012.6164944.

[SZSA+13] F. Sampaio, B. Zatt, M. Shafique, L. Agostini, S. Bampi, and J. Henkel. "Energy-Efficient Memory Hierarchy for Motion and Disparity Estimation in Multiview Video Coding". In: *Design Automation and Test in Europe Conference (DATE) (to appear)*. Grenoble, France, Mar. 2013.

[SVH13] M. Shafique, B. Vogel, and J. Henkel. "Self-Adaptive Hybrid Dynamic Power Management for Many-Core Systems". In: *Design Automation and Test in Europe Conference (DATE) (to appear)*. Grenoble, France, Mar. 2013.

B4: Hardware Monitoring System and Design Optimisation for Invasive Architectures

Doris Schmitt-Landsiedel, Ulf Schlichtmann

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

With the resource-aware programming support in invasive computing systems, applications get the ability to explore the system and make decisions for execution (e. g., number of processors to execute on) based on the current state—including physical hardware properties—of the hardware platform. For a realisation of invasive architectures, a closed-loop between applications, operating system/agent system and the underlying hardware is necessary. This results in the need to monitor and regulate physical hardware conditions. And it becomes even more important—especially with thousands or more processors on a single chip—when considering the significantly different processing capabilities and susceptibility to degradation of modern integrated circuits.

The research goal of Project B4 is to measure and preprocess the specific status of hardware elements. Different monitor types are necessary, including temperature evolution, power consumption, reliability, and maximum and age-dependent performance capability. This information is communicated—with different levels of detail—to other system components like higher hardware layers, operating system (agent system) and applications. The system is then able to act considering monitor information when, e. g., choosing processing elements to execute applications on or react when a critical status is detected. In turn these actions may influence the status of hardware elements and with that the measured monitoring data. Project B4 considers optimisation strategies and the design of corresponding circuits and interfaces, including the demonstration by simulation and emulation on the FPGA hardware prototype platform.

Different task allocation techniques and application characteristics as well as different physical conditions such as package types, material parameters and cooling all result in different temperature scenarios. Also economical processor packaging cannot cover the worst case hot

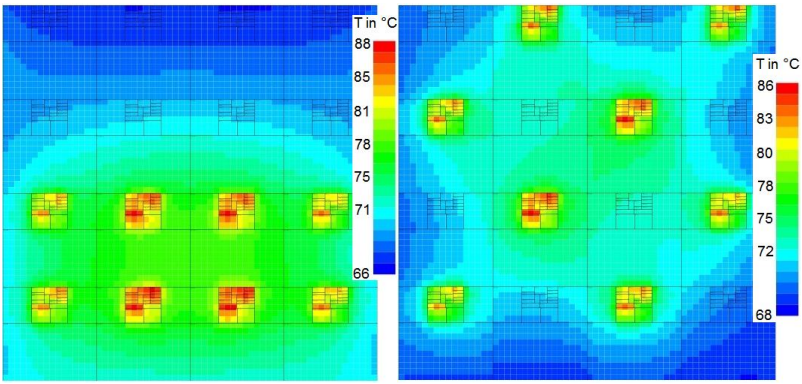


Figure 4.16: For a multicore system with 50% usage scenario different active core placements can be chosen. Compared to a square configuration (left side), a checkerboard-configuration (right side) decreases maximum and hotspot temperature by 3%.

spot scenario anymore. In [GS13] we modelled different scenarios in a multicore system and evaluated the temperature distribution of the cores. In a multicore system a reciprocal influence between the core temperatures of neighbouring cores occurs, so an intelligent active core placement in a non-full-usage scenario can further decrease the present temperatures as shown in Figure 4.16. We also evaluated different temperature limiting measures: The best choice is either an intelligent core choice combined with lower than 100% usage-rates or lowering of the input power, e. g., by implementing supply voltage or frequency scaling or use of intelligent task fragmentation techniques. Since temperature should be regulated during run-time, we recommend an implementation of different concepts and choosing the appropriate temperature limiting measure for the individual situation during run-time.

In [WHAP+12] we demonstrated the use of in-situ delay monitors for use in adaptive voltage scaling (AVS) and evaluated the performance. In-situ delay monitors are enhanced flip-flops that observe the timing of the circuit. Critical, 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. When using this monitor type in an online AVS technique, the supply voltage

can be regulated during normal circuit operation—without need for test intervals. In [PHSG+12] different designs to implement in-situ delay 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.

[LCXS13] investigates the challenges in hierarchical timing analysis considering process variations. With abstract statistical timing models containing interfacing constraints, this flow can reduce the complexity of design and verification of large SoC systems effectively. For each of the three basic circuit types—combinational, flip-flop-based and latch-controlled—methods to extract statistical timing models are proposed to prune the unnecessary timing information from the underlying modules. With additional methods for the reconstruction of correlation between modules and for system-level verification, the complete framework is several times faster than applying it to the flattened circuit directly, therefore providing an efficient flow for statistical timing verification of invasive architectures.

[LCS12] evaluates the statistical timing performance of circuits with level-sensitive latches, which are widely used in high-performance designs, such as CPUs. Circuits of this type, however, impose more complexity in timing analysis due to latch transparency. With reduced iterations and graph transformations, the proposed method extracts setup time constraints at latches and across sequential loops very efficiently, more than 10 times faster than other state-of-the-art methods, while still maintaining a good accuracy in the computed minimum clock period in a parametric form. The proposed method contributes a fast tool for statistical timing evaluation in the optimisation iterations of invasive systems, in which the aforementioned circuits always serve as the source of flexibility and robustness.

[CLS12] introduces a modelling framework for the timing behaviour of a flipflop by building a nonlinear functional relationship between the clock-to-q delay and the data/clock alignment. The proposed framework makes it possible to carry out static timing analyses at gate level taking into consideration the interdependency of different computation stages. An iterative timing analysis method is developed to find out whether a circuit can work at a given clock frequency and to determine the minimal clock period of the circuit. The new method will be able to further improve the performance and the yield of the ASIC design for invasive architectures, especially when process variation is considered.

We are currently working on integrating our temperature simulation

framework in the invasive functional simulator environment (Project C2) and use temperature monitor emulation for the Tightly-Coupled Processor Arrays (TCPAs, Project B2).

We are also working on the implementation of an energy model for invasive loosely-coupled MPSoCs. We apply an energy estimation methodology based on instruction-level characterisation for a Leon3 core, as well as for an integrated FPU. Figure 4.17 shows the flowchart of our approach.

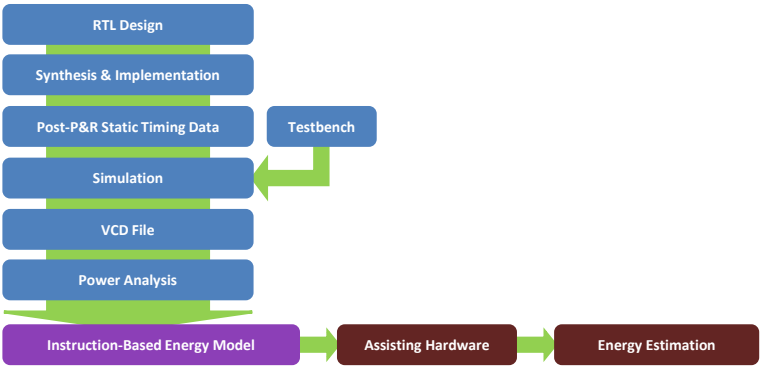


Figure 4.17: Flowchart of instruction-based energy estimation

Firstly, the RTL design of a Leon3 core for MPSoC is synthesised and implemented for an FPGA using Xilinx ISE tools. After that, post-place-and-route static timing data will be generated, which makes it possible to run post-P&R timing simulations with testbenches feeding different CPU or FPU instruction sequences. During the timing simulations, VCD files are dumped, which provide the power analyser, i. e., Xilinx XPower with accurate signal activities. And then, by analysing the floorplan, physical constraints and signal activities, XPower is able to generate power reports using Xilinx cell libraries. Based on those power reports for running different instruction sequences, an instruction-based energy model containing an energy look-up-table for different instructions or different combinations of instructions is created. And finally, an integrated assisting hardware for MPSoCs will be able to report energy estimations for a specific piece of code running on a processor. At run-time, this information will be able to help the operating system decide how to distribute *i*-lets to meet specified power constraints. As our current work is based on an MPSoC implemented in FPGA, the same approach will also be carried out later for tightly-coupled processor

arrays, and for the ASIC implementation of invasive architectures once ASIC floorplans are available.

Publications

[CLS12] N. Chen, B. Li, and U. Schlichtmann. “Iterative timing analysis based on nonlinear and interdependent flipflop modelling”. In: *Circuits, Devices Systems, IET* 6.5 (Sept. 2012), pp. 330–337. ISSN: 1751-858X. DOI: 10.1049/iet-cds.2011.0347.

[GS13] E. Glocker and D. Schmitt-Landsiedel. “Modeling of Temperature Scenarios in a Multicore Processor System”. 2013.

[LCS12] B. Li, N. Chen, and U. Schlichtmann. “Statistical Timing Analysis for Latch-Controlled Circuits with Reduced Iterations and Graph Transformations”. In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*. Nov. 2012, pp. 1670–1683.

[LCXS13] B. Li, N. Chen, Y. Xu, and U. Schlichtmann. “On Timing Model Extraction and Hierachical Statistical Timing Analysis”. to appear in: IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems. 2013.

[PHSG+12] N. Pour Aryan, L. Heiss, D. Schmitt-Landsiedel, G. Georgakos, and M. Wirnshofer. “Comparison of in-situ delay monitors for use in Adaptive Voltage Scaling”. In: *Advances in Radio Science (ARS)*. Vol. 10. 2012, pp. 205 –208.

[WHAP+12] M. Wirnshofer, L. Heiss, A.N.Kakade, N. Pour Aryan, G. Georgakos, and D. Schmitt-Landsiedel. “Adaptive voltage scaling by in-situ delay monitoring for an image processing circuit”. In: *IEEE 15th International Symposium on Design and Diagnostics of Electronic Circuits & Systems (DDECS)*. Apr. 2012, pp. 205 –208. DOI: 10.1109/DDECS.2012.6219058.

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. Invasive computing architectures [HHBW+12] include 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 4.18, 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 [HKB12], this requires novel protocols and hardware methodologies.

As part of the overall decentralised resource management, we investigated decentralised embedding strategies for task graphs called *self-embedding* within the *i*NoC. We extended an initial protocol to support not only tree-shaped topologies but also communication graphs with multiply predecessors. The self-embedding functionality is implemented itself as a hardware component inside the dedicated control network along with *i*NoC-internal invasive features, such as, end-to-end flow control and global self-optimisation.

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 (BE) communication channels can be used where no guarantees are

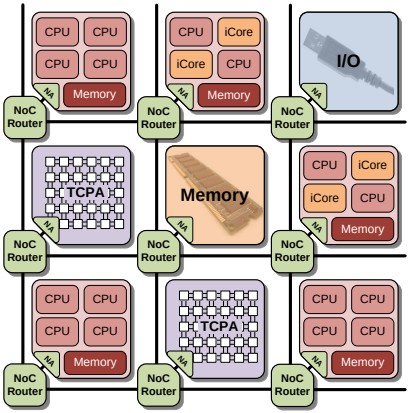


Figure 4.18: Heterogeneous invasive architecture consisting of heterogeneous tiles which are connected through the *i*NoC

B5

required without the need of any channel reservation. Thereby, BE channels may utilise resources that are not occupied by GS at that moment. In addition, we also investigated region based policies for GS and BE connections [HZWK+12]. With this technique, it is possible to adjust the resource utilisation on a region basis depending on the requirements of the application executed in the region. This strategy is beneficial for region-based resource management as developed by Project C1. The regions based policy reconfiguration mechanism is shown in Figure 4.19.

In addition to region based mechanisms, a self-optimisation strategy called Rerouting was investigated and realised in 2012. Rerouting enables to move existing GS connections in case of congestion of particular links. A congestion situation is detected by the *i*NoC router and an existing connection is selected for Rerouting. The selected connection is rerouted transparently from the perspective of the node using this GS connection. Rerouting enables to balance the load of the network and increases the probability of invading communication resources successfully. In Figure 4.20, the mapping of an MPEG task graph is investigated with and without rerouting. Rerouting increases the probability to map a task graph successfully if the architecture is utilised by additional GS connections.

To investigate the above mentioned invasion strategies and self-optimisation mechanisms at a higher level of abstraction, we developed a cycle-accurate SystemC model of the *i*NoC. It is highly parameteris-

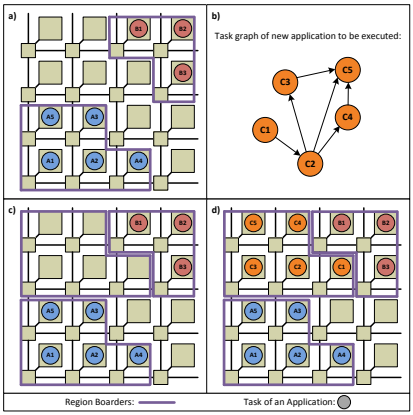


Figure 4.19: Policy reconfiguration for application mapping: a) Two applications are running, b) New application needs to be mapped, c) Region for new application is prepared by policy reconfiguration within the NoC routers, and d) Application mapped into new region and executed

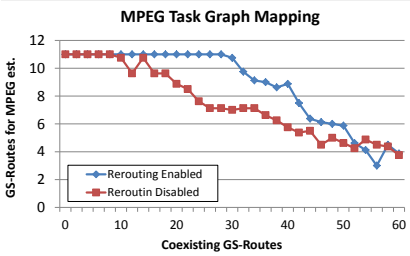


Figure 4.20: MPEG task graph using 11 GS connection mapped to the *i*NoC-based MPSoC architecture with additional GS connections

able and can be used for design space exploration as well as to provide performance characteristics for the simulation of invasive applications and invasive architectures (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 parameterability of the *i*NoC RTL implementation enables to instantiate a very light weight version of the *i*NoC [PHWAA+13] as well as complex ones [HKB12]. This RTL model of the *i*NoC plays a central role within the final integrated demonstrator platform [BFHK+12] integrated in Project Z2. The parameterability enables to setup different demonstrator platform

instances with low overhead. To approach that goal, a rudimentary architecture consisting of RTL models of Leon3 tiles, the NA and the *i*NoC was realised in collaboration with Project Z2 and 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 architectures. Only the tile interface layer needs to be replaced depending on the tile type.

To optimise the performance of the InvasIC architecture, two features have been added in 2012 with major contribution of the Project B5. An L2 cache was added between the tile local AHB bus and the Network Adapter to reduce the load on the NoC for tile external memory accesses. The second feature is a DMA unit which is located in each Network Adapter to enable efficient transfers of huge chunks of data between tile local memories and external DDR-RAM.

In 2013, Project B5 will further contribute to the demonstrator integration activities of the Project Z2. Besides more advanced self-optimisation features we will investigate the following: Inside the network adapter, a mechanism will be implemented to automatically detect communication partners which are addressed frequently. After a detection, the network adapter might then automatically setup a GS connection to such nodes to improve the performance of the communication and reduce the energy consumption for communication. Inside the *i*NoC routers, a prediction mechanism will be implemented. This mechanism might be used also to trigger power gating of *i*NoC components to reduce the static energy consumption in case of low utilisation.

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.

[HKB12] J. Heisswolf, R. König, and J. Becker. “A Scalable NoC Router Design Providing QoS Support Using Weighted Round Robin Scheduling”. In: *Parallel and Distributed Processing with Applications (ISPA), 2012 IEEE 10th International Symposium on*. July 2012, pp. 625 –632. DOI: 10.1109/ISPA.2012.93.

[HZWK+12] J. Heisswolf, A. Zaib, A. Weichslgartner, R. König, T. Wild, J. Teich, A. Herkersdorf, and J. Becker. “Hardware-assisted Decentralized Resource Management for Networks on Chip with QoS”. In: *Proceedings of the 19th Reconfigurable Architectures Workshop (RAW 2012)*. Shanghai, China, May 2012, pp. 1 –8.

[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. ISBN: 978-1-4673-0770-3. DOI: 10.1109/ASPDAC.2012.6164944.

[PHWAA+13] C. Pham, J. Heisswolf, S. Wenner, Z. Al-Ars, J. Becker, and K. Bertels. “Hybrid Interconnect Design for Heterogeneous Hardware Accelerators”. In: *Proc. Design, Automation & Test in Europe Conference & Exhibition, to appear*. Grenoble, France, Mar. 2013.

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 non-functional properties (e. g., energy consumption, timeliness).

Architectural Overview

Figure 4.21 provides a high-level view of the current *i*RTSS architecture, which we have refined further in 2012. Key elements are OctoPOS, the parallel operating system (POS) that 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 to cope with the scalability problem in large multicore systems. In order to provide these services also to C/C++ applications, the agent system has been ported from X10 to C++ and is now considered as part of the operating system layer, logically residing between the new operating-system abstraction layer (OSAL) and the OctoPOS kernel.

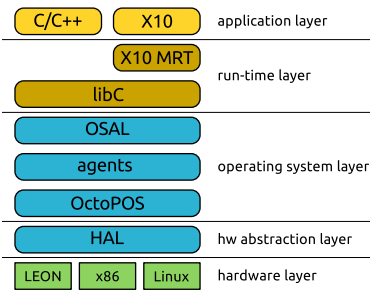


Figure 4.21: *i*RTSS Architecture

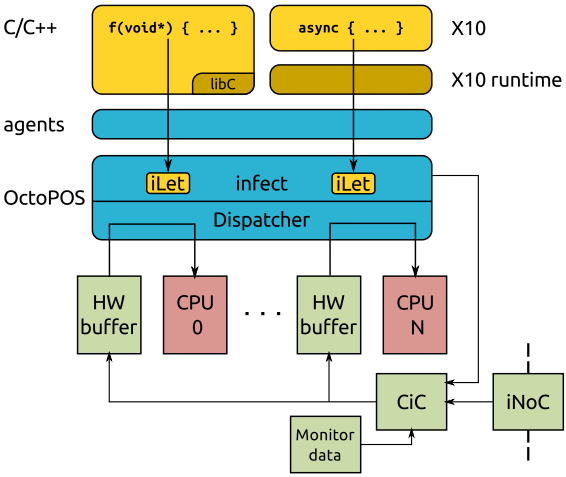


Figure 4.22: CIC-supported *i*-let dispatching in OctoPOS

The Configurable OctoPOS Kernel

C1

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 by tailoring operating-system mechanisms towards the hardware capabilities—and vice versa.

In this realm, we have designed in 2012 the central OctoPOS abstractions and integrated them with the hardware designs of the CiC and the iNoC. We have further investigated and quantified the benefits of "clever" hardware (mis-)use on commodity hardware for scheduling and dispatching of thread control flows: up to 170x lower latencies and the complete elimination of noise [HDMS+12]. These insights culminated in the ultra-light-weight OctoPOS control-flow abstraction—the OctoPOS *i*-let—and, in close collaboration with Project B3, its hardware-based low-noise/-latency dispatching by the CiC.

An OctoPOS *i*-let consists of just a code pointer and a data pointer and describes a (fundamental) block of the parallel program (Figure 4.22); large quantities of them are (tile-locally) scheduled and dispatched by the CiC purely in hardware. OctoPOS *i*-lets are considered to be short and to run to completion, even though OctoPOS provides support for blocking *i*-lets (Figure 4.23).

On top of the *i*-let abstraction we designed, in close collaboration with Project B3 and Project B5, a fundamental and novel abstraction for low-noise/-latency cross-tile interaction: Figure 4.24 depicts the

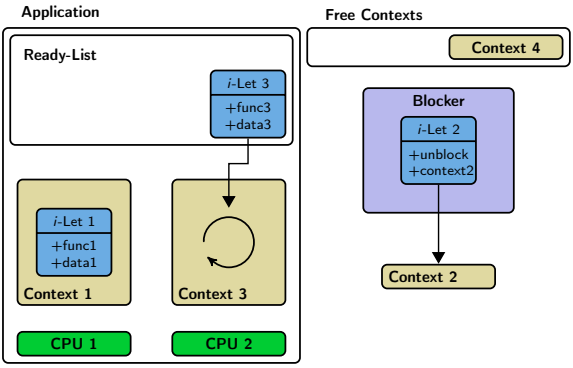


Figure 4.23: *i*-let incarnation and execution in OctoPOS

OctoPOS concept of a *dual-ported active message* (DPAM), which (upper half) consists of an (optional) message buffer and two *i*-lets that specify the after-delivery activities. A DPAM is given to the iNoC, which delivers it to the target tile (lower half) and, upon successful delivery, hands over the *i*-lets to the respective local CiCs to dispatch the related activities on *both* the source and the target tile.

C1

Nevertheless, we provide OctoPOS for a variety of platforms—with and without dedicated hardware support. The x86 bare-metal and Linux "guest mode" family members (Figure 4.21) ease the development and evaluation of the invasive software stack (applications, X10 extensions, compiler). They also make it possible to transfer recent techniques for energy-aware development and precise energy estimations [HEKSP12; HKSP12] to the OctoPOS development. Optimising for non-functional properties, such as energy, however, requires a fine-grained configuration and adaptation of all low-level OctoPOS abstractions, which we provide for by our approach of aspect-aware operating-system design [LSHSP12; MFGSP12].

The Agent System

To obtain an estimation of the overhead of our Agent System in a real hardware environment before the InvasIC demonstrator is available, we implemented our work presented in [KBHL+11] on top of Linux on the Intel SCC, a 48-core system. The Intel SCC communication infrastructure is optimised for running one big application (using all cores) which is programmed to expect specific messages in the right order. However, our Agent System requires multiple concurrent receivers

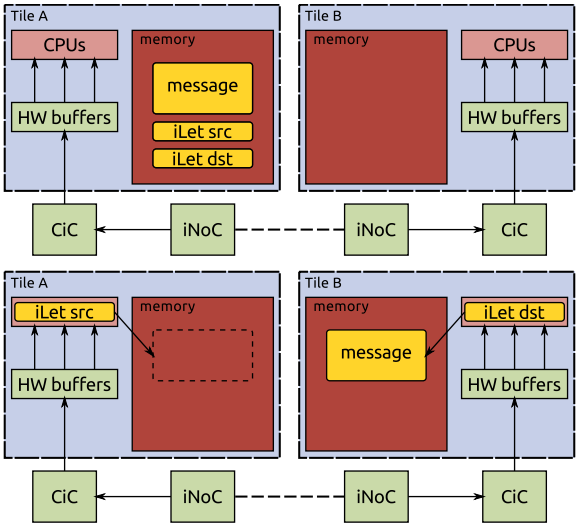


Figure 4.24: Cross-core interaction by dual-ported active messages.

C1

with many possible kinds of messages that should be handled. Therefore we had to implement a middleware to multiplex the communication infrastructure. The overhead of running one instance of Linux and our middleware on each core (i. e., 48 instances of Linux) influenced the measurements of the latencies of our Agents heavily, resulting in a high variance of the measurements. A "bare metal" implementation would be required to allow more accurate measurements. However, as the InvasIC demonstrator and OctoPOS are (close to being) operational by now, the focus will be measurements on that platform.

As the internal state of a many-core chip is not visible to the human eye, we designed a many-core demonstration platform that directly allows to visualise a) the ongoing communications in such a chip and b) the mapping of applications to cores. Each core of the system is represented by an 8-bit micro-controller, communication happens by using four serial links per micro-controller, creating a fully meshed network. The firmware running on each micro-controller implements a NoC emulation that allows for packet-based communication between any two cores in the network. Ongoing communications are visualised by two LEDs per link. To allow larger many-core systems to be emulated, the platform consists of stackable modules of four cores each. Figure 4.25 shows photos of the demonstration platform. We implemented a variant of or Agent System on this platform which gives direct insight on how

the Agent-based bargaining leads to the mapping of applications to cores. The claims of different applications are indicated by different colours of an RGB LED per core.

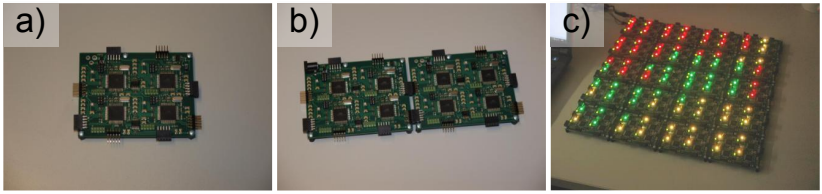


Figure 4.25: Photos of a) a single module consisting of four nodes, b) two modules stacked together, and c) our many-core demonstration platform running our Multi-Agent system on 80 cores

The previous decision to implement the Agent System in the InvadeX10 framework has been revised towards an implementation of the Agent System directly within the *i*RTSS using the C++ language. This design change led to the requirement of an interface between the X10 language features (i. e., invade and retreat) and the C++ implementation of our Agent System. This interface has been defined in cooperation with Project C3, and has been implemented in a first version. Future versions of the OctoPOS will include the Agent System.

Currently, we are refining our Agent System simulation environment used for design-space exploration. The new simulation back end will allow a more accurate analysis of the latencies of the Agent System and the direct influence of these latencies on the application performance. Therefore, a more accurate *i*NoC model (Project B5) as well as an improved application model is required. While heterogeneous resources already are supported in principle, an application performance model that is able to estimate the performance of an application (i. e., the foundation of our agent-based bargaining and theoretical strategies developed by Project A3) on our heterogeneous architecture is still under development.

Several LCPs have been proposed and/or advanced, mostly in cooperation with Project A1, Project B3, and Project D3. Interfaces on how to include the *i*-Core and the TCPA into the resource management have been discussed. Our cooperation with the ViPG (Project B3) led towards interfaces that include energy-efficiency enhancement strategies into the Agent System decisions. Hints expressed by the invasive applications are forwarded to the ViPG. Based on the application's hints, the ViPG can decide to virtually power-gate individual cores instead of retreating

from them. These virtually power-gated cores can be quickly revealed to a reinvading application.

Outlook

In 2013, we will finalise the interfaces towards the application/compiler interface (project area D, Project C3) and the invasive hardware platform (project area B, in particular Project B2). An important milestone in this realm is the stepwise integration of our results into the planned demonstrator scenarios of Project Z2. Residing between "a rock and a hard place" (application and hardware projects), Project C1 continues to play a central role in this setting.

Publications

[HDMS+12] W. Hofer, D. Danner, R. Müller, F. Scheler, W. Schröder-Preikschat, and D. Lohmann. "Sloth on Time: Efficient Hardware-Based Scheduling for Time-Triggered RTOS". In: *Proceedings of the 33rd IEEE International Symposium on Real-Time Systems (RTSS '12)*. (To appear). IEEE Computer Society Press, Dec. 2012. URL: http://www4.cs.fau.de/Publications/2012/hofer_12_rtss.pdf.

[HEKSP12] T. Hönig, C. Eibel, R. Kapitza, and W. Schröder-Preikschat. "SEEP: exploiting symbolic execution for energy-aware programming". In: *ACM SIGOPS Operating Systems Review* 45.3 (Jan. 2012), pp. 58–62. ISSN: 0163-5980. DOI: 10.1145/2094091.2094106.

[HKSP12] T. Hönig, R. Kapitza, and W. Schröder-Preikschat. *ProSEEP: A Proactive Approach to Energy-Aware Programming*. Poster. June 13–15, 2012, Boston, MA, USA, 2012.

[JH13] J. Jahn and J. Henkel. "Pipelets: Self-Organizing Software Pipelines for Many Core Architectures". In: *Design Automation and Test in Europe Conference (DATE) (to appear)*. Grenoble, France, Mar. 2013.

[JKPC+12] J. Jahn, S. Kobbe, S. Pagani, J.-J. Chen, and J. Henkel. "Work in Progress: Malleable Software Pipelines for Efficient Many-core System Utilization". English. In: *Proceedings of the 6th Many-core Applications Research Community (MARC) Symposium*. Ed. by E. Noulard and S. Vernhes. Toulouse, France: ONERA, The French Aerospace Lab, July 2012, pp. 30–33. URL: http://hal.archives-ouvertes.fr/hal-00719027.

[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.

[LSHSP12] D. Lohmann, O. Spinczyk, W. Hofer, and W. Schröder-Preikschat. “The Aspect-Aware Design and Implementation of the CiAO Operating-System Family”. In: *Transactions on AOSD IX*. Ed. by G. T. Leavens and S. Chiba. Lecture Notes in Computer Science 7271. Springer-Verlag, 2012, pp. 168–215. DOI: 10.1007/978-3-642-35551-6_5.

[MFGSP12] T. R. Mück, A. A. M. Fröhlich, M. Gernoth, and W. Schröder-Preikschat. “Implementing OS Components in Hardware using AOP”. In: *ACM SIGOPS Operating Systems Review* 46.1 (Jan. 2012). Best Papers from 2011 Brazilian Symposium on Computing Systems Engineering (SBESC), pp. 64–72.

[TSDSP+12] R. Tartler, J. Sincero, C. Dietrich, W. Schröder-Preikschat, and D. Lohmann. “Revealing and Repairing Configuration Inconsistencies in Large-Scale System Software”. In: *International Journal on Software Tools for Technology Transfer (STTT)* 14.5 (Feb. 2012), pp. 531–551. DOI: 10.1007/s10009-012-0225-2.

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 techniques that enable the validation and variants' exploration of all essential features of invasive computing. It has two major research fields: (a) Timed functional 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 therefore provides evaluation facilities and enables the optimisation of the concepts of invasive computing across all project areas, especially, without the need to have full hardware or software implementations available.

Timed Functional Simulation

The main idea of timed functional simulation is to provide a simulation platform for invasive parallel applications written in X10. Here, the simulator shall allow to model the underlying architecture at a high abstraction level and to change quickly its parameters such as topology, tile size and processor types. Furthermore, it allows to simulate the effects of resource invasion to gain important insights into the execution behaviour of invasive applications, like the interference with other applications, etc. These interactions are depicted in Figure 4.26. In summary, timed functional simulation shall allow for an early validation of invasive programming concepts as well as the investigation of a broad range of different invasive hardware platforms, but already with timing information. First concepts of timed functional simulation for resource-aware programming were published in [RHT12a] and a first case study

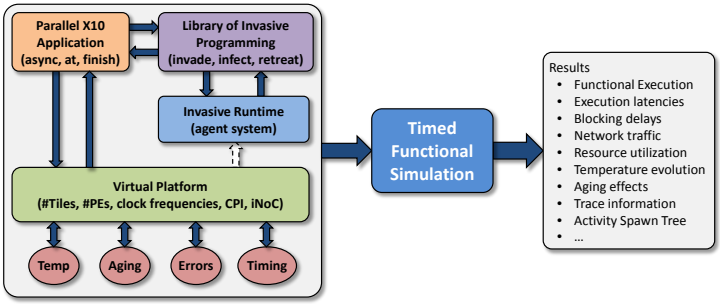


Figure 4.26: Overview of the resource-aware timed functional simulation approach.

of an architecture evaluation has appeared in [RHT12b]. Moreover, we presented our simulation framework at the HiPEAC ACACES summer school 2012 [RHT12c].

The main activities throughout the last year considered the integration of the simulation kernel into the X10 run-time system, the design of an *invasive Network-on-Chip* (*i*NoC) simulation model in order to determine also communication latencies between activities on different tiles, and the implementation of a basic resource management interface. This allows not only the integration of different resource management strategies but also studying different physical effects such as temperature, ageing and timing by simulation models in a modular manner. Furthermore, we organised a simulation workshop where the participants got the opportunity to work with the simulator and contribute to following developments with knowledge in their specific area.

C2

The main reason for the integration of the simulation kernel into the X10 run-time was that an invasive X10 application would be highly restricted in the use of X10 constructs elsewise. For example, no blocking statements (e. g., *finish*, *when*, or *clocks*), no communication primitives (e. g., *at*, *dist array*, *remote arrays*), and no parallel constructs (e. g., *async*) were allowed. The integration of the simulation kernel into the X10 run-time allows to handle all of these constructs within the simulation, because each X10 language construct is mapped to an X10 run-time call and can therefore directly trigger the simulation primitives. Details about the simulation kernel, which is composed of a) a timing model based on performance counters, b) analytic models (physical models such as temperature), and c) a synchronisation mechanism were presented in [RHT12a]. Now, all of the X10 constructs can be directly used within an application and the result of the simulation is a

timed execution of the application on the modelled invasive architecture with respect to the heterogeneous characteristics of different processing resources including communication latencies between tiles over the *i*NoC.

The latter was achieved through the design and implementation of a timed simulation model of an invasive network that provides similar communication latencies like in a real *i*NoC. The communication demand of an application is given by the program itself as follows: In X10, *at* statements represent the transport of data to remote places. The resulting amount of data that is transmitted to another tile (*place* in X10) is obtained from an X10 run-time call to the X10 communication library (X10RT) and fed into the simulation model to calculate the latency of the corresponding data transfer. The interference of several activities communicating over the network concurrently is also taken into account in our simulation model.

Another ongoing work, in cooperation with Project C1, is the design and implementation of a distributed agent-based resource management at X10-level. Here, different invasion strategies are implemented and evaluated in terms of the quality of resulting claims, e. g., the average speedup of all applications.

Summarising, the 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 the key feature of this timed functional simulation framework. As experiments in [RHT12b] show, our simulation is much faster than cycle-accurate simulation and thus allows the investigation of the behaviour of invasive applications with hundreds of *i*-lets running on parameterised heterogeneous invasive architectures.

Integrated Simulation Methodology

The integrated simulation methodology allows to model and simulate resource-aware applications on heterogeneous invasive architectures by integrating different architectural simulators in a modular way. In contrast to the functional simulator, this methodology gives much more detailed insights in the considered target architectures (e. g., memory and bus transactions, accurate timing). Separating the invasive control flow from the non-invasive parts of an invasive application is the most distinctive feature of the applied simulation methodology. It works by providing centrally defined interfaces to a controlling instance, which will separate the different modules of the simulation framework from each other. The details of this methodology were de-

scribed in [GHHH+12]. This enables to transparently exchange single modules without the need of adapting any other module. The developers can combine simulation methods with a varying level of complexity or granularity without modifying any other attached module. This enables to investigate the exact influences of certain algorithms or invasive hardware features.

Integrated Invasive Simulation Framework The part of the framework, which captures the invasive control flow of applications, consists of Abstract Invasive Machines (AIM) and a Resource Manager module (RM). An AIM is associated to an instance of an invasive program and executes its appropriately abstracted representation. For this purpose, it interacts with the RM, which takes care of invasion requests.

The non-invasive parts of invasive applications are simulated on Platform-Specific Engine modules (PSE), which represent different target architectures. The specific input required by a PSE is provided by the AIMs depending on the outcome of invasion requests.

The central component of the simulation framework is a centralised instance, which takes care of controlling the attached modules, synchronising the simulation steps and offering an abstracting communication layer to the simulation modules.

In the course of the last year, we specified a C++ API and built an implementation of the aforementioned concept in form of a centralised daemon (*invasicd*). This implementation includes a library (libiisf), which encapsulates all the communication channels between the different modules as shown in Figure 4.27.

In addition, a first prototype of a generic AIM implementation was built during the last year; the work in the following year will focus on creating a basic RM, providing the possibility of easily manipulable but transparent resource awareness to the applications running on top of the integrated invasive simulation framework.

After finishing the implementation of the synchronisation mechanisms inside *invasicd*, a transparent communication layer for data exchange between the simulation modules and the *invasicd*, running independently on the simulation host, will be developed. The modules need to adapt their communication interfaces to the new API. This will also be finished in the following year.

Platform Simulation Modules In close cooperation with Project B2, we have extended the platform-specific simulator for TCPAs in order to investigate different hierarchical power gating techniques. More specific,

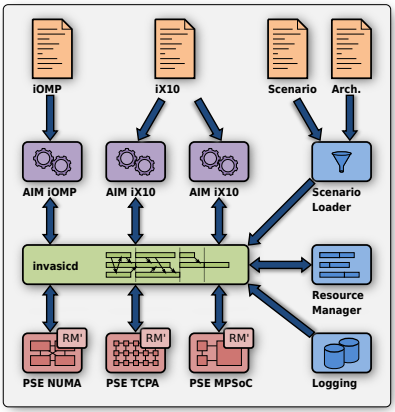


Figure 4.27: The architecture of the Integrated Invasive Simulation Framework from a developer's point of view as described in [GHHH+12].

we studied how processor invasion can be directly exploited for power management in TCPAs. Here, several hierarchical power gating techniques with different power domain granularity have been developed and evaluated with respect to latency and static power savings by using the TCPA simulator [LMBH+12a; LMBH+12b].

Moreover, the architecture simulator for loosely-coupled MPSoC (LCMPSoC) is currently being extended for inter-tile investigations. For this purpose, a transaction level model of an *i*NoC is under development, which will consider the communication at packet-level instead of flit-level to achieve higher simulation performance. In order to support simulation of real world applications on the LCMPSoC simulator, the applications are analysed and represented in a graph that represents the invasive control flow. For Example, an audio processing application from Project D1 will be used for initial investigations. After the finalisation of the *invasicd* API, the LCMPSoC simulator interfaces are currently being adapted for coupling it to the *invasicd*. This enables the LCMPSoC simulator to communicate with other modules (e. g., AIM, RM).

A new module will be the HPC focused NUMA simulator whose requirements were analysed in cooperation with Project D3. Application developers require data determined by simulations to understand the behaviour of their algorithms on invasive hardware. In 2012, the implementation of the simulator front end, which is responsible for tracing of a running application, was started and will now be refined to estimate detailed costs per instruction stream. Running a full memory

hierarchy simulation of a many core system in a serial simulator would take a lot of time for each single instruction. In the course of the NUMA simulator architecture development it became clear that such a full simulation is not needed for most scientific codes, as long as the shared memory stages are simulated in a synchronised or serialised matter. Due to the characteristics of most scientific simulation codes in the HPC area, the accuracy lost by this optimisation will be small compared to the overall simulation time. Therefore, we will develop a fully parallel non-shared cache simulation stage in the following year, which will make use of all available compute resources on the simulation host.

Publications

[GHHH+12] M. Gerndt, F. Hannig, A. Herkersdorf, A. Hollmann, M. Meyer, S. Roloff, J. Weidendorfer, T. Wild, and A. Zaib. "An Integrated Simulation Framework for Invasive Computing". In: *Forum on Specification & Design Languages (FDL)*. Vienna, Austria, Sept. 18–20, 2012, pp. 185–192. ISBN: 978-2-9530504-5-5.

[LMBH+12a] V. Lari, S. Muddasani, S. Boppu, F. Hannig, and J. Teich. "Design of Low Power On-Chip Processor Arrays". In: *Proceedings of the 23rd IEEE International Conference on Application-specific Systems, Architectures, and Processors (ASAP)*. Delft, The Netherlands: IEEE Computer Society, July 9–11, 2012, pp. 165–168. ISBN: 978-0-7695-4768-8. DOI: 10.1109/ASAP.2012.10.

[LMBH+12b] V. Lari, S. Muddasani, S. Boppu, F. Hannig, M. Schmid, and J. Teich. "Hierarchical Power Management for Adaptive Tightly-Coupled Processor Arrays". In: *ACM Transactions on Design Automation of Electronic Systems (TODAES), accepted for publication* 18.1 (Dec. 2012).

[RHT12a] 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. ISBN: 978-1-4673-0770-3. DOI: 10.1109/ASPDAC.2012.6164943.

[RHT12b] S. Roloff, F. Hannig, and J. Teich. "Fast Architecture Evaluation of Heterogeneous MPSoCs by Host-Compiled Simulation". In: *Proceedings of the 15th International Workshop on Software and Compilers for Embedded Systems (SCOPES)*. St. Goar, Germany: ACM Press, May 15–16, 2012, pp. 52–61.

[RHT12c] S. Roloff, F. Hannig, and J. Teich. “Simulation of Resource-Aware Applications on Heterogeneous Architectures”. In: *Proceedings of the 8th International Summer School on Advanced Computer Architecture and Compilation for High-Performance and Embedded Systems (ACACES)*. Fiuggi, Italy, July 8–14, 2012, pp. 127–130. ISBN: 978-90-382-1987-5.

C3: Compilation and Code Generation for Invasive Programs

Gregor Snelting, Jürgen Teich

Matthias Braun, Sebastian Buchwald, Frank Hannig, Manuel Mohr, Ericles Sousa, 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 tightly-coupled processor arrays (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 invaded processors.

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 4.28. For the concrete language and its interfaces as defined in Project A1, a compiler is being developed. Here, two compiler back ends for both loosely- and tightly-coupled invasive

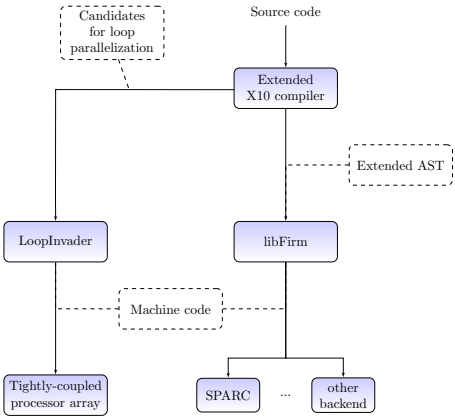


Figure 4.28: Compiler framework for Invasive Computing.

multi-processor architectures are distinguished. For SPARC-based processors, the back end is based on the FIRM infrastructure for code optimisation. FIRM provides static single assignment (SSA) form as a basis for program analysis and optimisation. Due to the completely different architecture of TCPAs and their massively parallel computing capabilities, a dedicated back end for TCPAs needs to be developed. This invasive loop compiler, called LoopInvader, implements symbolic loop parallelization techniques as investigated here as well as code generation techniques for TCPAs.

The compiler is based on the existing X10 compiler, but with front end extensions for invasive constructs and a new back end for libFIRM to support SPARC architectures, as well as optimisations for efficient utilisation of invasive hardware and operating system support, respectively. In order to exploit invasive computing concepts at the level of loop programs, LoopInvader has the important task of extracting potential loop candidates from X10, and then to convert them into single assignment code (SAC), which will reveal the entire data parallelism in the loop. Next, symbolic techniques are currently developed 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 after invasion, hence at run-time. Here, the main challenge is to define such parameterised schedules and mappings together with their proper mathematical foundations, as well as to derive compact case-dependent processor and time mappings.

First prototype code generators for loosely-coupled RISC-type processors as well as for TCPAs have been successfully developed within the last year. The output of the code generators produce assembly code for the architectures investigated in Project B2 and Project B3.

Symbolic Loop Parallelization (LoopInvader)

In this part of the project, we focus on compiler transformations for the parallel execution of invasive loop programs on processor arrays such as TCPAs. A simplified drawing of a TCPA with 24 processor elements (PEs) is sketched in Figure 4.29. Here, the different rectangular areas denote three applications running simultaneously on the array. Whereas static mapping and loop parallelization techniques for coarse-grained reconfigurable and TCPA architectures are well studied, we proposed and formalised for the first time symbolic tiling [THT12b] as an automatic program transformation for symbolic parallelization of nested loop programs with uniform data dependencies. As shown in [THT12a], this symbolic loop parallelisation step is essential for invasive program-

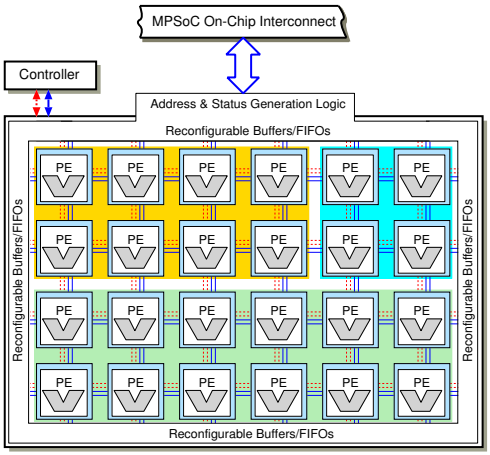


Figure 4.29: Tightly-coupled processor array (TCPA)

ming on MPSoCs, because the number of processors, which determines the shape and size of tilings during parallelization, is not known until run-time.

For executing loop programs on a processor array of fixed size, tiling is needed as a compiler transformation that assigns iterations within a tile of a loop to one available processing element each. For illustration, consider the nested loop program in Figure 4.30(a) and its iteration space visualised for $N = 6$ and $M = 4$. Each node represents an iteration of the loop program, that is, the execution of the loop body for an iteration vector (i_1, i_2). Data dependencies between different iterations are depicted by directed edges. Now, tiling increases the depth of the loop nest, in our example from a 2-deep nest to a 4-deep nest and the tile size is typically chosen statically to reflect the number of available processors in the architecture. For instance, let the number of tiles reflect the number of processors, then all the intra-iterations need to be executed by one processor sequentially. The size and shape of a tile can be represented by a so-called tiling matrix P. Figure 4.30(b) shows the resulting code of a statically tiled loop with 3×2 tiles. After tiling, the innermost loop iterates over the iterations contained in a tile and the outer loop iterates over the origins of the tiles. However, 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. Following this approach is obviously not feasible as the number of different configurations and thus the amount of necessary instruction memory would explode easily. The other possibility might be just-in-time compilation. However, a compiler framework could consume easily dozens to hundreds of megabytes of memory. In addition, parallelization and mapping of a loop program is a time-consuming process. Therefore, this approach is usually also not viable for embedded architectures. Due to the aforementioned arguments, we study the idea of *symbolic* loop tiling. Here, we consider a tiling transformation to be specified by a parametric tiling matrix $P = \text{diag}(p_i)$. For example if at run-time, a processor array of size $n \times m$ has been successfully invaded, a tiling of size $p_1 = \lceil N/n \rceil$, $p_2 = \lceil M/m \rceil$ would be then needed to map the loop program onto the considered processor array. An example of a symbolically tiled C loop and the corresponding symbolic tiling matrix P are shown in Figure 4.30(c). Whereas tiling is used mainly for iterations to processor assignment, the next task we are currently investigating is *symbolic scheduling*. A schedule is a mapping which assigns a time of execution to each computation of the nest in such a way that dependencies are preserved. In other words, scheduling is the process where to each loop iteration a start time of execution is assigned. Finally, like for tilings, symbolic schedules are parameterised in terms of tile sizes. For a given symbolically tiled

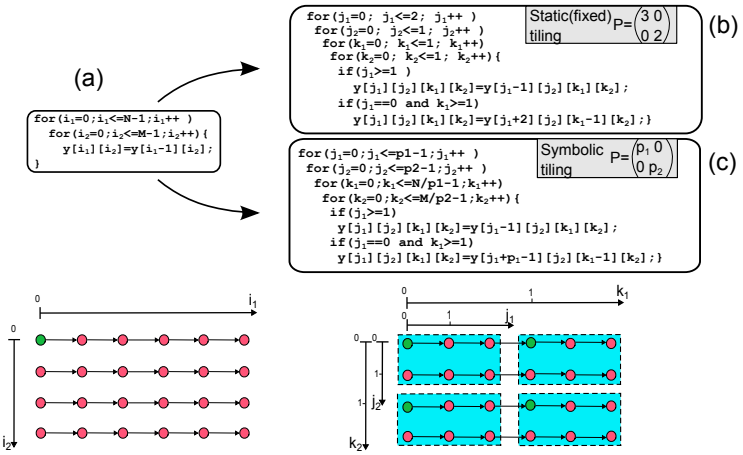


Figure 4.30: Tiling of a loop program (a), static tiling (b) for $p_1 = 3$ and $p_2 = 2$, symbolic tiling (c).

iteration space, a linear symbolic schedule denotes a $2n$-dimensional schedule vector $\lambda = (\lambda_J \ \lambda_K)$, where each schedule vector may involve arbitrary expressions including the tile parameters p_i. In the following, λ_J is called *intra-tile schedule*, and λ_K *inter-tile schedule*, respectively. A symbolic schedule vector is finally called feasible, if it satisfies all data dependencies d of the tiled algorithm. In the case of symbolic tiling, unfortunately, it is not possible to solve the problem of finding both a latency optimising inter-tile and intra-tile schedule vector easily because of products of tile parameters in corresponding scheduling constraints. Similar products of parameters and variables appear in the objective function. We therefore propose a four-step procedure as follows: First, we determine so-called *first tiles*, respectively *last tiles* of the iteration space of the symbolically tiled code. This step is of great importance because the latency L of a schedule λ is determined solely by the difference between the maximal time step (reached in a last tile) and the minimal time step (reached in the first tile). In the next step, we determine the set of all so-called *tight* intra-tile schedule vectors λ_J that schedule all $\det(P)$ iterations of a tile P sequentially. Then, for each feasible intra-tile schedule λ_J, we subsequently determine a latency-minimising inter-tile schedule λ_K such that the overall dependency L of the combined schedule $\lambda = (\lambda_J \ \ \lambda_K)$ is minimised. This will prove that only latency-optimal schedule candidates are determined. As a last step, we derive a set of conditions that choose the best statically determined schedule at run-time depending on the size of the invaded processor array.

C3

Hence, at run-time, only a few latency expressions need to be evaluated, and the corresponding minimal latency schedule determines then uniquely also the processor array loop codes that need to be loaded and executed. The presented symbolic tiling methodology has already been developed and implemented in LoopInvader. In 2013, we expect to finish the mathematical foundations for symbolic scheduling as well as integration and testing.

Code Generation for TCPAs and Resource Management for Other Targets

In order to map loop programs onto TCPAs, an appropriate code generation (assembly codes) is also needed. In close collaboration with Project B2, we studied first non-symbolic code generation techniques. In 2012, we managed to release the first version of a code generator as part of the back end of LoopInvader, and we are currently working on graph-based code optimisation techniques.

Feature	Status
firm java integration (jFirm)	completed
sequential code generation	completed
data structure layout	completed
firm object orientation support (liboo)	completed
dynamic dispatch	completed
interface calls	completed
generic code, name mangling	completed
intra-tile parallelism (`async`, `finish`)	completed
inter-tile dispatch (`at`)	ongoing
serialisation	completed
cross-compilation support	completed
garbage collection	ongoing
testsuite	125 tests

Table 4.2: iX10 Front End Feature Matrix

Furthermore, we have investigated invasion overheads for tightly-coupled processor arrays (TCPAs) that avoid the creation of threads and use hardware-based signalling concepts to invade processing elements. This results where compared with the overheads of invasion on existing MPSoC platforms, namely the Tilera TilePro64 architecture and on Intel's SCC [TWOS12].

X10 Front End and Intermediate Representation

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. A compiler targeting invasive computing architectures has to support a range of architectures including SPARC, TCPA and the *i*-Core. It has to use the *i*RTSS operating system and leverage the *i*NoC for DMA transfers.

We tackle these requirements by writing a custom back end for the existing X10 compiler front end. We call this extended X10 front end iX10. We transform the programs analysed by this front end into the libFIRM intermediate representation to facilitate generating SPARC code and supporting the *i*-Core extensions. This transformation also lowers parallel programming constructs like `async`, `at`, `invade` and `infect` into the lower level APIs provided by the *i*RTSS. We demonstrate this mapping from X10 language constructs to the *i*RTSS API using the following example:

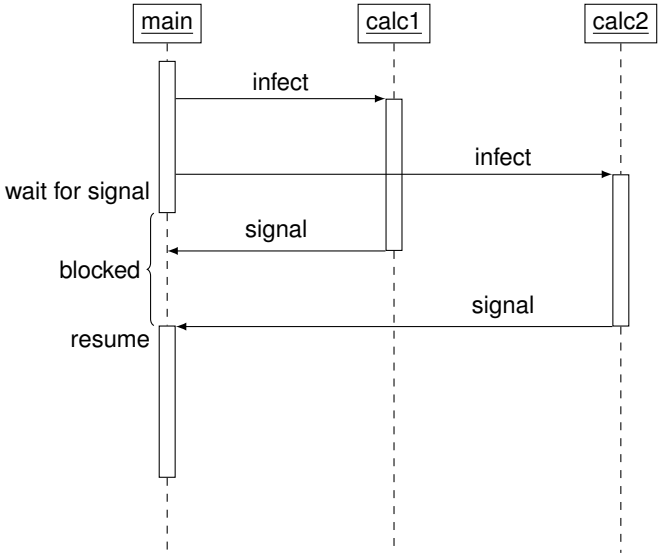


Figure 4.31: *i*-let interaction

```
finish {
        async { calc1(); }
        async { calc2(); }
}
```

C3

The X10 code calls the functions `calc1` and `calc2` asynchronously within a `finish` statement. The X10 front end compiles the code to the following calls to the X10 run-time library:

```
x10.lang.Runtime.finishBlockStart();
x10.lang.Runtime.executeParallel(()=>{calc1()});
x10.lang.Runtime.executeParallel(()=>{calc2()});
x10.lang.Runtime.finishBlockEnd();
```

Since the X10 run-time library implements partly functionality that is already provided by the *i*RTSS, we adapted the X10 run-time library to use the *i*RTSS API instead. For instance, the methods of our example look like this:

```
x10.lang.Runtime.finishBlockStart() {
        ...
        currentState.signal = simple_signal_init();
        ...
}
```

```
x10.lang.Runtime.executeParallel(closure) {
        ...
        infect(currentState.claim, closure);
        signal_add_callers(currentState.signal)
        ...
}

x10.lang.finishBlockEnd() {
        ...
        simple_signal_wait(enclosing.signal);
        ...
}
```

Figure 4.31 shows the interaction of the involved *i*-lets at run-time.

Table 4.2 details the current state of the front end. The compiler now correctly transforms X10 programs with all modern language features such as closures, generic code, parallelization and synchronisation through `async`, `at` and `finish`. We also support serialisation and deserialisation, which is needed for inter-tile communication via `at`. Our next milestones are the support of `at` on *i*RTSS and the integration of the invadeX10 framework developed in Project A1.

Memory Hierarchy Optimisations

Compute tiles in the invasive architecture have tile local caches which are not synchronised, so there is no globally coherent view on the memory. This requires splitting the main memory into different regions for each tile, so that each region is only written to by one tile. Sending data from one tile to another requires using of the DMA support in the *i*NoC. To achieve good performance programs have to be written in a way that communication between tiles happens in bulked data transfers. A form of read sharing is possible between multiple tiles, if the software enforces a synchronised cache flushing protocol.

We developed multiple possibilities to avoid and optimise inter-tile data transfers in X10 programs as described in [BBMZ12].

SPARC Back End

The general purpose cores found in our investigated invasive computing 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 SPEC CPU2000 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 shows that register allocation for programs in SSA form leads to an optimal assignment of registers. Translating out of SSA form, however, requires parallel copy constructs, which are traditionally implemented by sequences of register-register copy and exchange instructions. In practice, minimising the number of needed parallel copies is an NP-hard problem, which is why the remaining number of parallel copy operations can be quite high. We greatly reduce the cost of implementing parallel copies with an instruction set extension that adds additional instructions to permute the register contents within a single cycle.

The extended instruction set has been defined in its final form in collaboration with Project B1. We have developed an efficient compilation approach that is able to generate optimal code for all practically relevant parallel copy constructs using the additional instructions. Multiple register allocators that were already implemented have been adapted to support the generation of permutation instructions. Furthermore, we have modified the CPU emulator QEMU to support the additional instructions in order to get accurate measurements on the number of saved instructions.

With the modified QEMU version and the FPGA-based hardware implementation provided by Project B1, we were able to conduct benchmarks using real programs. We used the programs from the SPEC CPU2000 benchmark as inputs to the compiler and evaluated the qual-

ity of the generated code. To find out under which circumstances the permutation instructions offer the biggest benefit, we tested different compiler configurations. We varied the used register allocator and its parameters as well as the used coalescing strategy. It seems that the permutation instructions are especially useful in just-in-time compilation scenarios because here, the compiler cannot apply a computationally costly algorithm to minimise the number of parallel copies.

We plan to test more compiler configurations, where we artificially restrict the register allocator to use fewer registers and try to simulate more register constraints in the instruction set. This will hopefully allow us to predict how architectures that are different from SPARC would benefit from permutation instructions.

Publications

[BBMZ12] M. Braun, S. Buchwald, M. Mohr, and A. Zwinkau. *An X10 Compiler for Invasive Architectures*. Tech. rep. 9. Karlsruhe Institute of Technology, 2012. URL: `http://digbib.ubka.uni-karlsruhe.de/volltexte/1000028112`.

[THT12a] A. Tanase, F. Hannig, and J. Teich. "Symbolic Loop Parallelization of Static Control Programs". In: *Proceedings of the 8th International Summer School on Advanced Computer Architecture and Compilation for High-Performance and Embedded Systems (ACACES)*. Fiuggi, Italy, July 8–14, 2012, pp. 33–36. ISBN: 978-90-382-1987-5.

[THT12b] A. Tanase, F. Hannig, and J. Teich. "Towards Symbolic Loop Parallelization for Tightly-Coupled Processor Arrays". Work-In-Progress Presentation at the 49th Design Automation Conference (DAC), San Francisco, USA. June 3–7, 2012.

[TWOS12] J. Teich, A. Weichslgartner, B. Oechslein, and W. Schröder-Preikschat. "Invasive Computing - Concepts and Overheads". In: *Forum on Specification & Design Languages (FDL)*. Vienna, Austria, Sept. 18–20, 2012, pp. 193–200. ISBN: 978-2-9530504-5-5.

D1: Invasive Software–Hardware Architectures for Robotics

Rüdiger Dillmann, Tamim Asfour, Walter Stechele

Manfred Kröhnert, Johny Paul

In Project D1 we focus on exploring 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. The goal is to explore techniques of self-organisation to efficiently allocate available resources for the timely varying requirements of robotic applications. Compared to traditional resource allocation at compile time the resource-aware computing methodology is expected to lead to better load balancing and efficient resource utilisation. 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.

During the first year, the optical flow algorithm for robot navigation was implemented on TCPA with inbuilt resource-aware techniques. The implementations are described in the PAULA programming language and by now the complete algorithm runs on the TCPA simulator (Project C2). Additionally, an algorithm for calculating disparity maps was implemented in X10 using invasive methods.

In the second year, Project D1 focused on implementing algorithms which can be run on an FPGA-based demonstrator. The disparity map and object recognition are needed by the robot in order to recognise and grasp objects. An invasive algorithm for audio equalising was added to demonstrate the benefits of invasive computing on audio applications. In addition, a hardware monitoring framework was implemented to monitor behaviour and performance of algorithms on an FPGA. Work has also been started on motion planning and colour segmentation. Progress of the work is described in detail in the following sections.

The robot ARMAR-III currently uses an object recognition application consisting of three main stages. These include Harris Corner Detection, SIFT (Scale Invariant Feature Transform) feature extraction and SIFT feature matching. During the 2nd year of the project, Harris Corner Detection and SIFT feature extraction were converted to a multi-threaded parallel model capable of running on the proposed InvasIC hardware. The applications are now able to express their resource requirements to the run-time system. Currently, they execute on a single-tile hardware (standard Gaisler design) and are structured to be easily portable to the proposed multi-tile invasive hardware.

Application programs were profiled to understand their behaviour on single tile hardware. Also, evaluations were performed on how to efficiently use the core-local and tile-local memory blocks like scratchpad memories and tile local memories (TLM). A concept based on passing hints to the operating system (OctoPOS) for allocating stacks and heaps has appeared to be effective and results in simple and portable application code. Figure 4.32 shows how the execution time for Harris Corner Detection changes with available resources (only Leon3 PEs are used). AHB bus load and CPU usage (efficiency) are also provided along with execution time. It can be seen that the application scales well up to 10 cores and the execution time gets reduced when more resources (PEs) are being used. But the efficiency of the cores comes down as more cores operate in parallel due to the rising AHB bus load.

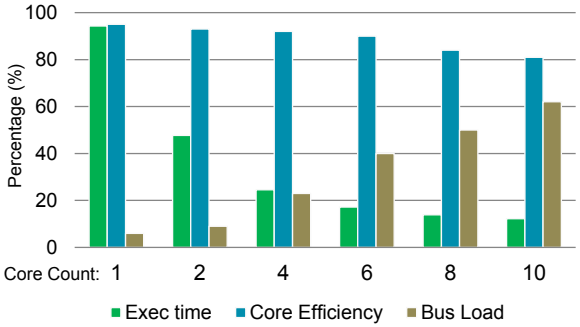


Figure 4.32: Performance plots for Invasive Harris Corner Detection application

Audio Processing

Speech synthesis and recognition are two important tasks for humanoid robots. An invasive version of such applications would result in better evaluation of invasive concepts. As the speech synthesis and recognition are very complex tasks and difficult to implement in the first phase of the project, a more suitable and simple audio application was selected. A multi-channel audio equaliser was used to demonstrate the benefits of invasive computing in audio applications. The application can react to changing resource conditions and adapt itself in order to deliver the best audio output.

Disparity Map on MPSoC

In the first year an invasive version of the Disparity Map algorithm was implemented using the InvadeX10 framework (Project A1) and simulated using the functional simulator of Project C2. Results were published in [PSKA+12].

In the second year, the application has been ported to C++ and adapted to run on the invasive OctoPOS operating system (Project C1). Using this approach makes it possible to run the application on real hardware to extract execution time and other metrics. This information can then be used to influence the design of invasive hardware and software components.

A single tile containing 6 Leon PEs was used to measure the performance of the implementation. Results are shown in Table 4.3.

Table 4.3: Versions of Disparity Map running on FPGA (Execution time in [ms])

#PEs	1	2	3	4	5	6
V 1 (without TLM)	8950	5130	3850	3540	3580	3700
V 2 (with TLM)	8700	4890	3500	2830	2480	2300

Version 1 of the algorithm can only access global DDR memory whereas Version 2 is capable of storing some of the data structures in tile local memory (TLM). Using TLM yields a boost in performance of up to 60 % (6 Cores) even on single-tile hardware. Additionally, performance degrades faster in Version 1 which is not using TLM.

Color Segmentation

Recognising and tracking of coloured objects can be realised using colour segmentation. The object colour to detect is given in the Hue-Saturation-Value (HSV) colourspace together with some margins. Region growing is then used to extract same coloured regions which in turn are matched against a database of known objects. If a pair of stereo cameras is used it is possible to calculate position and orientation of detected objects afterwards.

An initial version of the HSV segmentation, capable of segmenting coloured regions in images, has been ported to the OctoPOS operating system to run on a single PE. It can be run on the same FPGA hardware than the Disparity Map.

Motion Planning

Visually guided grasping is the final demonstration scenario of the first funding phase. So far, most presented algorithms are vision based. They are responsible for visually detecting and tracking objects including obstacles.

To complete the proposed scenario the robot needs to detect an object using vision. Afterwards this goal object must be reached without colliding with other objects. Collision free paths towards the goal object are calculated by means of motion planning. We are going to use randomised search trees, in particular Rapidly exploring Random Trees (RRTs), for motion planning.

In the second year, an initial version of the RRT was ported to run on OctoPOS. Motions between two positions can be planned and visualised in a 3D viewer. Collision checking will be implemented in the near future to allow for collision free paths.

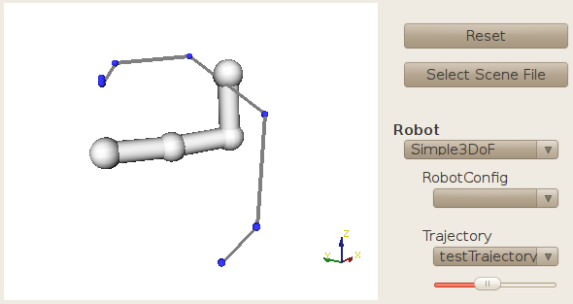


Figure 4.33: Visualisation of a trajectory planned for a simple robot with 3 joints.

Hardware Monitoring Framework

In addition to invasive applications, D1 also developed a monitoring framework for InvasIC. It includes performance counters incorporated into the existing hardware and additional visualisation tools that can plot the performance values in real-time on a host-PC. This framework is useful for evaluating the performance of application programs (count cache misses, cache latencies, FPU operations, CPI, AHB bus load etc.) and also helps to pin-point bottlenecks in the system. The present monitoring framework architecture is shown in Figure 4.34. Output from performance counters are read through the USB interface and visualised on a host PC. More parameters like power monitoring, link utilisation of the NoC, monitoring of the C*i*C rule evaluator and more will be added in the future.

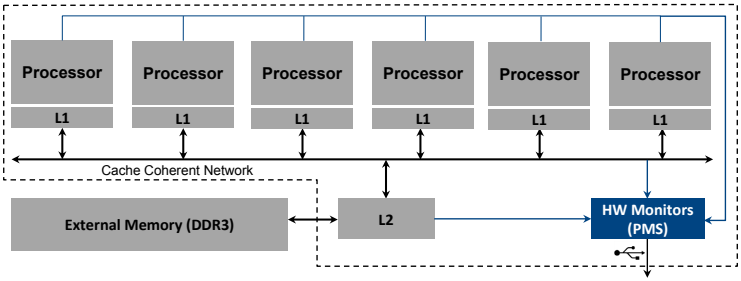


Figure 4.34: HW based performance monitoring framework

D1

Outlook

So far, several algorithms being used on the humanoid robot ARMAR-III have been investigated. Some of them already run invasively. Others were ported recently and do not take advantage of invasive computing.

In 2013, the benefits of resource aware programming will be studied more thoroughly. All presented algorithms will be enhanced to fully support invasive mechanisms. Once the applications are multi-tile ready, more evaluations on multi-tile hardware (including distributed memory) will be conducted early next year. Profiling information generated from test runs will then translate to hints to be passed to the run-time system (agents). Additionally, a more detailed evaluation of the Stack-/Heap-Mapping technique will be performed on multi-tile hardware. Here, more benefits are expected on hardware where external memory (DDR)

access is very expensive compared to the current single tile design with external memory attached to a single AHB bus.

However, testing single algorithms is not a real use case since there is no real competition for resources. Therefore, our plan is to combine several algorithms into a bigger scenario. Depending on the use case, some of the algorithms might run sequentially while others will run in parallel competing for the limited resources of the FPGA hardware.

To complete the demonstration scenario a connection between the robot ARMAR-III and the FPGA-based demonstrator hardware (CHIPit, Project Z2) will be established next year. Once finished, a closed loop between robot and the invasive architecture will be implemented on top. Based on images and joint angles read from the robot, a collection of invasive algorithms will calculate new commands which are then sent back to the robot.

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

There are two major research areas which we focus on in this project:

The first one is to provide numerical core algorithms omnipresent in scientific computing which are developed with the X10 programming language. These algorithms are extended and modified to be invasive. During this step, we give/receive feedback to/from other sub-projects via application-driven requirements analysis.

Our second research area is to examine the potential of Invasive Computing for state-of-the-art high-performance applications [BBGH+11; GHHH+12] on standard HPC architectures. We are developing iOMP as an invasive version of a standard programming model for HPC as well as an invasive Tsunami simulation code. Our main algorithm is based on a dynamical adaptive grid [SBB12] and is extended to make use of an invasive API. During the execution of multiple simulations in parallel, a resource manager [BBS12] determines an optimised distribution of computing resources to each simulation for improved utilisation of the hardware.

1) 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 of the hardware and operating system, D3 provides typical algorithms and algorithmic patterns from scientific computing.

In the 2nd year, we achieved our main goals for this year: We provided the basic scientific algorithms in X10 including utilisation of the currently existing invadeX10 API:

- **Multi-grid:** Multi-grid algorithms are one of the core algorithms in scientific computing to compute an approximated solution of a system of linear equations. During execution of this algorithm,

dynamically changing workload is created by restriction and interpolation operations. Due to this highly dynamic behaviour and changing resource demands of the multi-grid algorithm during its execution, we expect the most challenging demands on the invasive API—especially demands on data locality & migration issues.

Results: We implemented a multi-grid algorithm in X10 which computes the heat distribution of a laser beam on a metal plate [2]. This algorithm also makes use of the latest *invadeX10* API features using scalability graph hints as well as the max-core restrictions on each multi-grid level.

Outlook: Our next steps for this algorithm are, among others, to test the execution of multiple multi-grid algorithms in parallel on standard HPC platforms, to optimise the stencil computations of the algorithm with respect to halo-cells as well as to evaluate scheduling strategies of the agent system.

- **Quad-Tree:** To approximate integrals of functions which cannot be computed analytically, we use numerical quadrature. This is usually done by using recursive function calls. For Invasive Computing, this leads to the challenge of recursive invasions.

  Results: We extended our previously developed version of this algorithm to make use of the most recent *invadeX10* API using recursive invasion.

  Outlook: To compute numerical integrals (quadrature) by mainly evaluating computation intensive functions, this algorithm is clearly compute bound. We highly expect to show benefits of invasive computing by interleaving our computation bound quadrature algorithm with bandwidth limited algorithms once we are able to execute our algorithm on the demonstrator platform.

- **Matrix-exponentials:** Matrix-Matrix multiplications are one of the basic linear algebra algorithms in scientific computing. For invasive computing, we consider matrices of a size exceeding current L2 cache sizes (» 1 MB) creating additional demand on optimisation: Such optimisation are e. g. block-wise matrix-matrix multiplication, data-locality by using a space-filling-curve but also data-reutilisation by a software-managed Turing-cache.

[2] Video available @ `http://www.youtube.com/watch?v=vRQDr3zkupI`

Results: We extended our matrix-matrix multiplication algorithm to compute matrix-exponentials and used *invadeX10* to extend the block-recursive matrix-exponentials computation to an invasive application: During each matrix-matrix multiplication, the algorithm offers to be invaded (retreat) or to invade other resources. Since the performance hints, e. g. the scalability graphs, are not changing over time, this performance hints are provided to the agent system by an invade call at the beginning. To reduce the time for the serialisation of this constraints, a *reinvade()* without any parameters was implemented in the invadeX10 API with concrete results previously shown in [BBS12] on HPC systems (see below). This reinvade is used to (a) modify an existing claim and (b) to avoid forwarding the constraints by assuming that the previous constraints are reused.

Outlook: Next steps are further optimisation of this algorithm, e. g., support for different instruction sets (Leon, *i*-Core) and thus different performance on heterogeneous architectures while still maintaining load-balancing.

We frequently update all algorithms to stay up-to-date with the latest invadeX10 API provided by Project A1 and Project C2.

Performance measurements on the demonstrator platform and evaluation of the X10 compiler with smaller X10 programs are currently done.

Once our algorithms are able to run on the simulator, further performance optimisations in cooperation with other projects are done.

2) HPC Invasive Computing

A) Invasive OpenMP:

During the first year of the project we developed a first version of iOMP which supports the specification of constraints in a similar way as it is done in invadeX10. Experiments with iOMP applications run in the 2nd year showed that hard constraints, e. g., fixed number of cores, are difficult to accomplish if more then one application is running in parallel. We also experimented with soft constraints, i. e., ranges of cores, but this still is not sufficient for an effective resource management. In case of two applications that both ask for 1 to N-1 cores on a system with N cores, the first one will get nearly all cores while the second would get only one core. This first come first serve strategy doesn't work well. To get better distributions, we realised a uniform distribution of resources that gives the same number of cores to each application. This

strategy has shown its benefits for running many parallel instances of the tsunami simulation [GHMS+12]. This approach is, however, not very efficient for running different applications in parallel, since their scalability is different. The uniform distribution can be improved by specifying scalability hints.

A detailed evaluation of the overheads of invasive resource management in iOMP revealed that invade and retreat operations are expensive due to the interprocess communication between the resource manager and client and due to the pinning of threads to the process' resources. Therefore, a coarse grained invasion is important which can be realised by exploiting the iterative behaviour of scientific applications. Frequently a time stepping loop is executed where resource adaptation can happen for each time step but not for individual parallel regions in the time step execution. We extended the resource management to take the specification of the requested number of cores at the beginning of a time step as a hint for a request in the next time step, if it cannot be fulfilled due to a shortage of resources. If during the time step execution other applications retreat from resources, the resource management collects some of the resources for the invade in the next time step.

The distribution of resources is guided by the scalability hints discussed above. Those hints have to be gathered before the invasive execution and are clearly input dependent. In addition, the real scalability depends on the overall resource usage of all the applications running on the invasive computing system. For example, another application using a core sharing the L3 cache might limit the number of cache lines available to an application that invades another core sharing this cache.

Therefore we developed a concept to determine the hints or constraints for invasive operations dynamically at run-time. This approach is based on measurements with the hardware performance counters of modern processors. It takes into account the memory bandwidth requirement of the application, its execution time as well as the requirements of the other application competing for the resources. The current version is based on the perf event subsystem of the Linux kernel. The overall goal is to provide more information to the resource managements in order to optimises the overall resource usage [HG12].

Additionally, we implemented a tracing library for iOMP. It generates detailed trace files in the Open Tracing Format 2 (OTF2). This format is a successor of OFT1 and is currently under development within the BMBF-funded LMAC project. It is used as an input format for many advanced performance analysis tools. The traces contain trace records for invasive operations as well as for operations of the resource manager.

We use Vampir for the visualisation and analysis of those trace files. Vampir is a production quality visualisation tool for very large scale HPC systems.

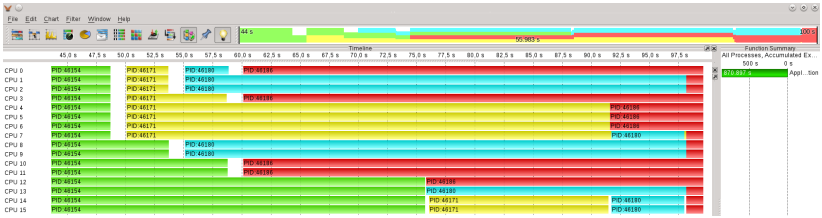


Figure 4.35: Time line diagram of an iOMP execution on the four socket Westmere EX server funded by InvasIC.

Figure 4.35 shows the dynamic distribution of resources when four iOMP application share the available resources. Each colour represents one application and each bar represents one CPU core. The resource management reacts on newly started applications. At first, CPU cores are withdrawn from one applications. In a second step, they are assigned to the requesting application. White gaps in the bars show that CPU cores are idle for a time until the requesting application arrives again at the invade command at the beginning of the time step loop. It demonstrates the effect of the above described resource management approach to take the constraint of an invade command as a hint, when not enough resource are available.

B) Invasive resource management:

D3

We also reached our main goal of the 2nd year: We implemented an invasive resource manager [BBS12] for OpenMP as well as TBB which is able to schedule resources in a globally optimised way with the help of resource-aware applications:

Our resource-aware application sends its scalability information (one slice in Figure 4.36) to the invasive resource manager. The resource manager distributes computational resources among all applications and ensures that CPUs are never oversubscribed. Resource management decisions are enforced by setting processor affinity. Our current implementation aims for optimisation of overall throughput by using scalability graphs. We also implemented asynchronous invasion (see [BBS12]) to reduce the overhead created by the latency of synchronous invasions.

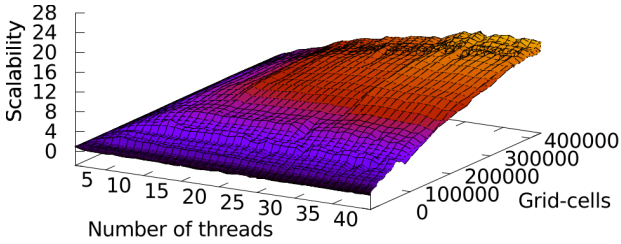


Figure 4.36: Changing scalability behavior depending on current phase (severe different number of cells) of simulation

C) Invasive application:

A fully-adaptive simulation [SBB12] based on shallow water equation is executed (see Figure 4.37). Such an adaptive simulation has changing demands on resources *during the simulation* due to adaptive refinement and coarsening. Our parallelization of this simulation is achieved via massive-tree splits and joins. With adaptivity, this leads to changing workloads and scalability during the execution of simulation. This different scalability is currently not considered by existing threading tools (e. g., OpenMP, TBB).

The simulation makes use of the invasive API between each time-step to provide necessary information about optimisation to the invasive resource manager.

Further information on our fully-adaptive framework are available at `http://www5.in.tum.de/sierpinski/`.

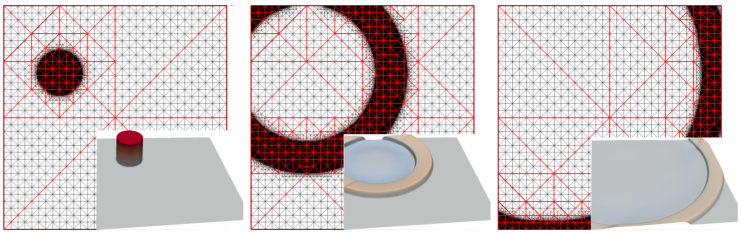


Figure 4.37: Changing number of grid-cells during the simulation. Red borders mark our smallest serial execution unit for the massive-tree splitting parallelization.

D) Invasive execution:

Figure 4.38 shows the globally optimised distribution of computing resources with respect to the scalability graphs of each application.

We developed a resource manager which distributes the cores to the application to **maximise global application throughput** depending on the current **scalability graph** and **core limitations** of the application.

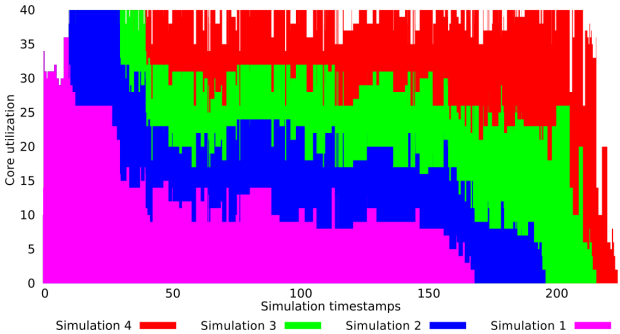


Figure 4.38: Invasive execution of a multiple simulations: (1) Since there is no gain in performance when application 1 is using more than 30 cores, the resource manager assigns only 30 cores for invasion. (2) During the start of the 2nd application (blue), less cores are assigned to it compared to the first application (red) since the scalability is worse. (3) Since resource redistribution of applications cannot be achieved synchronously, some resources idle for a short period (white areas on top of graph). Amongst the cost of the resource manager to search for the global optimum, this is the compromise to make to be invasive.

Our next tasks are e. g. to focus on the resource-aware extension and invasive execution of other standardised run-time-dynamical applications.

Collaboration within the SFB

D3

We steadily extended our algorithms in close collaboration with C1 and C3 to support the most recent features of invadeX10 and to add further features necessary to support invasion (see e. g. *reinvade()* in the matrix exponential algorithm). To support invasion within the multigrid algorithm, we determined new requirements for the X10 API as well as the operating system and discussed this with C1, C2 and C3 for further improvements.

To make an early evaluation of the demonstrator hardware, we developed simplified non-invasive algorithms in X10 and C including execution and evaluated this programs on the final demonstrator platform in collaboration with C3 (Compilation), C1 (Operating system OctoPOS) and Z2 (Demonstrator). This gave us better understanding of

the underlying hardware for our algorithms.

Besides the before mentioned cooperations, others are e. g.:

- B3, B5, C1, D1: We updated the matrix-exponential description including discussions about further demands.
- A1, C3: In order to optimise the algorithm by means of performance and invasion concepts, we are in frequent contact with the A1 and C3.
- C3, C2: Due to the development of the multigrid algorithm, we determined new X10 API features. E. g. an extension of the information on data distribution with PEs as well as data migration when re-infecting PEs different from the previous ones.
- B1: Acceleration of matrix-matrix sub-block multiplication using *i*-Core.
- B3: Energy efficiency optimisations for dynamical adaptive algorithms.
- C1, D1: Application driven considerations on tile- or core-granular invasions.

Publications

[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.

[BBS12] M. Bader, H.-J. Bungartz, and M. Schreiber. “Invasive Computing on High Performance Shared Memory Systems”. In: *Facing the Multicore-Challenge III*. 2012.

[GHHH+12] M. Gerndt, F. Hannig, A. Herkersdorf, A. Hollmann, M. Meyer, S. Roloff, J. Weidendorfer, T. Wild, and A. Zaib. “An Integrated Simulation Framework for Invasive Computing”. In: *Forum on Specification & Design Languages (FDL)*. Vienna, Austria, Sept. 18–20, 2012, pp. 185–192. ISBN: 978-2-9530504-5-5.

[GHMS+12] M. Gerndt, A. Hollmann, M. Meyer, M. Schreiber, and J. Weidendorfer. “Invasive computing with iOMP”. In: *FDL*. 2012, pp. 225–231.

[HG12] A. Hollmann and M. Gerndt. “Invasive Computing: An Application Assisted Resource Management Approach”. In: *MSEPT*. 2012, pp. 82–85.

[SBB12] M. Schreiber, H.-J. Bungartz, and M. Bader. “Shared Memory Parallelization of Fully-Adaptive Simulations Using a Dynamic Tree-Split and -Join Approach”. In: *Proceedings of HiPC 2012*. 2012.

Z: Central Services

Jürgen Teich, Jürgen Kleinöder, Katja Lohmann

Ina Derr, Frank Hannig

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 travelling abroad. Technical support and tools for communication and collaboration are provided as well as support and organisation of central publications. Last but not least, financial administration and bookkeeping is one of the central services.

The second part concerns public relations. Contacts with important research sites are established as well as an international Industrial and Scientific Board. Scientific ideas and results are 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` is maintained.

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

The goal of Project Z2 is to build a common FPGA-based demonstrator for validating the principles of invasive computing on a real compute platform. For this purpose the contributions of the different projects will be integrated into an invasive compute architecture to demonstrate the advantages of invasive computing such as improved quality of service, resource utilisation and speed-up of applications.

As validation and demonstration platform, we use a Synopsys CHIPit Platinum system (shown in Figure 4.39 on the right) at each site (FAU, KIT, TUM) since the beginning of the collaborative research centre. Thanks to its 6 FPGAs (Virtex5 LX330), the system allows for the prototyping of multi-million gate designs. In addition, the CHIPit platform enables co-simulation/emulation and transaction-based verification.

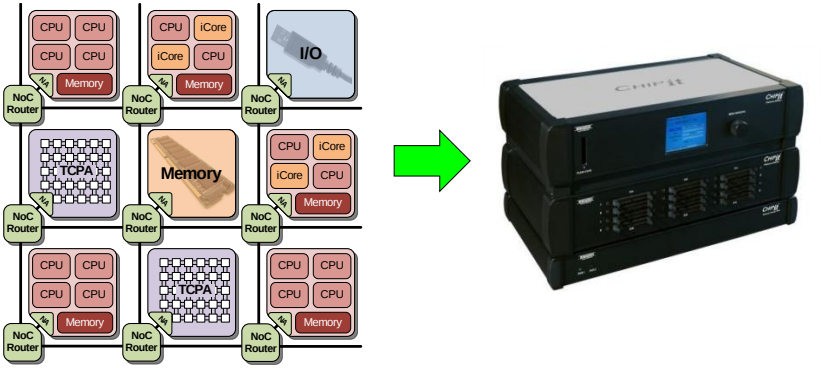


Figure 4.39: *InvasIC Demonstrator*: a concept validation demonstrator of an NoC-based 3×3 tiled array, consisting of different compute tiles (Leon cores, tightly-coupled processor arrays (TCPA), *i*-Core) as well as of memory and I/O tiles, will be prototyped on the CHIPit system.

In general, the work of Project Z2 can be subdivided in the following three major areas:

- Provisioning of a basic hardware infrastructure on the CHIPit system, encompassing common peripherals and components as well as the associated tool and design flow support.

- Integration of the different projects' contributions, i. e., hardware and software components, into the common demonstrator.

- Coordination work for definition and stepwise integration of the contributions, including milestone planning.

In the past year the following major achievements have been accomplished by Project Z2, which make up first important steps towards the final invasive computing demonstrator:

- A first demonstration, across all project areas (application, compilation, architecture), in form of a matrix-matrix multiplication algorithm running on a two tile Leon-based multiprocessor architecture was assembled and shown at Doctoral Researchers Retreat in spring 2012.

- An invasive TCPA architecture, which processes in real-time[3] video data from a DVI source and outputs the results as well by DVI, has been designed and prototyped on the CHIPit system. The design was demonstrated at the Invasive Computing annual meeting in October 2012 as well as at the demo night of DASIP 2012 [MBHK+12].

- Whereas the above demonstrators are mapped only to one FPGA of the CHIPit system, we can highlight also successful multi-FPGA designs, including a 2×2 tiled architecture mapped onto four FPGAs.

In the following sections, more detailed information is given on the three work areas mentioned above and the associated achievements in the last year.

[3] Processing of a video stream with a resolution of 1024 × 768 pixels and 60 frames per second.

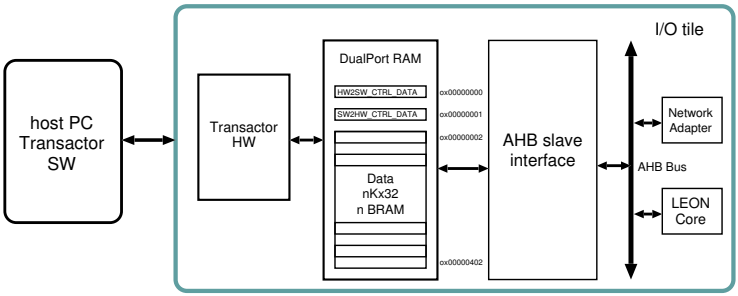


Figure 4.40: *I/O Transactor*: Architectural Block Diagram

Basic Demonstrator Components and Design Flow

The development, customisation and integration of both virtual and physical interfaces as well as drivers and controllers made up an extensive part of last year's work in Project Z2. Given IP modules from Synopsys and Gaisler had to be configured and appropriate wrapper components had to be designed to enable integration into the common demonstrator architecture. During the last year, special focus was put on several interfaces and controllers, which are essential for the communication between the host PC or peripherals and the actual FPGA system of the CHIPit platform.

I/O Transactor An I/O transactor transfers data between the host PC and the invasive compute architecture in the CHIPit system or vice versa. The transactor based interaction opens up the possibility to boot, configure and debug all processor cores in arbitrary tiles of the architecture. Further on, it allows access of different memory locations (global memory, tile local memory, etc.) within the architecture. The transactor consists of a software and a hardware part as shown in Figure 4.40. Currently, a slave I/O transactor design (see Figure 4.40) has been released while the development of a master transactor, which allows initiating data transfers from the host PC, is in progress.

DDR2 Controller Memory is one of the limited resources of the FPGAs within the CHIPit system when we implement the invasive compute architecture, which requires different memory (type, size) for caches, TLM and main memory. In addition to the SSRAMs, which are already located on the FPGA boards of the CHIPit system, we extended the system with two DDR2 extension boards. As background for the memory

controller we are using the DDR2_SPA controller from Gaisler. Since the pin placement of the CHIPit system is not as flexible as usual Virtex5 FPGAs, we needed to implement an appropriate physical interface for the controller. This includes the insertion of an additional signal to handle the bidirectional data and data strobe signals, because there is a bidirectional level shifter placed between the FPGA and the memory device on the CHIPit system. This signal indicates whether the signals are used as inputs or outputs. In addition the data strobe signals are required differential bidirectional signals for the memory device, but the CHIPit pin placement does not allow differential signals. Hence these signals are changed into inverted signals. Because of the limited frequency of all extension boards, the DDR2 memory only operates at a frequency of 50 MHz.

The DDR controller is now available as an additional package for the GRLIB library and could be easily integrated into a design, independent of the GRLIB version. Having a working DDR2 memory controller, we implemented a memory tile for the invasive architecture. This enables the provision of a big DDR2-based global memory, which is accessible through the network from all other tiles.

L2 Cache The Level-2 cache controller has been integrated into a 4-core Leon design and successfully tested. In this context we also had to update our designs to use the commercial Gaisler Leon library, where the L2 cache is a part of.

DVI Interface The CHIPit system can be extended by a DVI video extension board. Using this board, a DVI digital video input stream can be converted from TMDS (transition minimised differential signalling) to source-synchronous digital data in a DVI receiver (Rx, IC SiI-7171) and raw pixels are obtained for further processing in the FPGAs of the CHIPit system. The processed pixels are converted back to TMDS using a transmitter (Tx, IC SiI-7170). Both the Rx and Tx ICs are shown in Figure 4.41 and they can be configured using the *Client Application Programming Interface soft-IP Modules* (CAPIM) provided by the CHIPit infrastructure. The CAPIM modules are connected to the UMR-bus for programming through the host PC. During the last year, we have explored and gained the know-how of the DVI extension board. The extension board is integral part of a first invasive TCPA demonstration, which has been developed in collaboration with Project B2 and is described in the next section.

Partitioning of Designs For larger designs, like our common demonstrator platform, which will be introduced later, it is necessary to map the design onto several FPGAs. The partitioning of HDL sources onto several FPGAs of the CHIPit requires a special tool-flow, invoking steps and tools that are usually not required for one-FPGA designs. Although, an automated partitioning algorithm exists for mapping parts of a design on different FPGAs, it is unlikely that is gives the best results concerning the final performance, or that is able to do a successful partitioning at all. We studied the tool-flow and the hardware platform in detail and developed a script based approach to simplify the partitioning of an InvasIC tiled mesh. Almost all necessary steps are now automatically invoked and only a few user interactions are required, which makes the partitioning feasible for people without detailed knowledge about the system and tools.

Invasive Demonstrator Architecture and Integration

Another major part of Project Z2's work was devoted to the evolution of the common demonstration platform (3×3 heterogeneous tiled architecture [BFHK+12]) and first proof-of-concept demonstrations on the CHIPit system. Major achievements were the successful integration of a TCPA to perform video processing with DVI interface and the successful steps towards planning and implementing the tiled invasive compute architecture.

Invasive TCPA Video Demo Together with Project B2, we have implemented a first invasive TCPA prototype, which was demonstrated at the Invasive Computing annual meeting in October 2012 as well as at the demo night of DASIP 2012 [MBHK+12].

In general, a TCPA as shown in Figure 4.41 is a highly parameterisable coarse-grained processor architecture with an $N \times M$ array of weakly programmable processor unit (PU). TCPAs are well suited to serve as accelerators for numerous of computationally intensive tasks. In the research of Project B2, each PU in a TCPA was augmented with an invasion controller (iCtrl in Figure 4.41), which can invade neighbouring processors in a linear, meander walk, or rectangular fashion. The number of invaded processors can be dynamically increased or reduced depending on the requirements of the application. For further details, we refer to the section of Project B2 in this report.

As a first prototype architecture, an array of size 5×5 has been designed (see Figure 4.41). In this demonstrator prototype, depending

on the requirements of an application as well as the available amount of resources, the number of invaded PUs can be increased and reduced on-the-fly at run-time, which results in different quality of the output video stream while the frame rate remains the same. In order to visually demonstrate the trade-off between the number of invaded processors and the image quality, an entire processing chain has been set up, including a DVI source, which is fed into the prototype in the CHIPit system, and the DVI output from the prototype is connected to a display in order to visualise the benefits of invasive computing.

The targeted video applications on the invasive TCPA prototype are several real-time 1-D and 2-D image filters (e. g., FIR filtering, 2-D convolutions, edge and feature detection) on a streaming input video. Such image filters have many applications in imaging and video processing. The range of applications vary from very precise medical imaging systems to low precision industrial imaging and consumer video applications. Depending on the number of invaded PUs, the convolution window size can be varied, which results in a different quality of the out-

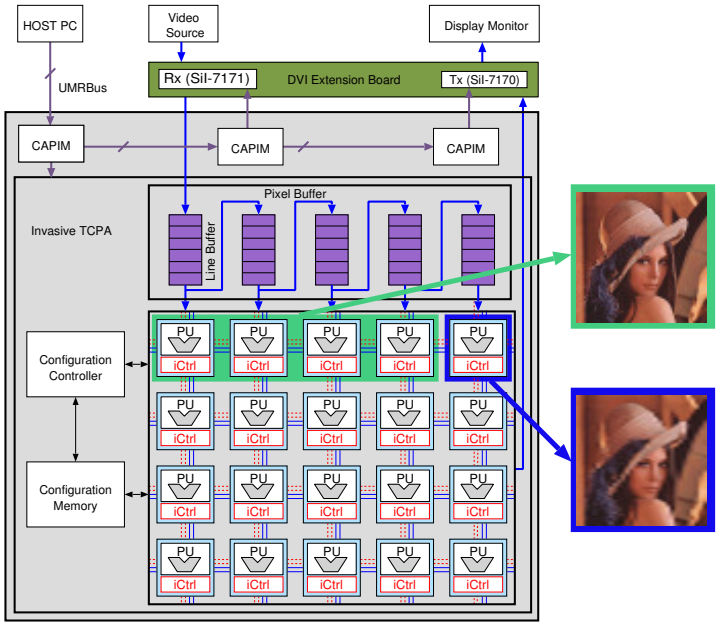


Figure 4.41: *Prototype Architecture*: Invasive TCPA performing 2-D image filtering on an input video stream

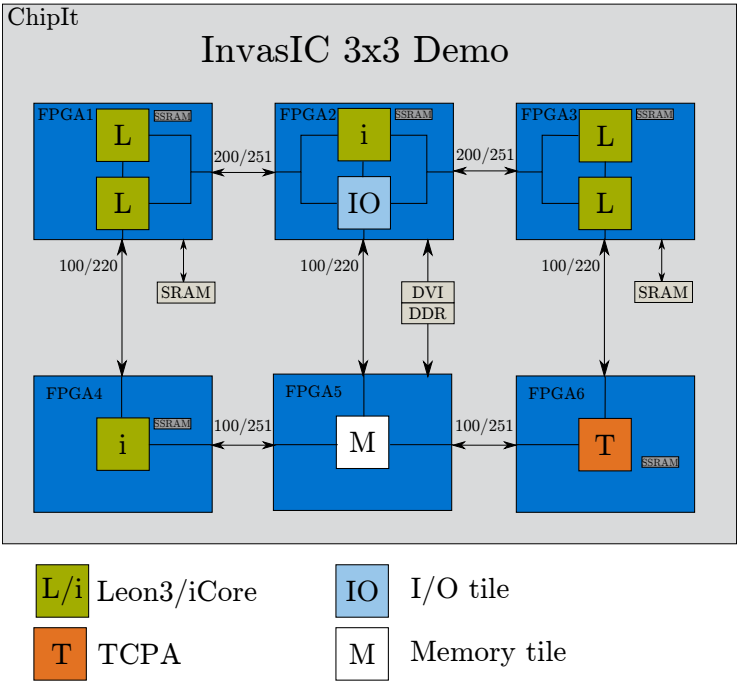


Figure 4.42: Overall architecture of the InvasIC Demonstrator

put images. As shown in Figure 4.41, the image in the green coloured box is filtered with four processors has better quality than the picture in the blue coloured box, which is filtered with only one processor.

Common Demonstrator for Invasive Computing In order to demonstrate the concepts and benefits of invasive computing in hardware, a common demonstrator architecture had to be defined. The configuration of this architecture, on the one hand had to fulfil the needs of the single sub-projects as well as the given constraints of the demonstrator platform, with respect to resource utilisation and interconnections. In consultation with the other sub-projects, a common demonstrator architecture has been defined. The proposed demonstrator architecture can be seen in Figure 4.42. The blue boxes in the figure represent the 6 FPGAs available on the CHIPit prototyping platform. Partitioned onto the FPGAs is a 3×3 tiled architecture, consisting of 4 regular Leon tiles, 2 *i*-Core tiles, one TCPA tile, one I/O tile and one memory tile. The memory tile is connected to a 256 MB DDR2 memory on an extension

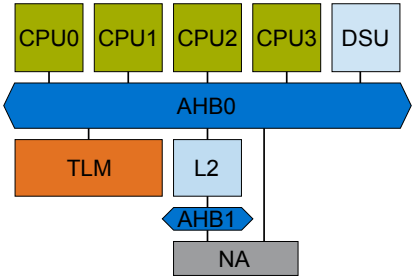


Figure 4.43: Block diagram of a Leon compute tile

board. The I/O tile will be connected to the DVI extension board. The Leon and *i*-Core tiles consist of 4 CPU cores, L2 cache, local memory, a debug support unit and the network adapter. A block diagram of a Leon tile is shown in Figure 4.43. Note that auxiliary blocks, like timers and interrupt controllers, are not shown here. Each CPU core has 8 KB instruction cache and 32 KB data cache. The level 2 cache will be 128 KB. The tile-local memory (TLM) is connected to SSRAM extension boards on the CHIPit system offering 8 MB per tile. The debug-support unit (DSU) will be used to debug the tile using the transactor presented afore.

The placement and the configuration of the individual tiles on the FPGAs is well conceived, as the possibilities are limited due to the constraints of the prototyping platform and the FPGAs themselves. The two main limiting resources are the amount of interconnections between the FPGAs and the amount of memory on the FPGAs. The structure shown in Figure 4.42, basically represents the largest architecture possible on the CHIPit platform, without making major compromises in the performance of the interconnections and the memory architecture.

The 3×3 demonstrator architecture is still a work in progress. However, a 2×2 mesh on 4 FPGAs has already been successfully tested. The resource utilisation of one Leon tile on an FPGA is roughly 40 % of Block-RAM and 25 % of Look-up tables. We expect that the tiles will be able to run at 50 MHz on the CHIPit platform, while the interconnections will be able to run at 100 MHz.

Demonstrator Coordination

The third major work of Project Z2 in close cooperation with all other projects is the definition of ***common demonstration scenarios*** that will be shown on the CHIPit platform in order to demonstrate the benefits

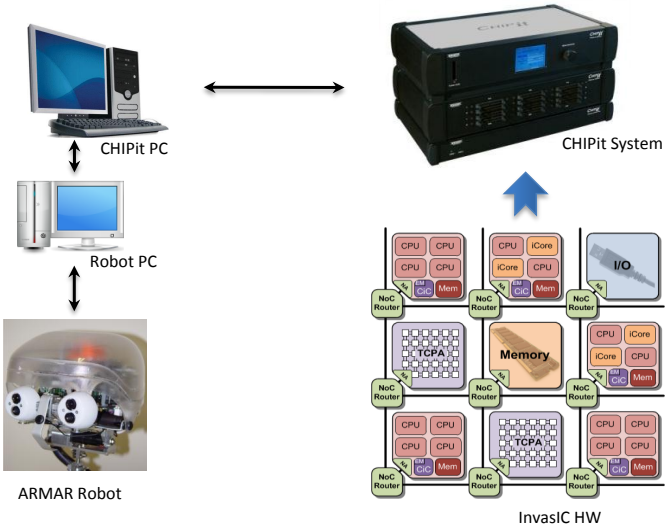


Figure 4.44: Robot demo scenario configuration

of invasive computing. These scenarios are driven by applications from Project D1 and Project D3.

Project D1 focus is on implementing a **cognitive robot control architecture** with its different processing hierarchies, on invasive TCPA, *i*-Core and RISC-based MPSoC. Various invasive applications that execute on the CHIPit system are shown in the flow diagram in Figure 4.45. The overall aim is to demonstrate object grasping using the humanoid robot named ARMAR. The goal is to explore techniques of self-organisation 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 utilisation of resources compared to traditional resource allocation at compile-time.

Z2

The Robot PC is responsible for robot control and for capturing the images from the cameras mounted on the robot head. The images from the robot head will be transferred to the CHIPit PC over the Ethernet interface (standard TCP/IP channel) as shown in Figure 4.44. An application running on the CHIPit PC will receive the images and convert them to VGA resolution (colour/greyscale or both, as required). These preprocessed images will be loaded to the DDR memory on

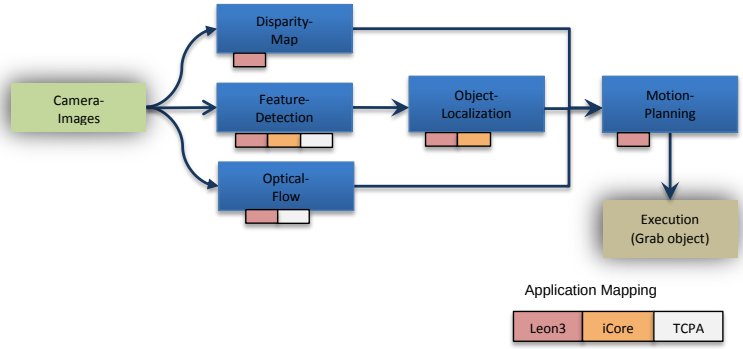


Figure 4.45: Robotic grasping application scenario

the CHIPit system using the Gaisler Ethernet Interface (through burst transfers). Either predefined locations or the address specified by the invasive application will be used. Short messages can be used to inform the application program (invasive applications running on CHIPit) about the availability of new frames.

The applications can execute on different types of processing elements (PEs) as shown in Figure 4.45. The coloured boxes represent various mapping schemes (TCPA/*i*-Core/Leon3). A particular type of PE is selected at run-time based on its availability, available power budget, reliability aspects etc. At first, the object to be grasped is located (SIFT based algorithm) and the distance to the object is calculated (disparity map). Simultaneously, the optical flow application can be used to detect any movements and avoid collision with other moving objects around the robot. In the next stage, the motion planning application is responsible for planning the movements of the robot's arm to grab the object. Once the motion planning is complete the results are transferred to the Robot PC (over the Ethernet interface), which will issue control commands to the servo motors on the robot.

Project D3 provides typical algorithms and algorithmic patterns from scientific computing and evaluates them on the common demonstrator platform. Amongst others, algorithms such as **numerical quadrature** and **matrix-exponentials** are considered. For more details and expectations how to show benefits of invasive computing, we refer to the corresponding pages of Project D3 in this annual report.

In order to ensure the interplay of all participating projects for demonstrator integration, the following actions have been performed.

- Definition of road maps, milestone plans, and interfaces to guarantee the timely availability of components and their consolidation.
- Project Z2 takes care of monitoring and updating of the aforementioned documents and reports the progress to the individual entities of the collaborative research centre.
- Organisation of several physical and virtual meetings as well as bi-lateral or multi-literal interaction with projects. Major physical meetings were the *Winter of Code* at TUM in January 2012, the *Demonstration Workshop* at FAU in May 2012, the *Autumn of Code* at KIT in September 2012 as well as sessions at the two Doctoral Researchers Retreats in 2012.

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.

[MBHK+12] S. Muddasani, S. Boppu, F. Hannig, B. Kuzmin, V. Lari, and J. Teich. "A Prototype of an Invasive Tightly-Coupled Processor Array". In: *Proceedings of the Conference on Design and Architectures for Signal and Image Processing (DASIP)*. Karlsruhe, Germany: IEEE, Oct. 23–25, 2012, pp. 393–394. ISBN: 978-1-4673-2089-4.

5 Working Groups

WG1: Working Group Architecture

Coordinators: Andreas Herkersdorf, Jörg Henkel

During 2011, the main objective of the working group was dedicated to the definition and fixation of a tiled invasive architecture platform, which allows a flexible composition of application-specific invasive architecture instances. This process implied a consolidation of requirements and perspectives between the architecture related B-projects, the Z2 demonstrator project and A-, C- as well as D-projects. In 2012, the architecture has been enhanced by a level 2 cache within the RISC-based compute tiles to improve performance for the demonstrators under discussion. The memory hierarchy has been finalised in terms of structure and size of the different types of memory. Please see subsection "Progress Summary of B Projects" for a concise description of the progress which has been achieved during 2012 in the individual architecture projects.

Tiled Invasive Architecture

As a reminder, one instance of a tiled invasive architecture 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 acting as accelerators for stream-based computations on a massively parallel array of low-complexity processing elements, ii) RISC compute tiles perform general-purpose computation on open source Leon3 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 are comprised of on-chip SRAM memory.

The way how and where the level 2 cache has been integrated in the

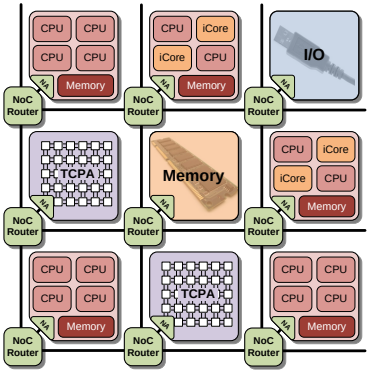


Figure 5.1: Tiled InvasIC Architecture

RISC-based compute tiles is depicted in Figure 5.2. The Gaisler level 2 cache has two AMBA AHB interfaces. Hence, locating the L2 cache between cores and the AHB0 bus wasn't an option. In consequence, the L2 cache doesn't cache content of the tile local memory (TLM), but data from tile-external global memory only. This is advantageous for the L2 cache efficiency as managed TLM accesses can be assumed to be as fast as L2 Cache accesses. The 256 KiB L2 cache is a "write-back" type cache which can be configured to either be direct mapped or support multi-way associativity of 2, 3 or 4 ways.

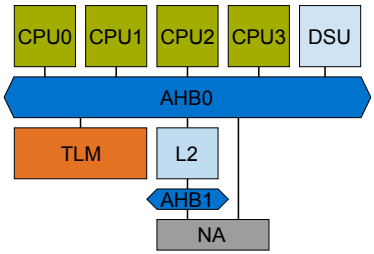


Figure 5.2: RISC compute tile with level 2 cache

A summary of the different memory types and their respective sizes within the memory hierarchy are depicted in Table 5.1 below.

WG1

2012 Architecture Projects Accomplishments

The following paragraphs summarise the main progress and accomplishments within the Invasic Architecture B-projects. A detailed description

Memory type	Size	Remarks
Scratchpad	8 KiB	Local variables, stack, frequently used
L1 cache	8 KiB I-cache 32 KiB D-cache	Coherency is guaranteed within a tile
L2 cache	128 KiB	Only global memory cached (no TLM)
Tile Local Memory (TLM)	8 MiB	Shared among all cores within a tile; remote access possible through message passing
On-Chip SSRAM	8 MiB	Partitioned on-chip memory accessible via ld/st over NoC
Off-Chip DDR2 SDRAM	256 MiB	Partitioned off-chip memory accessible via ld/st over NoC

Table 5.1: Memory hierarchy of a tiled invasive many-core

for each B project is found in Chapter 4, Research Projects.

Project B1 – Adaptive Application-Specific Invasive Architectures Project B1 implemented the first version of the reconfigurable fabric on a stand-alone Virtex-5 based platform. In particular this included the development of the dedicated interconnect between the fabric and the tile-local memory, the IP Core for the Reconfiguration Port, the integration of the *i*-Core into a multi-core system (one Leon and one *i*-Core on a shared bus, which can be viewed as a minimal tile configuration), and a demonstrator of the *i*-Core capabilities using a video encoder demo, which was presented to the InvasIC industrial board. The adaptive pipeline has been implemented onto a Leon core. Investigation into cache-adaptivity has been started using a stand-alone cache implementation. In collaboration with the other architecture projects a partitioning of the CHIPit demonstrator has been defined, with *i*-Core tiles placed onto their own FPGAs due to their higher memory requirements. Additionally, this demo was shown at the demo night at ReConFig conference 2012. A presentation on the invasive hardware architecture and its concept was given.

Project B2 – Invasive Tightly-Coupled Processor Array (TCPA) Architectures As part of the overall heterogeneous tiled invasive computing architecture, TCPAs are well suited to serve as accelerators for a myriad of computationally intensive tasks. In the first two years of research, Project B2 has proved that an invasion of an array of processing elements can be accomplished at an overhead of only a few clock cycles per pro-

WG1

cessor by exploiting regularity and by developing necessary hardware support (so-called *invasion controllers*). On this architectural foundation, two major highlights in TCPA research, which demonstrate the benefits of invasive computing, can be reported for 2012: The first one is the ability to directly exploit processor invasion for power management in TCPAs. Several hierarchical power gating techniques with different power domain granularity have been developed and evaluated with respect to latency and static power savings. As a second highlight, an invasive TCPA architecture that processes real-time video data from a DVI source and outputs the results as well by DVI, has been designed and prototyped on the CHIPit system. This was the first time that the trade-off between the number of invaded processors and image quality could be visually demonstrated. The demo was given at the Invasive Computing annual meeting in October 2012 as well as at the demo night of DASIP 2012.

Project B3 – Invasive Loosely-Coupled MPSoCs In Project B3, a) conceptual and architectural enhancements were performed to the CiC's functionality and b) a self-adaptive, hybrid DPM scheme for many-core architectures has been presented in the ViPG part. The CiC offloads i-let infections from the run-time system software (iRTSS) into the hardware by means of i-let FIFOs within its i-let mapper. Moreover, a scheme to classify applications based on their desired/expected monitoring conditions is proposed in order to achieve a better i-let to core mapping by the CiC using current sensor data. We presented a ViPG-based dynamic power management (DPM) scheme exploiting application's invade/retreat patterns to increase the energy efficiency of invasive applications. Dispensable cores are virtually power gated by local ViPG managers. Our cycle-accurate simulation infrastructure was extended to evaluate this DPM scheme. The Leon3 processor has been enhanced to support application-triggered clock gating. It was ported to an FPGA board with built-in power meters and presented to the industrial board.

Project B4 – Hardware Monitoring System and Design Optimisation for Invasive Architectures Project B4 investigates different hardware monitor types and systems. Temperature distributions of cores in different scenarios in a multicore system were evaluated. Different in-situ delay monitor types were analysed and their performance for use in adaptive voltage scaling was evaluated. Timing behaviour of circuits was investigated by evaluating statistical timing performance of circuits level-sensitive latches and by analysing challenges in timing analysis

considering process variations. A modelling framework for flipflop timing behaviour was introduced, that makes it possible to do static timing analyses at gate level.

Project B5 – Invasive NoCs An agent-based resource management strategy is part of the invasive concept. A given multicore architecture is divided into regions. Here, strategies for region-based communication resource management have been investigated in Project B5. One of the implemented strategies enables to adjust the communication infrastructure on a region basis according to the QoS requirements of the application using the region. Another strategy enables to define regional virtual networks which can be used for autonomous communication of single applications. A region based data collection mechanism was implemented to support the CiC and the agent system in collecting status resp. monitoring information.

The investigations of Project B5 showed that so-called guaranteed service (GS) connections, which are supported by the *i*NoC to enable QoS, can also be used to reduce the energy consumption compared to packet switching. Thus we are currently investigating two strategies to automatically increase the amount of GS traffic in the network, to reduce energy consumption. One strategy is located inside the NA. It sets up GS connections automatically if frequent communication is detected. Another strategy is implemented inside the routers. It replaces existing GS connections by new connections having higher communication demands.

In addition, Project B5 put a lot of effort in getting the first invasive *i*NoC architecture running on the CHIPit platform in collaboration with Project Z2. A 2x2 mesh could be synthesised and used successfully on a single FPGA.

In Summary

A 128 KiB write-back level 2 cache has been integrated into the Leon3 RISC compute tiles. Other than that, the invasive architecture platform remained unchanged and stable since its consolidation in 2011. Architecture projects participated and contributed to discussions with System Software and Language working groups on the subject of granularity of invasion / infection. Progress within the individual architecture B projects has been according to program internal milestone planning.

WG2: Working Group System Software

Coordinator: Wolfgang Schröder-Preikschat

The *AKSS* (abbr. Ger. "Arbeitskreis Systemsoftware") working group is involved in the entirety of all programs which control function and operation of invasive computing systems 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 arising 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:

2012-05-11 FAU Erlangen, host team Schröder-Preikschat

2012-09-17 FAU Erlangen, host team Schröder-Preikschat

Besides the definition of common terms, subjects of debate of the working group on the one hand comprised demonstrator-specific topics such as the memory bandwidth and the location of dynamic program memory (i. e., stacks and heaps) against the background of the CHIPit hardware and on the other hand fundamental questions regarding the granularity 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.

WG2

Common Terms Discussions within the collaborative research centre revealed that the generally usual notion of "application" has quite different meanings in the diverse technical disciplines. The range goes from

a single “thread” within a (non-sequential, multi-threaded) program looking into a very dedicated task to a (possibly complex formation of a) logically self-contained assembly of programs that jointly performs a certain computation or control function. By way of example, the former case relates to read-out of a sensor device and the latter case to some feedback control system consisting of many sensors, actuators, and (hardware/software) means for human-computer interaction. AKSS defined that, for InvasIC, “application” much more corresponds to the latter than the former.

For other common terms used in this report and relevant to InvasIC such as *i*-let (including its sub-classification into candidate, instance, incarnation and execution), claim, and team, refer to [TKL12, p. 93].

Granularity of Invasion and Infection The question of the adequate granularity—namely core or tile—of invasion, as significant at `invade`-time, *and* infection, as significant at `infect`-time, in terms of the hardware units affected by the respective measures was a central issue of several meetings over the year. These two actions while executing an invasive (parallel) program establish the moment of *allocation* of hardware units requested by an application (entity) and *dispatching* of *i*-lets to some processing element (i. e., core), respectively. A common understanding was to differentiate between these moments (“separation of concerns”) and, as a further consequence, to accept different granularity depending on the level of abstraction considered.

Granularity of invasion is interlinked with the *guarantees* the hard- and software system has to give to applications. This depends on (1) the resource-allocation constraints of the application specified by `invade`, (2) the scheduling criteria implemented by the *i*RTSS and (3) the assertiveness of the system-software/hardware stack to enforce the claimed constraints. Several artefacts of the system software and the hardware may be the cause of failure to comply with the constraints stated by an application. Besides temporal unavailability of a certain hardware unit (e. g., due to overheating or transient errors), typical cases of such artefacts come with coordinated sharing of system-level (hard-/software) resources such as cores, caches, busses or memory, and with the *interference* of otherwise unrelated application processes. Further anomalies may arise through the kind of (process) scheduling criteria that form the basis of design and implementation of (parts of) an operating system. Here, user-oriented criteria (e. g., response time, cycle time) are in opposition to system-oriented criteria (e. g., utilisation). The latter imply potential hazard to applications that assume a predictable

WG2

run-time behaviour of the underlying computing system; they are typical of general-purpose systems. The former are likely to let hardware resources rest in favour of deterministic operation, they are typical for a special-purpose system. Being in charge of juggling with both kinds of scheduling criteria at the same time (in practice within an operating system) means, however, to give priority to either of them. This breeds interference of the other, respectively. *As predictable run-time behaviour is an important aspect of InvasIC, in i*RTSS*, user-oriented criteria dominate system-oriented criteria. This is reflected by an API that demands the specification of (mandatory/optional) constraints from an application process in order to claim (i. e., `invade`) hardware resources.*

For *i*RTSS, the meaning of constraints is two-fold and distinguishes mandatory from optional specifications on the part of a particular application process. *Mandatory constraints* of invasion declare the resource demands of an imminent computation phase and provide an indication of the expected benefit of resource allocation, in functional and non-functional terms. *Optional constraints* qualify the willingness to share the allocated resources with competing processes of other (unrelated) applications, in spatial and temporal terms, and notify toleration of temporary under-/oversupply of spare cores for *i*-let dispatching. The former are for the *quantification* of application requirements, while the latter are for *immunisation* of (parts of) an application. *By default, resources are exclusively allocated to applications, but the exclusiveness may gradually be loosened by way of optional constraints.*

In the reporting year, the focus of resource allocation was on physical processing elements such as cores or tiles (of cores), respectively. But note that this focus also depends on the position taken within a multi-layer computing system such as InvasIC. At a lower (i. e., more hardware-oriented) level of abstraction, OctoPOS operates in a coarse-grained manner and allocates tiles to the agent system upon request. On a higher (i. e., more application-oriented) level, the agent system works in a fine-grained fashion and allocates the cores of one or more tiles to the (C/C++, X10) run-time system upon request. Such an approach of task sharing in resource management is very common in today's computing systems and has proved itself. Thus, core granularity of resource allocation is seen at application level even though tile granularity forms the basis on a lower level within the system.

WG2

An important influencing factor on the granularity of resource allocation is given with the (optional) constraint of application immunisation as mentioned above. Assume that an application wants to exclusively use a compute tile in order to avert interference by some other applica-

tion as far as possible. In such a situation, which reflects the default case, core allocation to the latter application always starts from a "virgin" tile even if the (last) tile that was allocated to the former has one or more cores to spare. That is to say, *i*RTSS tolerates *internal fragmentation* of a tile for the benefit of a more predictable run-time behaviour. As a consequence, this means a tile granularity of resource allocation, namely to assure immunisation of (parts of) an application. *Thus, tile granularity will be the (default, but overridable) praxis although core granularity is logically seen at application level.*

Granularity of infection largely depends on the nature and configuration of the claim of hardware that is going to be `infect`-ed by (a team of) *i*-lets in order to initiate a parallel computation. At that point in time, *i*RTSS (more specifically, OctoPOS) deploys *i*-let incarnations with the aid of the C*i*C. At the lowest (i. e., hardware) level of abstraction, the *i*-let dispatching according to the constraints of the team's claim always takes place at a core granularity. The C*i*C makes its (rule-based) dispatching decisions on the basis of the claim identification associated with the deployed *i*-lets. Only in case of an *i*-let tagged with a "wildcard" identifier (`null`) will the C*i*C select any core of the compute tile, adhering to system-oriented optimisation criteria (such as utilisation) for tile-wide load balancing at *i*-let arrival time. In case of a valid ("non-`null`") claim identifier, however, the C*i*C first and foremost adheres to user-oriented optimisation criteria (such as response or cycle time), and dispatches the *i*-lets to the cores of the tile as constrained by that very identifier. That is to say, if resource allocation—by means of `invade` and overriding the system default of exclusive use—resulted in the sharing of a single compute tile amongst (entities of) different applications, the C*i*C will send *i*-lets only to those cores that belong to the claim of the respective *i*-let. In that case, system-oriented optimisation criteria come after user-oriented ones, if at all.

This claim-based differentiation is made for better control of *interference* in case of multi-programmed compute tiles that are claimed (i. e., shared) by applications of different and possibly conflicting quality requirements in terms of non-functional properties (such as timing, jitter, energy or noise). In the process of setting out a claim (`invade`), the agent system of *i*RTSS establishes the appendant C*i*C dispatching rule that later on gets activated by OctoPOS in the process of *i*-let deployment (`infect`). When a team of *i*-lets is assorted for a specific claim—after return from a successful call to `invade`, but before the call to `infect` for that very claim—the association between *i*-let and claim identifier or wildcard, respectively, is established. During infection, OctoPOS then

WG2

tags all *i*-lets with the identifying information related to the claim of their team.

For the purpose of better system utilisation, the C*i*C will be capable of dispatching *i*-lets of an application to spare cores even of an exclusively taken compute tile that, however, was not entirely allocated to the application. The number of spare cores then corresponds to the portion of internal fragmentation (of such a tile) as result of application immunisation as explained above. Utilisation of these cores then leads to a temporary oversupply of computing resources to the application running on the respective compute tile. This also brings about interference and causes unpredictable run-time behaviour of an application. Just like oversupply, also a temporary undersupply of computing resources may occur. An example of this is an over-heated core that will be masked by the C*i*C and, thus, excluded from further *i*-let processing until its operating temperature has dropped below a certain threshold. Both over- and undersupply affect application processing in non-functional terms. By default, *i*RTSS will not instruct the C*i*C to oversupply an application with spare cores, but this presetting may be overridden by means of optional constraints specified by an application (at `invade`-time). The same goes for the undersupply of (computing) resources, which is also considered an optional constraint of invasion to give application-side toleration notice to *i*RTSS.

Outlook for 2013 The upcoming focus of work on systems software will be the connection between the X10 run-time system and *i*RTSS. Aspects such as the representation of blocking X10 activities on the basis of run-to-completion *i*-lets to be managed by OctoPOS, creating an assortment of teams of *i*-lets, and bottom-up signalling of exceptional events to X10 applications will be in the fore. In addition, rule-based interaction with the C*i*C and integration of TCPA-like compute tiles are on the agenda.

Publications

[TKL12] J. Teich, J. Kleinöder, and K. Lohmann, eds. *Invasive Computing*. Annual Report 2011. DFG Transregional Collaborative Research Centre 89, 2012.

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. To start with, the language must support parallel programming in distributed, heterogeneous memory architectures. The language must support fundamental invasive operations (invade infect, retreat), as well as a full-fledged spectrum of parameters and constraints for different types of invasion. To demonstrate the benefits of invasive computing, the language must support fully dynamic resource-aware programming, as well as recursive invasion. The language must finally also provide interfaces for dynamic resource parameters and system state.

After the fundamental decision in 2010, namely to base the language on IBM's X10 language, a programming framework called InvadeX10 was defined and evaluated in a sequence of revisions and extensions. The language is defined in Project A1, and its compiler is developed in Project C3. All language decisions were discussed with other projects (in particular D3, C1, C2). The language is exercised and validated on real algorithms and problems. By the end of 2012, the language runs on the simulator, and the compiler (which compiles InvadeX10 down to the CHIPit hardware demonstrator) is nearing completion. Several demonstrator programs are available.

InvadeX10 examples. For the demonstrator, it is planned to first present a pedagogical example which explains fully dynamic resource awareness as well as recursive invasion. We chose invasive Quicksort that decides dynamically whether it recursively invades an additional core to sort one of the two created partitions. The corresponding decision tree is depicted in Figure 5.3. The dynamic decision is made strictly locally and depends on the subproblem size and the expected invasion overhead, which takes into account whether the invaded core is tile-local or on another tile. Thus, the smaller the subproblem, the stronger are

the constraints on "invade" used in the invasive Quicksort shown in Figure 5.4. Often, the code can deduce that additional invades are not sensible (e. g., subproblem too small, or invasion overhead too high).

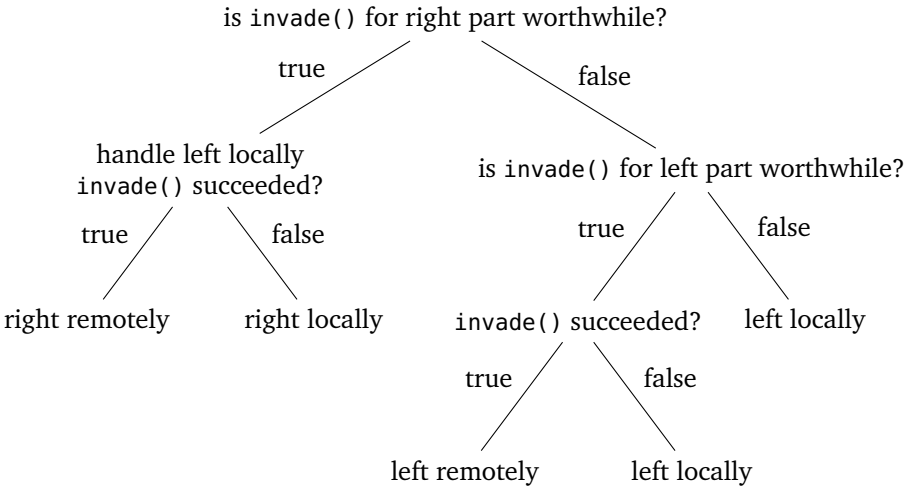


Figure 5.3: Decision tree for invasive Quicksort example in Figure 5.4. First, the algorithm checks whether the right partition is worth an invasion and, if so, invades an additional core. If the invade succeeds, the invaded core recursively sorts the right partition. Otherwise, the partition is sorted locally. If the invasion of an additional core for the left partition is not beneficial from the beginning, the algorithm checks the right partition instead.

Note that in this pedagogical example the partitioning is still sequential. More realistic examples, such as a multigrid solver based on recursive invasion, are described in Project D3.

Granularity of invasion and OS issues. From an operating system point of view, invasion is a complex and expensive operation, where generality and/or usability must be balanced against performance and implementation simplicity. Furthermore, future hardware will perhaps offer a very high number of tiles with a small number of cores on each tile. Therefore, some PIs argued that an invade should always acquire at least a complete tile, rather than individual cores on a tile. Other PIs argued that often one wants to exploit shared memory on a tile after an invasion. Other questions concerned the strictness of invade constraints. For example, after a successful invade of 3 PEs, may the resulting *i*-lets be scheduled to 2 or 4 PEs instead of 3? Arguments of practicality and simplicity vote for the latter, but dynamic resource-awareness may

WG3

require that constraints are 100% respected. Furthermore, often it is important to base dynamic resource-awareness strictly on locally available data, instead of manipulating global claims via "reinvades".

A meeting in early December solved these questions as follows:

- some applications perform their own load-balancing, based on functions which estimate the net gain of an additional invasion vs. invasion (overhead) costs. If such applications successfully obtain a claim according to specific constraints (e. g., n PEs on the current tile), they must rely that the constraints are 100% respected at "infect" time[1]. Neither oversupply nor undersupply is allowed, as both would destroy the load balancing estimations. Of course, other applications may be satisfied with less strict constraints, and allow that the n *i*-lets are (temporarily) scheduled to, say, $n-1$ or $n+1$ PEs.

- some applications need recursive invasion, where subtasks decide locally about benefits of additional invasions (see, e. g., the invasive Quicksort example). In such cases, it may again be necessary that all (recursive) constraints are 100% respected. Other applications do load balancing not locally, but more globally, and to this end exchange information between subtasks; such applications may use re-invades instead of recursive invades.

- some applications require that additional *i*-lets (e. g., created by recursive invades/infects) run on the same tile as other *i*-lets of the same application, in order that they can exploit shared memory on a tile. Hence, it may well be possible that several claims of an application are dispatched to the same tile.

- On the other hand, it is preferable to allocate PEs tile-wise, not core-wise. This may result in wasted cores, but simplifies the OS tasks. For future hardware, where a lot of tiles with few cores each are available, wasted cores may be acceptable – for the demonstrator, the situation is certainly different, and core-wise invasion will be common.

To provide hardware support by the C*i*C for the above issues, it was decided to extend the *i*-let information blocks by claim- and/or application-identification tags ("tagged C*i*C"). The AK language strongly supports this decision, even though it makes the C*i*C slightly more complex.

[1] If they cannot be respected any more later, e. g., due to hardware failure, an exception must be thrown.

WG3

Related projects. The core projects contributing to the working group are Project A1 and Project C3. Project A1 defines and validates the language resp. the framework. 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 may execute programs written in the InvadeX10 framework and approximate the timing behaviour depending on an architecture model. 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 invasic constructs such as invade, infect, and retreat. Applications in Project D3 will use the InvadeX10 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, e. g., memory model or invasion-specific instructions affect the design of language, compiler, and run-time system; but cannot be solved by the Projects A1, C1 and C3 alone. The compiled InvadeX10 programs will eventually run on the CHIPit demonstrator platform, thus there is an ongoing interaction and integration with Project Z2.

Outlook for 2013. It is expected that the language design will stabilise, that the compiler (non-optimising version) will be completed, and that several pedagogical examples as well as realistic applications will demonstrate the benefits of invasive programming. The latter requires extensive measurements and evaluations.

```
val p = partition(data, left, right);
val i = p.first;
val j = p.second;
val leftsize = j - left;
val rightsize = right - i;
finish {
  if (isInvasionWorthwhile(rightsize)) {
    if (leftsize > 0) async {
      leftresult = qsort(data, left, j); }
    val constraints = new AND();
    constraints.add(new PEQuantity(0, 1));
    if (!isOtherPlaceWorthwhile(rightsize)) {
      constraints.add(new ThisPlace());
      constraints.add(new ScalabilityHint([0, 90]);
    } else {
      constraints.add(new ScalabilityHint([0, 60]);
    }
    val claim = Claim.invade(constraints);
    if (claim.size() == 1) {
      val ilet = (id: IncarnationID)
        => qsort(data, i, right);
      rightresult = claim.infect(ilet)(0);
    } else { rightresult = qsort(data, i, right); }
    claim.retreat();
  } else {
    if (rightsize > 0) async {
      rightresult = qsort(data, i, right); }
    if (isInvasionWorthwhile(leftsize)) { /* see above */ }
  }
}
return combine(leftresult, rightresult);
```

Figure 5.4: An InvadeX10 Quicksort program exploiting dynamic resource awareness and recursive invasion. Note that this is a pedagogical example as it uses a sequential partitioning algorithm. The predicates `isInvasionWorthwhile()` and `isOtherPlaceWorthwhile()` are used to decide *locally* if a new invasion is worthwhile and which constraints will be used for the invasion, based on parameters such as the input size.

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 major events and activities in 2012. These events include Internal Meetings (Section 6), Trainings and Tutorials (Section 7) as well as further scientific activities (Section 8). Last but not least, we present the current constitution of the Industrial and Scientific Board and provide a short summary of the Board's meeting in October 2012 in Section 9.

Figure 5.5: At the annual meeting in Kloster Irsee, October 2012

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. In 2012, researchers met at the following opportunities:

Event	Date	
InvasIC "Winter of Code"	Jan. 20, 2012, Munich	At the InvasIC Winter of Code Workshop, researchers from all projects met to present the status of their work for validating and demonstrating Invasive Computing on the common CHIPit platform.
Semi-annual Meeting 2012	Feb. 13/14, 2012, Erlangen	At the semi-annual meeting the status quo of projects and the three working groups (architecture, languange and applications, system software) was summarised
Doctoral Researcher Retreat	Feb. 15–17, 2012, Obertrubach	This year the doctoral researchers gathered in Obertrubach to discuss further challenges on the way to the invasive SoC and to show up solutions.
Workshop Systemsoftware Group	May 11, 2012, Erlangen	Subjects of debate of the working group were memory bandwidth, dynamic program memory (i. e., location of stacks and heaps), and benchmarking of invasion (before/after scenarios).
Demonstrations Workshop	May 30, 2012, Erlangen	Project-specific demonstrations and a first common demonstration scenario on the FPGA-based prototyping platform were refined.
Workshop Systemsoftware Group	Sep. 17, 2012, Erlangen	Subjects of debate of the working group were the definition of common terms (here: application), granularity of invasion, and benchmarking of invasion (system software basis for before/after scenarios).
InvasIC "Autumn of Code"	Sep. 25, 2012, Karlsruhe	In the Autumn of Code Meeting the current state of the works for the invasive compute platform was discussed along with the definition of next steps targeting the integration activities of the common demonstrator scenarios.
Doctoral Researcher Retreat	Oct. 8–10, 2012, Kloster Irsee	The 3rd InvasIC DRR took take place in connection with the Annual Meeting 2012 in Kloster Irsee, Germany.

Annual Meeting 2012	Oct. 11/12, 2012, Kloster Irsee	Projects and working groups presented their progress with focus on the next working packages and first project ideas for the second funding phase. On the meeting's second day members of the "Industrial and Scientific Board" attended to evaluate the ideas and progress of the presented projects.
Workshop "Nicht nur für Simulanten"	Dec. 5, 2012, Erlangen	Project C2 organised an one-day workshop on the usage and the application areas of the functional simulator
Expertentreffen "Granulatität der Invasion"	Dec. 7, 2012, Karlsruhe	The Expertentreffen was convened to clarify core- vs. tile-granularity.
Workshop Architecture Group	Dec. 17, 2012, Munich	The main topic of the meeting was a continuation of the discussion on the granularity of invade / infect operations with the emphasis on aspects affecting the invasive-specific hardware architecture.

Figure 6.1: Demonstrations Workshop May 2012 in Erlangen

7 Trainings and Tutorials

Workshops and trainings were organised 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 invasive computing related topics.

Event	Date	
Leadership Qualities	Mar. 29, 2012 Erlangen	The trainers Marion Bredebusch and Martin Conrath conducted a basic training on leadership qualities for the scientific staff and professors of InvasIC.
HiPEAC Summer School ACACES	Jul. 8–14, 2012 Fiuggi, Italy	Prof. Dr.-Ing. J. Teich (FAU) gave a one week lecture on the subject of "Domain-specific and resource-aware computing on multi-core architectures"

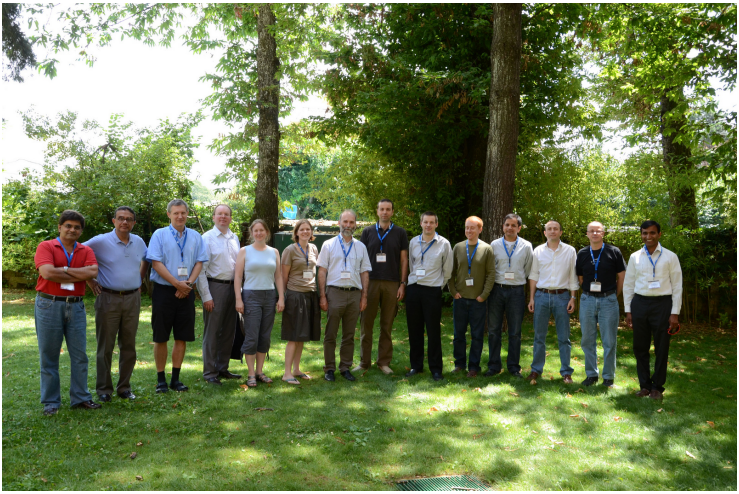


Figure 7.1: Lecturers of the HiPEAC European Network of Excellence Summer School ACACES 2012, Fiuggi, Italy

8 Industrial and Scientific Board

For the promotion of our ideas to the industrial community and for the discussion with peer colleagues world-wide, we established the InvasIC Industrial and Scientific Board. Members of the board in its current constitution are 7 experts from four institutions industry and university:

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)

Siemens

Urs Gleim (Head of Research Group Parallel Systems Germany, Siemens Corporate Technology)

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)

The members of the InvasIC Industrial and Scientific Board are periodically informed about progress and news of the TRR InvasIC. In October 2012, the members of the Board met during the Annual Meeting in Kloster Irsee. In talks and demonstrations given by the different projects as well as during a poster session, the Board's members could get an idea about the current state of research in InvasIC. In a concluding plenary session, the opinions and suggestions of the Board's members were collected and discussed.

Figure 8.1: Members of the Industrial and Scientific Board during the plenary session at the Annual Meeting in Kloster Irsee, October 2012. From left to right: Urs Gleim (Siemens), Elmar Maas (Intel), Michael O'Boyle (University of Edinburgh), Klaus-Dieter Schubert (IBM), Peter-Hans Roth (IBM), Hans-Christian Hoppe (Intel)

9 InvasIC Activities

To promote the ideas and results of InvasIC and discuss them with leading experts from industry and academia, guest speakers were invited to the "InvasIC Seminar". Moreover, PIs of InvasIC gave talks at important research sites ("Invited Talks") 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 to the other sites. For further information, we refer to our website `http://www.invasic.de`.

Figure 9.1: Dr. Alain Darte giving a talk at the InvasIC Seminar

Figure 9.2: Prof. Norbert Wehn together with Prof. Jürgen Teich

InvasIC Seminar

Time and Place	Title	Speaker
Erlangen, Feb. 10, 2012	Kernfragen: Multicore-Prozessoren in der Industrie	U. Gleim (Siemens Corp. Techn.)
Erlangen, Feb. 17, 2012	Efficiency Metrics and Bandwidth - A Memory Perspective	Prof. Dr.-Ing. N. Wehn (TU Kaiserslautern)
Erlangen, Apr. 4, 2012	New and old Features in MPI-3.0: The Past, the Standard, and the Future	Dr. T. Hoefler (University of Illinois)
Erlangen, Apr. 20, 2012	Expanding the Envelope – European Intel Research in Visual Computing, Exascale and Parallelism	H.-C. Hoppe (Intel ExaCluster Lab)
Erlangen, May 18, 2012	Kernel Offloading for FPGA with Optimized Remote Accesses	Dr. A. Darte (CNRS)
Erlangen, May 21, 2012	HOPES: A Model-Based Design Framework of Parallel Embedded Systems	Prof. Dr. Soonhoi Ha (Seoul National University)
Erlangen, May 24, 2012	Robust System Design	Prof. S. Mitra (Stanford University)
Erlangen, Jun. 22, 2012	Programmability and Performance Portability for Heterogeneous Many-Core Systems	Prof. Dr. S. Benkner (University of Vienna)
Erlangen, Jun. 29, 2012	Software System Engineering: Was fehlt noch?	Prof. Dr. David L. Parnas McMaster University, Hamilton Canada
Erlangen, Jun. 29, 2012	Application-driven Embedded System Design	Prof. Dr. A. Fröhlich (UFSC / LISHA)
Erlangen, Jul. 6, 2012	When does it get hot	Prof. Dr. L. Thiele (ETH Zurich)
Erlangen, Jul. 25, 2012	Organic Computing – Quo vadis?	Prof. Dr.-Ing. C. Müller-Schloer (LU Hannover)
Erlangen, Sep. 14, 2012	Embedded Multicore Design Technologies: The Next Generation	Prof. Dr. R. Leupers (RWTH Aachen)
Erlangen, Oct. 19, 2012	Large Scale Multiprocessing: From SoCs to Supercomputers	S. Sarkar (Intel Exascience labs)
Erlangen, Nov. 20, 2012	Building Future Embedded Systems – Internet of Things and Beyond	Prof. Z. Salcic (University of Auckland)
Munich, Nov. 27, 2012	Massively parallel simulations on Octree based meshes	J. Zudrop (GRS-Sim, Aachen)
Munich, Nov. 27, 2012	Efficiency and scalability on SuperMUC: Lattice Boltzmann Methods on complex geometries	S. Zimny (GRS-Sim, Aachen)

Munich, Nov. 28, 2012	Modelling of microscopic flows using particle-based methods	S. Strobl (FAU)
Erlangen, Nov. 30, 2012	Design and Analysis for Timing Critical Systems	Prof. Dr. Jian-Jia Chen (KIT)

Figure 9.3: Prof. Teich spent four weeks as a visiting Professor at the University of Auckland, New Zealand

Invited Talks

Date and Place	Title	Speaker
May 18, 2012, Hong Kong	i-Core: Adaptive Computing for Multi-core Architectures	Prof. Dr. J. Henkel (KIT)
Jun. 12, 2012, CSAIL - MIT	Proactive Energy-Aware Programming	T. Hönig (FAU)
Aug. 9, 2012, University of Auckland	Invasive Computing - or - How to Tame 1000+ Cores on a Chip?	Prof. Dr.-Ing. J. Teich (FAU)
Sep. 20, 2012, Intel Braunschweig	Invasive Computing - or - How to Tame 1000+ Cores on a Chip?	Prof. Dr.-Ing. J. Teich (FAU)
Sep. 20, 2012, Intel Braunschweig	Automatic Code Generation for Image Processing Algorithms on Accelerators in Heterogeneous Architectures	Richard Membarth (FAU)
Sep. 28, 2012, Paris (UPMC, LIP6)	Invasive Computing: A Systems-Programming Perspective	Prof. Dr.-Ing. W. Schröder-Preikschat (FAU)
Oct. 26, 2012, IBM Böblingen	Invasive Computing - or - How to Tame 1000+ Cores on a Chip	Prof. Dr.-Ing. J. Teich (FAU)
Nov. 9, 2012, University of California, Riverside	Guest Lecture: Why do we see more and more domain-specific accelerators in multi-processor systems?	Dr.-Ing. F. Hannig (FAU)

Workshops and Conferences

Date and Place	Title	Speaker
Feb. 1, 2012, Sydney, Australia (Special Session 3: Design and Prototyping of Invasive MPSoC Architectures, ASP-DAC 2012)	Talk: Approximate Time Functional Simulation of Resource-Aware Programming Concepts for Heterogeneous MPSoCs	Prof. Dr.-Ing. J. Teich (FAU)
Feb. 1, 2012, Sydney, Australia (Special Session 3: Design and Prototyping of Invasive MPSoC Architectures, ASP-DAC 2012)	Talk: Invasive Manycore Architectures	Prof. Dr. J. Henkel (KIT)
Feb. 1, 2012, Sydney, Australia (Special Session 3: Design and Prototyping of Invasive MPSoC Architectures, ASP-DAC 2012)	Talk: Hardware Prototyping of Novel Invasive Multicore Architectures	Prof. Dr.-Ing. J. Becker (KIT)
Feb. 1, 2012, Sydney, Australia (Special Session 3: Design and Prototyping of Invasive MPSoC Architectures, ASP-DAC 2012)	Talk: Invasive Computing for Robotic Vision	Prof. Dr.-Ing. W. Stechele (TUM)
Feb. 29, 2012, Munich (3rd Workshop on Parallel Programming and Run-Time Management Techniques for Manycore Architectures, ARCS 2012)	Introduction to Invasive Computing and Overhead Analysis for a Shared-memory MPSoC	Prof. Dr.-Ing. J. Teich (FAU)
Mar. 16, 2012, Dresden (Workshop Quo Vadis, Virtual Platforms? Challenges and Solutions for Today and Tomorrow, Date 2012)	Actor-Based Virtual Prototype Generation	Prof. Dr.-Ing. J. Teich (FAU)
May 20–25, 2012, Schloss Dagstuhl (Seminar 12212)	Talk: Resource Aware Programming	Prof. Dr. M. Gerndt (TUM)
Sep. 18, 2012, Wien, Austria (Special Session Invasive Programming of Heterogeneous Multi-core Systems, FDL 2012)	Talk: An Integrated Simulation Framework for Invasive Computing	Marcel Meyer (TUM)
Sep. 18, 2012, Wien, Austria (Special Session Invasive Programming of Heterogeneous Multi-core Systems, FDL 2012)	Talk: Invasive Computing - Concepts and Overheads	Prof. Dr.-Ing. J. Teich (FAU)
Sep. 18, 2012, Wien, Austria (Special Session Invasive Programming of Heterogeneous Multi-core Systems, FDL 2012)	Talk: Invasive Computing with iOMP	Prof. Dr. M. Gerndt (TUM)

Oct. 15, 2012, Ghent, Belgium ("Models and Assistive Tools for Programming Emerging Architectures", HiPEAC CSW 2012)	Talk: Invasive Computing - or - How to Tame 1000+ Cores on a Chip	Prof. Dr.-Ing. J. Teich (FAU)
Nov. 8, 2012, San Jose, CA, USA (ICCAD 2012)	1st International Workshop on Domain-Specific Multicore Computing (DSMC)	Prof. J. Teich (FAU) Prof. V. Narayanan (PSU)
Nov. 8, 2012, San Jose, CA, USA (DSMC at ICCAD 2012)	Talk: Invasive Tightly-Coupled Processor Arrays	Dr.-Ing. F. Hannig (FAU)

Figure 9.4: Participants of DSMC during ICCAD 2012 in San Jose, USA

10 Publications

[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.

[BBS12] M. Bader, H.-J. Bungartz, and M. Schreiber. “Invasive Computing on High Performance Shared Memory Systems”. In: *Facing the Multicore-Challenge III*. 2012.

[BGSH12] L. Bauer, A. Grudnitsky, M. Shafique, and J. Henkel. “PATS: a Performance Aware Task Scheduler for Runtime Reconfigurable Processors”. In: *20th Annual International IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM)*. Toronto, Canada, May 2012.

[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.

[BBMZ12] M. Braun, S. Buchwald, M. Mohr, and A. Zwinkau. *An X10 Compiler for Invasive Architectures*. Tech. rep. 9. Karlsruhe Institute of Technology, 2012. URL: `http://digbib.ubka.uni-karlsruhe.de/volltexte/1000028112`.

[CLS12] N. Chen, B. Li, and U. Schlichtmann. “Iterative timing analysis based on nonlinear and interdependent flipflop modelling”. In: *Circuits, Devices Systems, IET* 6.5 (Sept. 2012), pp. 330–337. ISSN: 1751-858X. DOI: `10.1049/iet-cds.2011.0347`.

[GHHH+12] M. Gerndt, F. Hannig, A. Herkersdorf, A. Hollmann, M. Meyer, S. Roloff, J. Weidendorfer, T. Wild, and A. Zaib. “An Integrated Simulation Framework for Invasive Computing”. In: *Forum on Specification & Design Languages (FDL)*. Vienna, Austria, Sept. 18–20, 2012, pp. 185–192. ISBN: 978-2-9530504-5-5.

[GHMS+12] M. Gerndt, A. Hollmann, M. Meyer, M. Schreiber, and J. Weidendorfer. “Invasive computing with iOMP”. In: *FDL*. 2012, pp. 225–231.

[GS13] E. Glocker and D. Schmitt-Landsiedel. “Modeling of Temperature Scenarios in a Multicore Processor System”. 2013.

[Han12] F. Hannig. “Invasive Tightly-Coupled Processor Arrays”. Talk, 1st International Workshop on Domain-Specific Multicore Computing (DSMC) at International Conference on Computer-Aided Design (ICCAD), San Jose, CA, USA. Nov. 8, 2012.

[HKB12] J. Heisswolf, R. König, and J. Becker. “A Scalable NoC Router Design Providing QoS Support Using Weighted Round Robin Scheduling”. In: *Parallel and Distributed Processing with Applications (ISPA), 2012 IEEE 10th International Symposium on*. July 2012, pp. 625 –632. DOI: 10.1109/ISPA.2012.93.

[HZWK+12] J. Heisswolf, A. Zaib, A. Weichslgartner, R. König, T. Wild, J. Teich, A. Herkersdorf, and J. Becker. “Hardware-assisted Decentralized Resource Management for Networks on Chip with QoS”. In: *Proceedings of the 19th Reconfigurable Architectures Workshop (RAW 2012)*. Shanghai, China, May 2012, pp. 1 –8.

[Hen12] J. Henkel. “i-Core: Adaptive Computing for Multi-core Architectures”. Embedded System Design from MultiMedia to Cloud, Hong Kong, Invited Talk. May 18, 2012.

[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. ISBN: 978-1-4673-0770-3. DOI: 10.1109/ASPDAC.2012.6164944.

[HDMS+12] W. Hofer, D. Danner, R. Müller, F. Scheler, W. Schröder-Preikschat, and D. Lohmann. “Sloth on Time: Efficient Hardware-Based Scheduling for Time-Triggered RTOS”. In: *Proceedings of the 33rd IEEE International Symposium on Real-Time Systems (RTSS ’12)*. (To appear).

IEEE Computer Society Press, Dec. 2012. URL: http://www4.cs.fau.de/Publications/2012/hofer_12_rtss.pdf.

[HG12] A. Hollmann and M. Gerndt. “Invasive Computing: An Application Assisted Resource Management Approach”. In: *MSEPT*. 2012, pp. 82–85.

[HEKSP12] T. Hönig, C. Eibel, R. Kapitza, and W. Schröder-Preikschat. “SEEP: exploiting symbolic execution for energy-aware programming”. In: *ACM SIGOPS Operating Systems Review* 45.3 (Jan. 2012), pp. 58–62. ISSN: 0163-5980. DOI: 10.1145/2094091.2094106.

[HKSP12] T. Hönig, R. Kapitza, and W. Schröder-Preikschat. *ProSEEP: A Proactive Approach to Energy-Aware Programming*. Poster. June 13–15, 2012, Boston, MA, USA, 2012.

[HGTH12] M. Hübner, D. Göhringer, C. Tradowsky, and J. Henkel. “Adaptive Processor Architecture”. In: *12th International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS XII)*. July 2012.

[JH13] J. Jahn and J. Henkel. “Pipelets: Self-Organizing Software Pipelines for Many Core Architectures”. In: *Design Automation and Test in Europe Conference (DATE) (to appear)*. Grenoble, France, Mar. 2013.

[JKPC+12] J. Jahn, S. Kobbe, S. Pagani, J.-J. Chen, and J. Henkel. “Work in Progress: Malleable Software Pipelines for Efficient Many-core System Utilization”. English. In: *Proceedings of the 6th Many-core Applications Research Community (MARC) Symposium*. Ed. by E. Noulard and S. Vernhes. Toulouse, France: ONERA, The French Aerospace Lab, July 2012, pp. 30–33. URL: http://hal.archives-ouvertes.fr/hal-00719027.

[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.

[LMBH+12a] V. Lari, S. Muddasani, S. Boppu, F. Hannig, and J. Teich. “Design of Low Power On-Chip Processor Arrays”. In: *Proceedings of the 23rd IEEE International Conference on Application-specific Systems, Architectures, and Processors (ASAP)*. Delft, The Netherlands: IEEE Computer Society, July 9–11, 2012, pp. 165–168. ISBN: 978-0-7695-4768-8. DOI: 10.1109/ASAP.2012.10.

[LMBH+12b] V. Lari, S. Muddasani, S. Boppu, F. Hannig, M. Schmid, and J. Teich. “Hierarchical Power Management for Adaptive Tightly-Coupled Processor Arrays”. In: *ACM Transactions on Design Automation of Electronic Systems (TODAES), accepted for publication* 18.1 (Dec. 2012).

[LCS12] B. Li, N. Chen, and U. Schlichtmann. “Statistical Timing Analysis for Latch-Controlled Circuits with Reduced Iterations and Graph Transformations”. In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*. Nov. 2012, pp. 1670–1683.

[LCXS13] B. Li, N. Chen, Y. Xu, and U. Schlichtmann. “On Timing Model Extraction and Hierachical Statistical Timing Analysis”. to appear in: IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems. 2013.

[LSHSP12] D. Lohmann, O. Spinczyk, W. Hofer, and W. Schröder-Preikschat. “The Aspect-Aware Design and Implementation of the CiAO Operating-System Family”. In: *Transactions on AOSD IX*. Ed. by G. T. Leavens and S. Chiba. Lecture Notes in Computer Science 7271. Springer-Verlag, 2012, pp. 168–215. DOI: 10.1007/978-3-642-35551-6_5.

[MFGSP12] T. R. Mück, A. A. M. Fröhlich, M. Gernoth, and W. Schröder-Preikschat. “Implementing OS Components in Hardware using AOP”. In: *ACM SIGOPS Operating Systems Review* 46.1 (Jan. 2012). Best Papers from 2011 Brazilian Symposium on Computing Systems Engineering (SBESC), pp. 64–72.

[MBHK+12] S. Muddasani, S. Boppu, F. Hannig, B. Kuzmin, V. Lari, and J. Teich. “A Prototype of an Invasive Tightly-Coupled Processor Array”. In: *Proceedings of the Conference on Design and Architectures for Signal and Image Processing (DASIP)*. Karlsruhe, Germany: IEEE, Oct. 23–25, 2012, pp. 393–394. ISBN: 978-1-4673-2089-4.

[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.

[PHWAA+13] C. Pham, J. Heisswolf, S. Wenner, Z. Al-Ars, J. Becker, and K. Bertels. “Hybrid Interconnect Design for Heterogeneous Hardware Accelerators”. In: *Proc. Design, Automation & Test in Europe Conference & Exhibition, to appear*. Grenoble, France, Mar. 2013.

[PHSG+12] N. Pour Aryan, L. Heiss, D. Schmitt-Landsiedel, G. Georgakos, and M. Wirnshofer. “Comparison of in-situ delay monitors for use in Adaptive Voltage Scaling”. In: *Advances in Radio Science (ARS)*. Vol. 10. 2012, pp. 205–208.

[RHT12a] 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. ISBN: 978-1-4673-0770-3. DOI: 10.1109/ASPDAC.2012.6164943.

[RHT12b] S. Roloff, F. Hannig, and J. Teich. “Fast Architecture Evaluation of Heterogeneous MPSoCs by Host-Compiled Simulation”. In: *Proceedings of the 15th International Workshop on Software and Compilers for Embedded Systems (SCOPES)*. St. Goar, Germany: ACM Press, May 15–16, 2012, pp. 52–61.

[RHT12c] S. Roloff, F. Hannig, and J. Teich. “Simulation of Resource-Aware Applications on Heterogeneous Architectures”. In: *Proceedings of the 8th International Summer School on Advanced Computer Architecture and Compilation for High-Performance and Embedded Systems (ACACES)*. Fiuggi, Italy, July 8–14, 2012, pp. 127–130. ISBN: 978-90-382-1987-5.

[SZSA+13] F. Sampaio, B. Zatt, M. Shafique, L. Agostini, S. Bampi, and J. Henkel. “Energy-Efficient Memory Hierarchy for Motion and Disparity Estimation in Multiview Video Coding”. In: *Design Automation and Test in Europe*

Conference (DATE) (to appear). Grenoble, France, Mar. 2013.

[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.

[SS12] P. Sanders and J. Speck. “Energy Efficient Frequency Scaling and Scheduling for Malleable Tasks”. In: *Euro-Par 2012 Parallel Processing*. Vol. 7484. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2012, pp. 167–178. ISBN: 978-3-642-32819-0. DOI: 10.1007/978-3-642-32820-6_18. URL: http://dx.doi.org/10.1007/978-3-642-32820-6_18.

[SBB12] M. Schreiber, H.-J. Bungartz, and M. Bader. “Shared Memory Parallelization of Fully-Adaptive Simulations Using a Dynamic Tree-Split and -Join Approach”. In: *Proceedings of HiPC 2012*. 2012.

[SVH13] M. Shafique, B. Vogel, and J. Henkel. “Self-Adaptive Hybrid Dynamic Power Management for Many-Core Systems”. In: *Design Automation and Test in Europe Conference (DATE) (to appear)*. Grenoble, France, Mar. 2013.

[THT12a] A. Tanase, F. Hannig, and J. Teich. “Symbolic Loop Parallelization of Static Control Programs”. In: *Proceedings of the 8th International Summer School on Advanced Computer Architecture and Compilation for High-Performance and Embedded Systems (ACACES)*. Fiuggi, Italy, July 8–14, 2012, pp. 33–36. ISBN: 978-90-382-1987-5.

[THT12b] A. Tanase, F. Hannig, and J. Teich. “Towards Symbolic Loop Parallelization for Tightly-Coupled Processor Arrays”. Work-In-Progress Presentation at the 49th Design Automation Conference (DAC), San Francisco, USA. June 3–7, 2012.

[TSDSP+12] R. Tartler, J. Sincero, C. Dietrich, W. Schröder-Preikschat, and D. Lohmann. “Revealing and Repairing Configuration Inconsistencies in Large-Scale System Software”. In: *International Journal on Software*

Tools for Technology Transfer (STTT) 14.5 (Feb. 2012), pp. 531–551. DOI: 10.1007/s10009-012-0225-2.

[Tei12] J. Teich. “Hardware/Software Co-Design: The Past, Present, and Predicting the Future”. In: *Proceedings of the IEEE* 100.Centennial-Issue (May 2012), pp. 1411–1430. DOI: 10.1109/JPROC.2011.2182009.

[TKL12] J. Teich, J. Kleinöder, and K. Lohmann, eds. *Invasive Computing*. Annual Report 2011. DFG Transregional Collaborative Research Centre 89, 2012.

[TWOS12] J. Teich, A. Weichslgartner, B. Oechslein, and W. Schröder-Preikschat. “Invasive Computing - Concepts and Overheads”. In: *Forum on Specification & Design Languages (FDL)*. Vienna, Austria, Sept. 18–20, 2012, pp. 193–200. ISBN: 978-2-9530504-5-5.

[TCDH+12] C. Tradowsky, E. Cordero, T. Deuser, M. Hübner, and J. Becker Jürgen. “Determination of On-Chip Temperature Gradients on Reconfigurable Hardware”. In: *Proceedings of the International Conference on Reconfigurable Computing and FPGAs (ReConFig)*. Cancun, Mexico: IEEE Computer Society, Dec. 5–7, 2012.

[TTHB12a] C. Tradowsky, F. Thoma, M. Hübner, and J. Becker. “LISPARC: Using an architecture description language approach for modelling an adaptive processor microarchitecture (Best Work-in-Progress (WiP) Paper Award)”. In: *7th IEEE International Symposium on Industrial Embedded Systems (SIES’12)*. June 2012.

[TTHB12b] C. Tradowsky, F. Thoma, M. Hübner, and J. Becker. “On Dynamic Run-Time Processor Pipeline Reconfiguration”. In: *Proceedings of the International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*. May 2012.

[WZT13] S. Wildermann, T. Ziermann, and J. Teich. “Game-Theoretic Analysis of Decentralized Core Allocation Schemes on Many-core Systems”. In: *Proceedings of Design, Automation and Test in Europe Conference (DATE)*. Mar. 18–22, 2013.

[WHAP+12] M. Wirnshofer, L. Heiss, A.N.Kakade, N. Pour Aryan, G. Georgakos, and D. Schmitt-Landsiedel. “Adaptive voltage scaling by in-situ delay monitoring for an image processing circuit”. In: *IEEE 15th International Symposium on Design and Diagnostics of Electronic Circuits & Systems (DDECS)*. Apr. 2012, pp. 205 –208. DOI: 10.1109/DDECS.2012.6219058.

[Zwi12] A. Zwinkau. *Resource Awareness for Efficiency in High-Level Programming Languages*. Tech. rep. 12. Karlsruhe Institute of Technology, 2012. URL: http://pp.info.uni-karlsruhe.de/uploads/publikationen/zwinkau12high.pdf.