

# **Domain-specific and Resource-aware Computing**

*Frank Hannig*

December, 2017



# Domain-specific and Resource-aware Computing

---

*Domänenspezifisches und ressourcengewahres Rechnen*

Der Technischen Fakultät der  
Friedrich-Alexander-Universität Erlangen-Nürnberg

als

## **HABILITATIONSSCHRIFT**

vorgelegt von

Dr.-Ing. Frank Hannig

Erlangen — 2017

Tag der Einreichung: 15. Dezember 2017

Erteilung der Lehrbefähigung (*venia legendi*): 11. Juli 2018

**Fachmentorat:** Professor Dr.-Ing. Jürgen Teich,  
Friedrich-Alexander-Universität Erlangen-Nürnberg  
Professor Dr.-Ing. habil. Wolfgang Schröder-Preikschat,  
Friedrich-Alexander-Universität Erlangen-Nürnberg  
Professor Dr.-Ing. Dr.-Ing. habil. Robert Weigel,  
Friedrich-Alexander-Universität Erlangen-Nürnberg

**Gutachter:** Professor Paul H. J. Kelly,  
Imperial College London  
Professor Dr. rer. nat. Theo Ungerer,  
Universität Augsburg

# Abstract

This cumulative habilitation treatise summarizes the research I have conducted with my group Architecture and Compiler Design (ACD) at the Chair of Hardware/Software Co-Design, focusing on selected results published within the last four years. My research can be divided mainly into two categories: *Resource-aware computing* and *domain-specific computing*. Both computing paradigms try to tackle the very complex programming and design challenge of parallel heterogeneous computer architectures, having different—to some extent common—goals in mind, e.g., performance, resource utilization, energy efficiency, predictability of even multiple execution qualities, or programming effort.

While resource-aware computing provides a full control loop from hardware status information to the program level and back, domain-specific computing drastically separates the concerns of algorithm development and target architecture implementation (including parallelization and low-level implementation details).

In the context of resource-aware computing, my research can be further subdivided into (1) *modeling and system simulation* and (2) *architecture/compiler co-design of invasive tightly coupled processor arrays (TCPAs)*. In the area of domain-specific computing, three approaches are presented: (3) *domain-specific high-level synthesis (HLS)*, (4) the *heterogeneous image processing acceleration framework HIPAcc*, and (5) the *ExaStencils: Advanced stencil-code engineering* approach.



# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	Contributions . . . . .	3
1.2	Papers of this Habilitation Treatise . . . . .	5
1.3	Structure of this Habilitation Treatise . . . . .	8
<b>2</b>	<b>Resource-aware Computing</b>	<b>9</b>
2.1	Invasive Computing . . . . .	10
2.2	Modeling and System Simulation . . . . .	11
2.2.1	Goals . . . . .	11
2.2.2	Approach . . . . .	11
2.2.3	Results . . . . .	13
2.2.4	Key Papers . . . . .	13
2.3	Architecture/Compiler Co-Design of Invasive TCPAs . . . . .	15
2.3.1	Challenges . . . . .	15
2.3.2	Approach . . . . .	15
2.3.3	Results . . . . .	17
2.3.4	Key Papers . . . . .	18
<b>3</b>	<b>Domain-specific Computing</b>	<b>23</b>
3.1	Domain-specific Languages . . . . .	24
3.1.1	Definition . . . . .	26
3.1.2	Classification of DSLs . . . . .	27
3.2	Domain-specific High-level Synthesis . . . . .	29
3.2.1	Goals for Domain-specific HLS . . . . .	30
3.2.2	Approach . . . . .	31
3.2.3	Results . . . . .	32
3.2.4	Domain-specific HLS Key Papers . . . . .	33
3.3	HIPAcc: The Heterogeneous Image Processing Acceleration Framework	34
3.3.1	HIPAcc Goals . . . . .	35
3.3.2	HIPAcc Approach . . . . .	35
3.3.3	HIPAcc Results . . . . .	37
3.3.4	HIPAcc Key Papers . . . . .	38
3.4	ExaStencils: Advanced Stencil-Code Engineering . . . . .	41
3.4.1	ExaStencils Goals . . . . .	42
3.4.2	ExaStencils Approach . . . . .	42

3.4.3	ExaStencils Results . . . . .	44
3.4.4	ExaStencils Key Papers . . . . .	44
<b>4</b>	<b>Conclusions</b>	<b>47</b>
<b>A</b>	<b>Bibliography</b>	<b>49</b>
A.1	General Bibliography . . . . .	49
A.2	Personal Bibliography . . . . .	60
<b>B</b>	<b>Image Credits</b>	<b>81</b>
<b>C</b>	<b>Paper Reprints</b>	<b>83</b>
C.1	Resource-aware Computing . . . . .	87
C.1.1	Modeling and System Simulation Papers . . . . .	87
	DAC '15: Execution-driven Parallel Simulation of PGAS Appli- cations on Heterogeneous Tiled Architectures . . . . .	87
	X10 '16: ActorX10: An Actor Library for X10 . . . . .	93
	ESTIMedia '17: High Performance Network-on-Chip Simulation by Interval-based Timing Predictions . . . . .	99
C.1.2	Papers on Architecture/Compiler Co-Design of Invasive TCPAs	109
	ACM TECS '14: Invasive Tightly-Coupled Processor Arrays: A Domain-Specific Architecture/Compiler Co-Design Approach . . . . .	109
	RSP '17: Constructing Fast and Cycle-Accurate Simulators for Configurable Accelerators Using C++ Templates . . .	139
	Springer JSPS '14: Symbolic Mapping of Loop Programs onto Processor Arrays . . . . .	147
	MEMOCODE '14: Symbolic Inner Loop Parallelisation for Mas- sively Parallel Processor Arrays . . . . .	177
	ACM TECS '17: Symbolic Multi-Level Loop Mapping of Loop Programs for Massively Parallel Processor Arrays . .	187
	ASAP '16: Modulo Scheduling of Symbolically Tiled Loops for Tightly Coupled Processor Arrays . . . . .	215
	Springer JSPS '14: Compact Code Generation for Tightly-Coupled Processor Arrays . . . . .	225
C.2	Domain-specific Computing . . . . .	251
C.2.1	Domain-specific HLS Papers . . . . .	251
	ASAP '14: Domain-Specific Augmentations for High-Level Syn- thesis . . . . .	251
	FPL '14: An Image Processing Library for C-based High-Level Synthesis . . . . .	257



	Springer JSPS '17: A Novel Image Impulse Noise Removal Algorithm Optimized for Hardware Accelerators . . . . .	261
	ASAP '17: Hardware Design and Analysis of Efficient Loop Coarsening and Border Handling for Image Processing	279
C.2.2	HIPAcc Papers . . . . .	289
	IEEE TPDS '16: HIPAcc: A Domain-Specific Language and Compiler for Image Processing . . . . .	289
	DATE '14: Code Generation for Embedded Heterogeneous Architectures on Android . . . . .	305
	CODES+ISSS '14: Code Generation from a Domain-specific Language for C-based HLS of Hardware Accelerators . .	311
	Elsevier JPDC '14: Towards a Performance-portable Description of Geometric Multigrid Algorithms using a Domain-specific Language . . . . .	321
	FPL '16: FPGA-based Accelerator Design from a Domain-Specific Language . . . . .	333
	Springer JSPS '17: Loop Parallelization Techniques for FPGA Accelerator Synthesis . . . . .	343
	LCTES '17: Auto-vectorization for Image Processing DSLs . . .	369
C.2.3	ExaStencils Papers . . . . .	379
	ICCSA '14: An Evaluation of Domain-Specific Language Technologies for Code Generation . . . . .	379
	Euro-Par '14: ExaStencils: Advanced Stencil-Code Engineering	389
	WOLFHPC '14: ExaSlang: A Domain-Specific Language for Highly Scalable Multigrid Solvers . . . . .	401
	Springer LNCSE '16: Systems of Partial Differential Equations in ExaSlang . . . . .	411



## List of Abbreviations

<b>ACD</b>	Architecture and Compiler Design
<b>ADAS</b>	Advanced Driver Assistance System
<b>ALU</b>	Arithmetic Logic Unit
<b>APGAS</b>	Asynchronously Partitioned Global Address Space
<b>ASIC</b>	Application-Specific Integrated Circuit
<b>ASIP</b>	Application-Specific Instruction set Processor
<b>AST</b>	Abstract Syntax Tree
<b>AVX</b>	Advanced Vector Extensions
<b>BRAM</b>	Block Random Access Memory
<b>CGRA</b>	Coarse-Grained Reconfigurable Architecture
<b>CMOS</b>	Complementary Metal Oxide Semiconductor
<b>CNC</b>	Computer Numerical Control
<b>CPU</b>	Central Processing Unit
<b>DAG</b>	Directed Acyclic Graph
<b>DLP</b>	Data-Level Parallelism
<b>DoP</b>	Degree of Parallelism
<b>DSL</b>	Domain-Specific programming Language
<b>DSP</b>	Digital Signal Processor
<b>FLOPS</b>	FLoating-point Operations Per Second
<b>FPGA</b>	Field Programmable Gate Array
<b>FU</b>	Functional Unit
<b>GPU</b>	Graphics Processing Unit

## LIST OF ABBREVIATIONS

---

<b>HDL</b>	Hardware Description Language
<b>HLS</b>	High-Level Synthesis
<b>HPC</b>	High-Performance Computing
<b>HSA</b>	Heterogeneous System Architecture
<b>IC</b>	Integrated Circuit
<b>ILP</b>	Instruction-Level Parallelism
<b>IR</b>	Intermediate Representation
<b>LoC</b>	Lines of Code
<b>LPGS</b>	Locally Parallel, Globally Sequential
<b>LSGP</b>	Locally Sequential, Globally Parallel
<b>LUT</b>	Look-Up Table
<b>MIPS</b>	Million Instructions Per Second
<b>MPI</b>	Message Passing Interface
<b>MPSoC</b>	Multi-Processor System-on-Chip
<b>NoC</b>	Network-on-Chip
<b>NPP</b>	Nvidia Performance Primitives
<b>OpenCL</b>	Open Computing Language
<b>OpenCV</b>	Open Source Computer Vision
<b>PC</b>	Personal Computer
<b>PDE</b>	Partial Differential Equation
<b>PE</b>	Processing Element
<b>QoR</b>	Quality of Results
<b>RISC</b>	Reduced Instruction Set Computer
<b>SDK</b>	Software Development Kit
<b>SIMD</b>	Single Instruction, Multiple Data

---

<b>SNR</b>	Signal-to-Noise Ratio
<b>SoC</b>	System-on-Chip
<b>SQL</b>	Structured Query Language
<b>SSE</b>	Streaming SIMD Extensions
<b>TCPA</b>	Tightly Coupled Processor Array
<b>TI</b>	Texas Instruments
<b>TPDL</b>	Target Platform Description Language
<b>UML</b>	Unified Modeling Language
<b>VHDL</b>	VHSIC Hardware Description Language
<b>VHLL</b>	Very High-Level programming Language
<b>VHSIC</b>	Very High Speed IC
<b>VLIW</b>	Very Long Instruction Word



# 1 Introduction

Diversity accompanies electronics ab initio. First electronic devices were assembled from discrete components to fulfill a particular purpose, i.e., they were designed and implemented in an application-specific way. Quickly, more and more transistors, the basis for all logic gates and registers, could be crammed onto integrated circuits and offered an almost unlimited plethora of possibilities for digital circuits. Thanks to the steady advances in Complementary Metal Oxide Semiconductor (CMOS) technology, the number of transistors on the same chip footprint has grown exponentially within the last half-century. This observation/projection is widely known as Moore's law [107] and goes along with Dennard scaling [32], which roughly says that the power density remains constant as transistors are getting smaller, i.e., performance per watt is also growing exponentially. In the context of microprocessors, Central Processing Unit (CPU) designers harnessed this growth 30 years long by turning it into performance gains. First, because not only more transistors but also faster transistors have been built, and thus even CPUs could run increasingly at higher clock rates. Second, ever more complex CPUs could be designed that can do more work per clock cycle. That is, more powerful instructions and larger data types, deeper processor pipelines, sophisticated branch prediction, multiple parallel instructions, or instruction reordering in the case of out-of-order execution. A third performance boost has been accomplished thanks to larger and hierarchically organized caches. However, the seemingly never-ending performance boost of single-core processors was disrupted around 2005. Herb Sutter described this turning point in microprocessor history in his highly cited article "A Fundamental Turn Toward Concurrency in Software" [134] and the 2009 updated version "The Free Lunch is Over" [135], respectively. Herein, Sutter states that there is almost no progress in achieving higher clock frequencies. Although single transistors and small Integrated Circuits (ICs) have been experimentally proven to run at frequencies in the two-digit to three-digit gigahertz (GHz) range, processor clock rates have nearly saturated due to several physical effects. (a) Shrinking features sizes came along with severe current leakage<sup>1</sup> issues, (b) the power consumption is limited, and the two items as mentioned above lead to (c) profound heat problems, in other words, heat is hard to dissipate. Collectively, these issues are referred to as the *power wall*. In addition, support for instruction level parallelism cannot be arbitrarily scaled within a processor core because increasing the number of functional units gets quickly exceedingly complex for both superscalar and Very Long Instruction Word (VLIW) processors [30]. All these threats

---

<sup>1</sup>Leakage in semiconductor devices denotes the effect of capacity-connected components, such as transistors or diodes, to draw a small amount of current even though they are switched off.

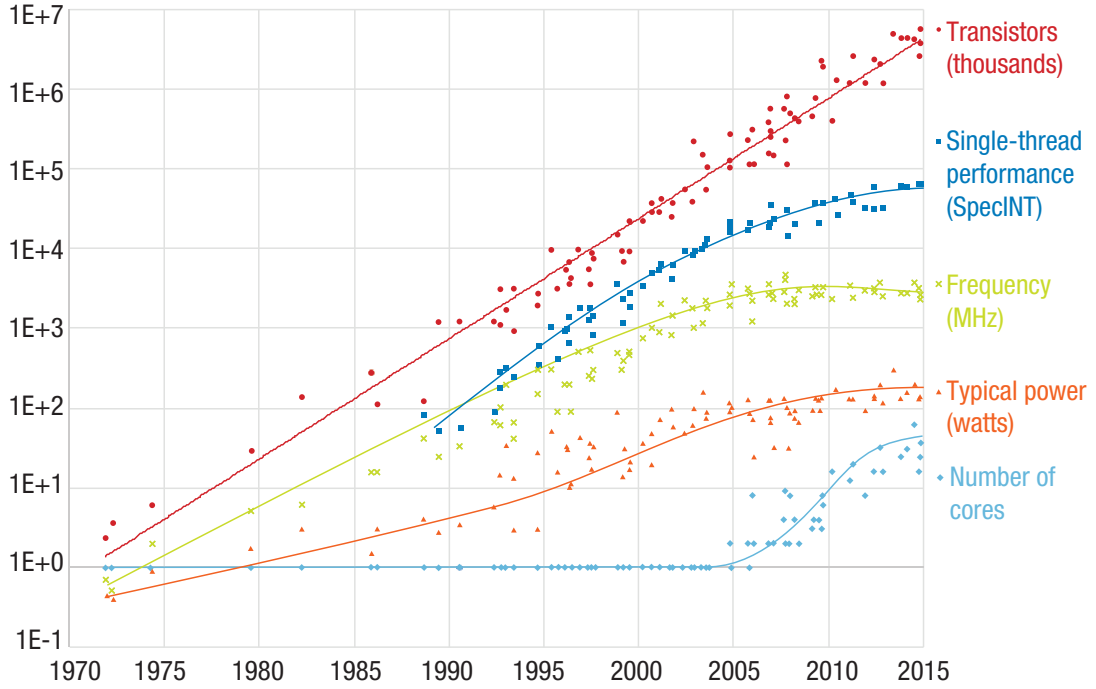


Figure 1.1: Evolution of microprocessors with respect to (i) number of transistors per chip, (ii) benchmarked performance, (iii) clock rate, (iv) typical power consumption, and (v) number of processor cores. Figure reprinted from [23, p. 46]. © 2015 by IEEE.

induced the end of Dennard scaling in 2005 and turned processor architectures into multicore designs henceforward (see Figure 1.1).

While the speed of Moore’s projection has slightly slowed down, the steady miniaturization of feature sizes and the associated exponential growth of processor designs is still ongoing. However, as mentioned above, technology shrinking also continuously leads to higher energy densities, and thus the situation has become even more severe concerning power consumption because chips can handle only a limited power budget. As a consequence, the potentially available chip area might not be entirely utilized or at least not simultaneously. This phenomenon is also known as *utilization wall* [45] and accordingly as *dark silicon* [35], which denotes chip areas that must remain inactive most of the time. As a conclusion: Future systems will only scale if their energy efficiency will considerably improve—this reasoning holds for both embedded and portable devices, such as smartphones and tablets as well as large-scale systems used for High-Performance Computing (HPC). *Customization* and *heterogeneity*, e.g., in the form of custom-tailored memory hierarchies, sophisticated interconnection networks, and application-specific compute components, such as accelerators, are the key to success for future performance gains [127, 91].



Additionally, each CMOS process shrink is progressively ridden with imperfections (i.e., variability [14, 6]) and unreliability of the devices. These technological hurdles combined with the sheer complexity of heterogeneous Multi-Processor System-on-Chip (MPSoC) architectures raise numerous questions on how to design, test, and program such systems while having multiple—possibly contrary—goals in mind. It is very challenging to find solutions satisfying different objectives, such as performance, resource utilization, energy efficiency, resiliency [69], predictability of even multiple execution qualities [140], effective exploitation of concurrency, or programming effort. Thus, novel design methodologies and computing paradigms are required to fuel research and scientific discoveries with undiminished pace.

## 1.1 Contributions

In recent years, my research mission has been *to master the design and programming complexity of parallel systems as well as their rising heterogeneity*. The considered systems span a wide range, from architectures targeted for embedded applications to clusters for HPC. A particular focus is on accelerators (e.g., processor arrays, Graphics Processing Units (GPUs), or dedicated hardware implemented in Field Programmable Gate Arrays (FPGAs)) or their combination as part of a heterogeneous system (e.g., compute nodes equipped with accelerator technology, MPSoCs, or NoC-based tiled heterogeneous architectures). Investigated are programming languages, methods and techniques for compilation and application mapping, as well as the synthesis and simulation of parallel processor architectures. Here, two strategic approaches are researched—which may appear utterly different at first. The one is *resource-aware computing*, and the other is *domain-specific computing*. While the former makes numerous static architecture properties as well as runtime status information explicitly visible (e.g., at the program level as in the case of *resource-aware programming* [139, 142, P69]), the latter tries to wholly separate application development from an underlying architecture (i.e., a programmer is shielded from low-level implementation, mapping and parallelization details). Concerning these two computing paradigms, my principal research contributions can be very briefly summarized as follows:

### Resource-aware Computing

**(1) Modeling and System Simulation.** Invasive computing [139, 142] is resource-aware computing at its best. To study resource-aware programming [139, P69] and *invasive* resource management strategies [68], means for application modeling and to mimic the execution behavior on not yet existing manycore architectures are required. The modeling of *invasive* applications includes programming extensions for resource-aware programming [P69] as well as concurrent models of computation, such as *actor models* [P42, P16]. Timed functional simulation

techniques [P65, P24] are used to assess execution qualities and to explore variants [P62] of tiled heterogeneous manycore architectures. Measures for fast full system simulation (parallel simulation [P24] and Network-on-Chip (NoC) simulation [P47, P4]) have been investigated and evaluated for streaming-based multimedia applications [P16] as well as X10 benchmarks [P24, P4].

- (2) **Architecture/Compiler Co-Design of Invasive TCPAs.** Invasive Tightly Coupled Processor Arrays (TCPAs) [J18] combine architectural research in the field of parallel on-chip accelerators with the paradigm of invasive computing. The concepts to dynamically *invade* processors and to *retreat* from them are directly integrated into an invasive TCPA at the hardware level [P71, P67, J18]. This opens opportunities for ultrafast resource reservation and adaptivity (e.g., power management [J19, J18] or fault tolerance [P23, P21, 84]). These techniques have been mainly evaluated using cycle-accurate simulation [P101, J18, P5].

Regarding compilation for TCPAs, (a) efficient, yet *compact code generation* [P46, J17] as well as (b) *symbolic tiling* and *symbolic scheduling* [P48, J15, P32, P19, P15, J1] as program transformations for the symbolic parallelization of nested loop programs with uniform data dependences have been investigated.

### Domain-specific Computing

- (3) **Domain-specific High-level Synthesis.** Domain-specific High-Level Synthesis (HLS) provides programming abstractions to ease the problem specification and thus productivity. In the case of this habilitation treatise, the domain is image processing. The considered techniques are based on *template metaprogramming* and *generative programming*. The proposed domain-specific concepts have been investigated employing external Domain-Specific programming Language (DSL) constructs followed by code transformations in the case of the PARO HLS research tool [P37, J2] and the form of a template library in the case of C-based commercial tools, such as Xilinx Vivado HLS [P33, P9].

- (4) **The Heterogeneous Image Processing Acceleration Framework.** HIPAcc is a DSL embedded in C++ and a compiler framework for the domain of image processing. It captures domain knowledge in a compact and intuitive language and employs source-to-source translation combined with various optimizations to achieve an excellent productivity paired with performance portability. The HIPAcc approach has been applied and evaluated for a broad variety of parallel accelerator architectures, including manycore processors [J9, J12], such as Nvidia and AMD GPUs and Intel Xeon Phi, embedded GPUs [P41], Xilinx and Intel/Altera FPGAs [P31, P13, J5], and vector units [P11].

**(5) The ExaStencils Approach.** To reduce the foreseen performance/productivity gap of upcoming exascale platforms, a unique, tool-assisted co-design approach specific for the domain of multigrid methods based on stencil computations is developed within ExaStencils [P35], [P38]. The fundamental concept of ExaStencils is a multi-layered external DSL in combination with a modular transformation and optimization framework [P29]. The approach has been evaluated concerning productivity and scalability, among other things, for a generated geometric multigrid solver on the JUQUEEN supercomputer [C1].

I have conducted the in this habilitation treatise cumulated research jointly together with nine doctoral researchers as well as bachelor and master students from my research group *Architecture and Compiler Design (ACD)*.<sup>2</sup> I have been the principal investigator for (1), (4), and (5) within the DFG Transregional Collaborative Research Centre 89 “Invasive Computing,” the DFG Research Training Group 1773 “Heterogeneous Image Systems,” and the DFG Priority Programme 1648 “Software for Exascale Computing,” respectively. In the case of (2), I have been a conceptual contributor, and in the case of (3), I have been the scientific leader and conceptual contributor.

## 1.2 Papers of this Habilitation Treatise

This document is a cumulative habilitation treatise. From my 165 peer-reviewed publications listed in Appendix A.2 (page 60ff.), I have opted for the following 25 key contributions of my research. These form the chief part of my cumulative habilitation treatise. The full texts (i.e., reprints) of these publications are provided on page 83ff. in Appendix C.

### Resource-aware Computing

#### Modeling and System Simulation Papers

<b>DAC ’15</b> page 87ff.	Roloff, Schafhauser, Hannig, and Teich. “Execution-driven parallel simulation of PGAS applications on heterogeneous tiled architectures” [P24]
<b>X10 ’16</b> page 93ff.	Roloff, Pöpl, Schwarzer, Wildermann, Bader, Glaß, Hannig, and Teich. “ActorX10: An actor library for X10” [P16]
<b>ESTIMedia ’17</b> page 99ff.	Roloff, Hannig, and Teich. “High performance network-on-chip simulation by interval-based timing predictions” [P4]

---

<sup>2</sup>In November 2017, my team, the ACD group, consists of eleven doctoral researchers and one post-doctoral researcher.

### Papers on Architecture/Compiler Co-Design of Invasive TCPAs

<b>ACM TECS '14</b> page 109ff.	Hannig, Lari, Boppu, Tanase, and Reiche. "Invasive tightly-coupled processor arrays: A domain-specific architecture/compiler co-design approach"	[J18]
<b>RSP '17</b> page 139ff.	Witterauf, Hannig, and Teich. "Constructing fast and cycle-accurate simulators for configurable accelerators using C++ templates"	[P5]
<b>Springer JSPS '14</b> page 147ff.	Teich, Tanase, and Hannig. "Symbolic mapping of loop programs onto processor arrays"	[J15]
<b>MEMOCODE '14</b> page 177ff.	Tanase, Witterauf, Teich, and Hannig. "Symbolic inner loop parallelisation for massively parallel processor arrays"	[P32]
<b>ACM TECS '17</b> page 187ff.	Tanase, Witterauf, Teich, and Hannig. "Symbolic multi-level loop mapping of loop programs for massively parallel processor arrays"	[J1]
<b>ASAP '16</b> page 215ff.	Witterauf, Tanase, Hannig, and Teich. "Modulo scheduling of symbolically tiled loops for tightly coupled processor arrays"	[P15]
<b>Springer JSPS '14</b> page 225ff.	Boppu, Hannig, and Teich. "Compact code generation for tightly-coupled processor arrays"	[J17]

## Domain-specific Computing

### Domain-specific HLS Papers

<b>ASAP '14</b> page 251ff.	Schmid, Tanase, Hannig, Teich, Bhadouria, and Ghoshal. "Domain-specific augmentations for high-level synthesis"	[P37]
<b>FPL '14</b> page 257ff.	Schmid, Apelt, Hannig, and Teich. "An image processing library for C-based high-level synthesis"	[P33]
<b>Springer JSPS '17</b> page 261ff.	Bhadouria, Tanase, Schmid, Hannig, Teich, and Ghoshal. "A novel image impulse noise removal algorithm optimized for hardware accelerators"	[J2]
<b>ASAP '17</b> page 279ff.	Özkan, Reiche, Hannig, and Teich. "Hardware design and analysis of efficient loop coarsening and border handling for image processing"	[P9]

## HIPAcc Papers

<b>IEEE TPDS '16</b> page 289ff.	Membarth, Reiche, Hannig, Teich, Körner, and Eckert. "HIPAcc: A domain-specific language and compiler for image processing"	[J9]
<b>DATE '14</b> page 305ff.	Membarth, Reiche, Hannig, and Teich. "Code generation for embedded heterogeneous architectures on Android"	[P41]
<b>CODES+ISSS '14</b> page 311ff.	Reiche, Schmid, Hannig, Membarth, and Teich. "Code generation from a domain-specific language for C-based HLS of hardware accelerators"	[P31]
<b>Elsevier JPDC '14</b> page 321ff.	Membarth, Reiche, Schmitt, Hannig, Teich, Stürmer, and Köstler. "Towards a performance-portable description of geometric multigrid algorithms using a domain-specific language"	[J12]
<b>FPL '16</b> page 333ff.	Özkan, Reiche, Hannig, and Teich. "FPGA-based accelerator design from a domain-specific language"	[P13]
<b>Springer JSPS '17</b> page 343ff.	Reiche, Özkan, Hannig, Teich, and Schmid. "Loop parallelization techniques for FPGA accelerator synthesis"	[J5]
<b>LCTES '17</b> page 369ff.	Reiche, Kobylko, Hannig, and Teich. "Auto-vectorization for image processing DSLs"	[P11]

## ExaStencils Papers

<b>ICCSA '14</b> page 379ff.	Schmitt, Kuckuk, Köstler, Hannig, and Teich. "An evaluation of domain-specific language technologies for code generation"	[P38]
<b>Euro-Par '14</b> page 389ff.	Lengauer, Apel, Bolten, Größlinger, Hannig, Köstler, Rüde, Teich, Grebhahn, Kronawitter, Kuckuk, Rittich, and Schmitt. "ExaStencils: Advanced stencil-code engineering"	[P35]
<b>WOLFHPC '14</b> page 401ff.	Schmitt, Kuckuk, Hannig, Köstler, and Teich. "ExaSlang: A domain-specific language for highly scalable multigrid solvers"	[P29]
<b>Springer LNCSE '16</b> page 411ff.	Schmitt, Kuckuk, Hannig, Teich, Köstler, Rüde, and Lengauer. "Systems of partial differential equations in ExaSlang"	[C1]

### 1.3 Structure of this Habilitation Treatise

The rest of my habilitation treatise is organized as follows: In the next

Chapter 2 “Resource-aware Computing,” (pages 9 to 21)

I briefly introduce *resource-aware computing* and provide a short overview of the Trans-regional Collaborative Research Centre 89 “Invasive Computing”. Subsequently, I summarize my research results that emerged from the Collaborative Research Centre in the areas of *modeling and system simulation* (Section 2.2) and *architecture/compiler co-design of invasive tightly coupled processor arrays* (Section 2.3) as well as the corresponding publications; their reprints can be found in Appendix C in order not to impair reading fluency and to ease partial printing.

Chapter 3 “Domain-specific Computing” (pages 23 to 46)

introduces the key concepts of *domain-specific computing* and fundamentals of domain-specific programming languages briefly. Afterward, I provide an overview of the respective projects (domain-specific HLS in Section 3.2, HIPAcc in Section 3.3, and ExaStencils in Section 3.4) as well as their basis forming design and implementation techniques, followed by the corresponding research results. Reprints of the related publications can also be found in Appendix C in order not to impair reading fluency and to ease partial printing.

Chapter 4 “Conclusions” (pages 47 to 48)

summarizes and concludes my cumulative habilitation treatise. Herein, I identify commonalities of my two main research branches and provide ideas for future research directions.

Appendix A “Bibliography,” (pages 49 to 79)

Appendix B “Image Credits,” and (page 81)

Appendix C “Paper Reprints” (page 83ff.)

provide both general and personal bibliographies (including a complete list of my own publications), image credits, and reprints of the papers I opted for my cumulative habilitation treatise, respectively.

## 2 Resource-aware Computing

Since the term *resource-aware computing* as a whole is not very widespread, let us decompose it into its parts according to the Oxford English Dictionary<sup>3</sup>:

**“resource”** *noun*; “A stock or supply of money, materials, staff, and other assets that can be drawn on by a person or organization in order to function effectively.”

**“aware”** *adjective*; “[with *adverb* or in combination] Concerned and well informed about a particular situation or development.”

**“computing”** *noun*; “The use or operation of computers.”

*Resources* in the sense of a computing system can be *physical* components, such as processing units (e.g., arithmetic units or processor cores), memory and input/output devices, or *virtual* components, e.g., files, network connections, and memory areas. Obviously, the resources of a computer are limited, thus, some means to manage them are required (allocation and releasing of resources, dealing with contention, etc.). Resource management necessitates some *awareness* about the static architecture properties (amount and type of resources) as well as about the status of resources at runtime (i.e., dynamically changing qualities, such as availability, utilization, reliability, or temperature). Static properties are sometimes considered during the design of a program or at compile time, e.g., that an application is parallelized for a certain number of compute resources. But often, an underlying architecture is completely abstracted away by chopping the workload into smaller portions that are handled by a runtime system, which takes care of assignment and scheduling. Similarly, if the workload is not known at compile time (e.g., unknown problem size or the computational effort depends on the composition of input data) or barely predictable combinations of (parallel) applications have to be executed, resource management, workload distribution, and scheduling typically happen in a runtime system. However, such an implicit resource management by the runtime system—in combination with programs written in high-level languages—*thieves* the control of basic resources from the programmer and thus might only lead to a suboptimal program execution.

The *challenge of resource-aware program execution* becomes even more intricate when having the computer architecture trends in mind (see Chapter 1). Consequently, many research questions arise: How to manage and program heterogeneous architectures with thousands of cores? How to deal with heat dissipation and power consumption? How

---

<sup>3</sup><https://en.oxforddictionaries.com>

to optimize yield by coping with the increasing variability in semiconductor fabrication? How to treat faults and aging processes over the lifetime of a chip? How to provide predictability—not only regarding timing but also other non-functional qualities of parallel program execution, such as power and reliability?

One fundamental principle of parallel computing that examines above raised questions is *invasive computing*.

### 2.1 Invasive Computing

In [139], Jürgen Teich coined the terms *invasive algorithms* and *invasive architectures*. The associated novel concepts have been generalized and subsumed shortly afterward under *invasive computing* and by the DFG-funded Transregional Collaborative Research Centre 89 of the same name<sup>4</sup> [142].

Resource awareness has top priority in invasive computing by introducing it at the level of application programming. Here, *resource-aware programming* facilitates a given program to explore and dynamically distribute its computations to (neighbor) processors. Then, to execute portions of code that have a high Degree of Parallelism (DoP) in parallel based on the available region on a given multi-processor architecture. Afterward, once the program terminates or if the DoP should be lower again, the program should deallocate resources and resume execution again, e.g., sequentially on a single processor. In the nomenclature [139, 142, P69] of invasive computing these different phases of operation are called *invade*, *infect*, and *retreat*, denoting (i) resource exploration and claiming, (ii) code and data distribution as well as parallel program execution, and (iii) release of resources, respectively. By means of invasion, an application will thus be able to spread its computations for parallel execution based on the availability and the actual state of the underlying resources, such as utilization, load, or temperature. Transitions from one phase (i.e., *invade*, *infect*, and *retreat*) to another one not necessarily have to follow always a straight order. As illustrated by the back edges of the state chart depicted in Figure 2.1, there can be also situations where a claimed set of resources should be extended (*reinvasion*) or partially freed (*partial retreat*), or used for execution again (*reinfect*). The introduced programming constructs for resource-aware programming are embedded into the parallel programming language X10 [28] as developed by IBM using a library-based approach [P69].

While the Transregional Collaborative Research Centre 89 looks at invasive computing in its entirety (four project areas covering (A) fundamentals, language, and algorithm research, (B) architectural research, (C) compiler, simulation, and run-time support, (D) applications), my research contributions are in the areas of *modeling and simulation of*

---

<sup>4</sup>The CRC/Transregio 89 “Invasive Computing” was established by the German Research Foundation (DFG) in July 2010 and is currently in its second funding phase until end of June 2018.



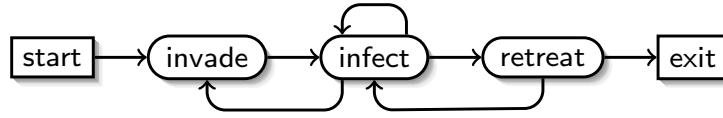


Figure 2.1: State chart of an invasive program. Figure reprinted from [P69, p. 49].

*invasive applications and invasive architectures* as well as *compilation and architecture research* for tightly coupled processor arrays.

## 2.2 Modeling and System Simulation

### 2.2.1 Goals

Invasive computing is a hotbed for a novel parallel computing paradigm with multifarious research directions, including theory, programming languages, and applications as well as architectural design. In order to explore and optimize different invasive architectures, invasion strategies, and invasive programming approaches, simulation techniques that cover all these aspects are indispensable. Without the need to have full hardware or software implementations available, the goals are primarily (a) modeling and behavioral simulation of invasive applications and (b) simulation of invasive architectures.

### 2.2.2 Approach

In order to fulfill the goals mentioned above, the simulation platform InvadeSIM [P65, P62, P24] has been developed that provides a fast simulation approach for hundreds of competing applications on large heterogeneous architectures, allows the modeling of customized heterogeneous invasive multi-tile architectures, and supports the simulation of the complete set of X10 programming language constructs as well as all novel invasive programming constructs, such as *invade*, *infect*, and *retreat*. An overview of the developed timed functional simulation platform InvadeSIM is shown in Figure 2.2. The simulator allows to quickly customize an invasive multi-tile architecture to be evaluated by changing a number of parameters, including topology network parameters, or number of tiles and processor types in each tile—denoted as architecture model in the figure. Application modeling in InvadeSIM is carried out in X10 [28] and may use the InvadeX10 language extensions [P69, P55] for exploiting the invasive command set [142, P69], an actor-based programming model [P42, P16], or combinations of both. An actor model [1, 58] exposes asynchronous buffered communication paths and decouples these from the control flow of the concurrently executed application parts. Our actor model is called ActorX10 [P16] and is implemented on top of the Asynchronously Partitioned Global Address Space (APGAS) principles of X10.

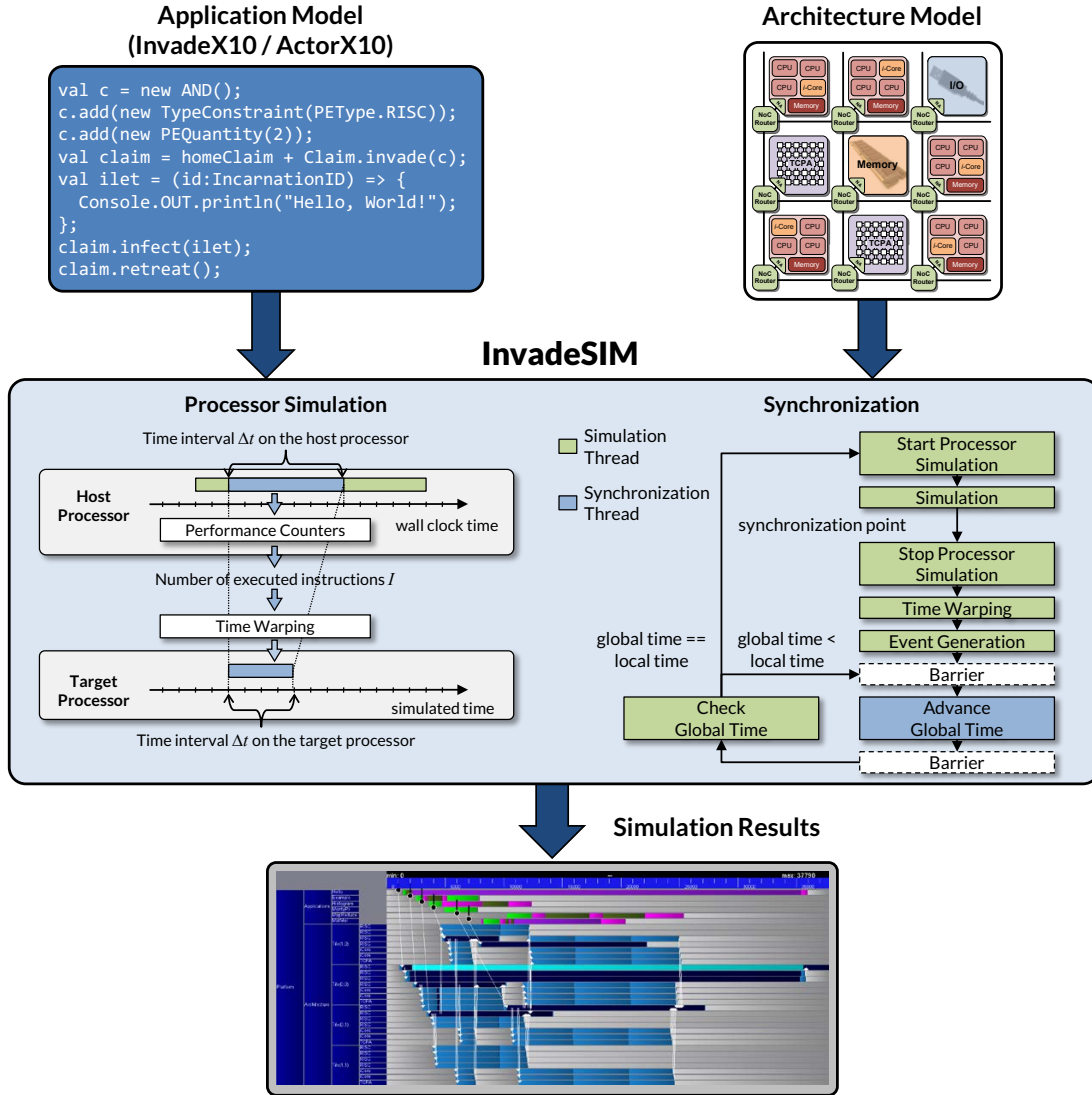


Figure 2.2: Overview of the resource-aware timed functional system simulation platform InvadeSIM. For more details on the synchronization and *time warping* concepts, we refer to [P65, P62].

The core components to simulate parallel applications on a modeled heterogeneous multi-tile MPSoC architecture are sketched in the middle of Figure 2.2. These components contain a novel concept of *approximately timed simulation* and a *discrete event synchronization* mechanism for the simulation of multiple processors. Many existing simulation frameworks for heterogeneous MPSoCs are based either on cycle-accurate or trace-based approaches and are typically much too slow. Instead, our processor simulation approach is a hybrid method based on performance counters and analytic

models, which we call *time warping* [P65]. A synchronization mechanism preserves the causality of simulation events in case of multiple processor simulations. The discrete event simulation approach combines the functional execution of an application (consisting of so-called *i-lets* in the notion of invasive computing [P69, 142]) with the timing simulation according to the computational properties of the target processor architecture on which it was mapped at runtime. Eventually, the simulation results may be analyzed or visualized (e.g., by a graphical user interface or trace viewer), or the timing values serve as quality number for the evaluation of application mappings [P18, 140, 130] or architecture exploration [P62]. Since the latter two tasks are very complex regarding computing time, it is of utmost importance to have an ultrafast yet highly timing-accurate system simulator. Here, novel *parallel simulation* [P24] and *hybrid network-on-chip simulation* [P4] techniques have been developed, implemented, and evaluated.

### 2.2.3 Results

To evaluate the scalability and performance of our parallelized version of the InvadeSIM system simulator, we have modeled multi-tile processor architectures up to 64 cores (Reduced Instruction Set Computer (RISC) or Application-Specific Instruction set Processor (ASIP)) and run the simulator on a host machine with twelve cores [P24]. We could demonstrate a linear speedup in simulation time of the parallel simulation (running on the host with up to twelve cores) with respect to a sequential simulation—while the absolute simulation performance is in a five-figure Million Instructions Per Second (MIPS) range.

The developed hybrid NoC simulation technique [P4] achieves speedups of one to three orders of magnitude compared with cycle-accurate NoC simulation, while depending inversely related on the NoC size ranging from  $4 \times 4$  to  $16 \times 16$  tiles, and preserving a timing accuracy of above 95%. The NoC simulation has been integrated into InvadeSIM, to evaluate how much the full system simulation can benefit. Where we could showcase decent speedups for complex streaming-based multimedia applications [P16, P4] written in ActorX10 [P16] and X10 applications from the IMSuite [48] benchmark.


### 2.2.4 Key Papers

**DAC '15**      Roloff, Schafhauser, Hannig, and Teich. "Execution-driven parallel simulation      [P24]  
page 87ff.      of PGAS applications on heterogeneous tiled architectures"

The DAC '15 paper presents the parallel execution-driven simulator for the efficient simulation of heterogeneous tile-based multicore architectures. Four novel parallel discrete-event simulation techniques were proposed, which map thread-level parallelism within the appli-


cations to core-level parallelism on the simulated target architecture and back to thread-level parallelism on the simulation host machine. In all these approaches, the correct synchronization and activation of the host threads are essential. Experiments with parallel real-world applications are used to compare the different techniques against each other and demonstrate that, on average, 8.2 times faster simulations than a sequential simulation can be achieved on a 12-core Intel Xeon processor while reaching a simulation speed of up to 86,000 MIPS.

**X10 '16**  
page 93ff.

Roloff, Pöpl, Schwarzer, Wildermann, Bader, Glaß, Hannig, and Teich. "ActorX10: An actor library for X10" [P16] 

Based on the foundation "Towards Actor-oriented Programming on PGAS-based Multicore Architectures" by Sascha Roloff et al. in [P42], the X10 '16 paper consequently formalizes the actor model on the basis of the APGAS programming model as used in X10. The implementation, named ActorX10, seamlessly integrates into X10 in the form of a class library. It eases parallel program development by making the communication between different parts of the program explicit and by separating the control flow aspects from the computational aspects. This kind of abstraction is very versatile as we demonstrate for applications from the embedded system (object detection streaming pipeline) and HPC domain (proxy application for tsunami simulation).

**ESTIMedia '17**  
page 99ff.

Roloff, Hannig, and Teich. "High performance network-on-chip simulation by interval-based timing predictions" [P4] 

The ESTIMedia '17 paper focuses on NoC simulation and presents a hybrid approach, where the timing of packet delays in a NoC is modeled accurately, yet, for scalability reasons, predictable and contiguous communications are detected. These are exploited to forward the simulation time appropriately instead of simulating *flit-by-flit*.<sup>a</sup> We evaluated our hybrid simulation technique for pure NoC simulation as well as full system simulation. In both cases decent speedups were achieved.

---

<sup>a</sup>Flit is an acronym for *flow control digit* and denotes the smaller units of a subdivided network packet.

The work above was conducted with doctoral researcher Sascha Roloff. He significantly contributed to the research on the simulation of invasive computing systems, assisted by David Schafhauser with his bachelor's thesis on parallel simulation techniques. The conceptual design of ActorX10 was carried out jointly with partners from the CRC/Trans-regio 89, while Sascha Roloff additionally performed the X10 library implementation.

## 2.3 Architecture/Compiler Co-Design of Invasive Tightly Coupled Processor Arrays

Looking at programmable accelerators in its entirety has a long tradition in my research group Architecture and Compiler Design (ACD). The co-design aspect of processor architecture design (incl. means for simulation and prototyping) and compiler research (e.g., mapping, scheduling, and code generation) has always been an overarching objective—ACD’s motto is to build both *compiler-friendly architectures* as well as *architecture-friendly / retargetable design tools and compilers*. In the course of research, we developed a highly parametrizable class of parallel on-chip processor arrays, which consists of lightweight and weakly programmable VLIW Processing Elements (PEs) that are tightly interconnected over a reconfigurable network [P108, P104, P103], as well as the corresponding tool flow [P99, C9, J24]. These Tightly Coupled Processor Arrays (TCPAs) are particularly suitable for the acceleration of nested affine loop programs [33], and have been proven for high-throughput processing [T1, C8] while being low power [P90, J23, J21] at the same time, i.e., such architectures are highly energy-efficient [J22, J18, J4].

### 2.3.1 Challenges

Many research questions emerged from invasive computing in the context of processor arrays: How can the principles of resource-aware programming be supported in hardware, i.e., ultrafast within a few clock cycles? Can invasive computing be an enabler for optimizing execution qualities, such as energy efficiency, and non-functional requirements, such as fault tolerance? Previous research on mapping loop programs onto processor arrays or Coarse-Grained Reconfigurable Architectures (CGRAs) considered both fixed problem sizes and a static number of PEs preferentially—but, how to proceed if the number of available resources (PEs) is only known at runtime? How can costly compilation at runtime or storing of all possible mappings be circumvented?

### 2.3.2 Approach

As mentioned before, TCPAs are primarily used as accelerators, reflected by dedicated TCPA tiles as part of an invasive heterogeneous tiled architecture (see Figure 2.3). To adopt the principles of invasive computing at hardware level, each PE of a TCPA is equipped with an *invasion controller* (or short *iCtrl* in Figure 2.3), giving it the capability to acquire, reserve, and then release the PEs in its neighborhood in a fast cycle-based manner [P71, P67, J18]. This decentralized approach ensures scalable and reliable resource management in future many-core architectures with hundreds to thousands of PEs, where centralized approaches do not scale anymore because of latency and fault-tolerance reasons.

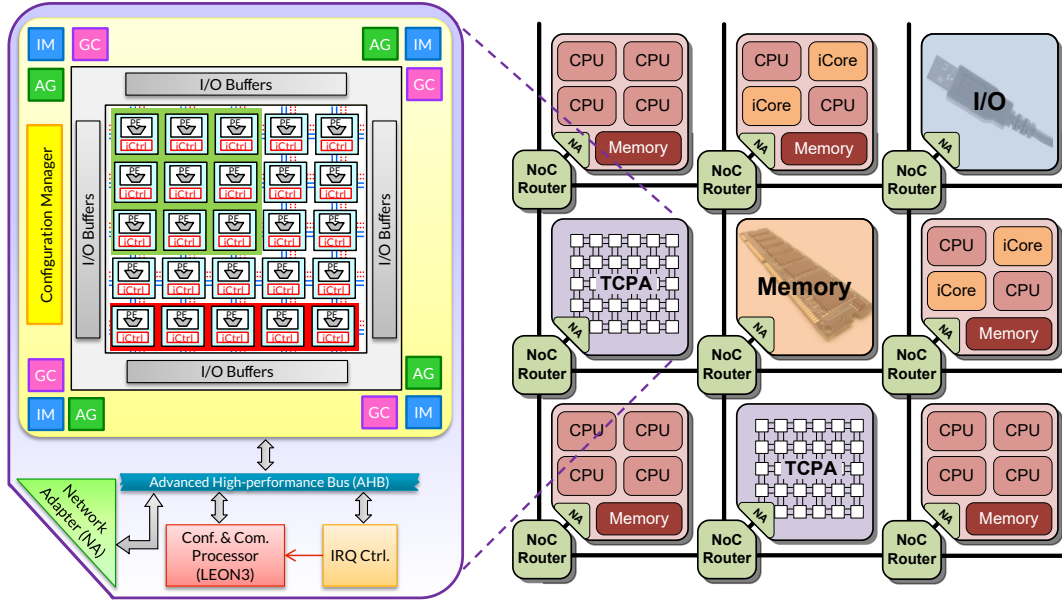


Figure 2.3: On the right, a schematic representation of an invasive heterogeneous tiled architecture is shown. An abstract architectural view of a TCPA tile is shown on the left. The abbreviations AG, GC, IM, and NA stand for address generator, global controller, invasion manager, and network adapter. Figure adapted from [J18<sup>5</sup>, 133:5].

The invasion controllers enable also hierarchical power management in TCPAs by dynamically turning regions of PEs on and off, depending on invade and retreat phases [P57, J19]. Most of the investigations on hardware-based management of TCPA resources [P71, P67, J18<sup>5</sup>], power management [P57, J19, J18<sup>5</sup>], and fault tolerance [P23, P21, 84] were assessed using cycle-accurate simulation [P101, J18<sup>5</sup>, P5<sup>5</sup>]. Since simulation is essential, our most recent approach [P5<sup>5</sup>] profoundly exploits C++ templates, which allow for highly configurable architectures using parameters at synthesis time.

Regarding compilation techniques, the main idea is to provide (a) symbolic transformations for loop tiling, i.e., to use parametrized tile sizes to represent a (statically) unknown number of PEs, as well as (b) the determination of corresponding symbolic latency-optimal schedules. The approach can be summarized as follows [J15<sup>5</sup>]: First, the iteration space of a nested loop program is tiled symbolically into parametric orthotopes.<sup>5</sup> Next, the resulting tiled program is also scheduled symbolically, resulting in a set of latency-optimal parametrized schedule candidates. Finally, at runtime, once the allocated PE region (size) becomes known, simple comparisons of latency-determining expressions finally steer which of these schedules will be selected and the corresponding program configuration executed on the corresponding processor array.

<sup>5</sup>An *orthotope* is a parallelotope whose edges are all mutually orthogonal to each other. The orthotope is a generalization of the rectangle and rectangular parallelepiped.

After partitioning and modulo scheduling [T1], instruction and register binding are performed before code is generated. Our code generation approach [P46, J17] starts with subdividing the set of PEs of a TCPA into so-called *processor classes*, such that all processors that belong to the same class obtain the same program but may differ only by a delayed start of execution. For each processor class a corresponding program block control flow graph is traversed to emit the final assembly code. Unique to our approach is that local control is represented by the code generated for each processor class, whereas global control is implemented by a global controller [J18, J17], which produces global control signals that trigger the program executions and branching. As a result of this, the entire static control flow (i.e., zero-overhead loops and static branching) can be hidden.

### 2.3.3 Results

In [J18], we could demonstrate a greatly superior energy efficiency of TCPAs in comparison with an embedded GPU (ARM Mali-T604), mainly due to a much better resource utilization, data locality, and less expensive functional units (integer arithmetic). However, we envision TCPAs complementary to GPUs in an MPSoC since TCPAs are limited to integer arithmetic so far. Whereas, GPUs are high-performance devices, which are well suited for 3D imaging when floating-point calculations are needed.

Our highly configurable and flexible simulation approach based on C++ templates is about five times faster when simulating a VLIW PE of a TCPA compared to implementations not using templates.

We could push the research on loop parallelization by providing symbolic parallelization techniques according to the above outlined multi-step approach. Previous works [121, 54, 55, 9, 122] partition only the iteration space symbolically but not the data dependences of loop-body statements and also only consider either merely sequential or atomic execution of the tiles.<sup>6</sup> In contrast, our approach allows for communicating results as soon as they are computed, which is especially beneficial for streaming architectures, such as TCPAs. We applied our novel formalism to schedule and assign *atomic iterations*<sup>7</sup> to processors symbolically for a given loop nest with uniform data dependences to the most common loop partitioning techniques: (1) outer loop parallelization [P48, J15] (a.k.a. clustering, blocking [156], or Locally Sequential, Globally Parallel (LSGP) [60]), (2) inner loop parallelization [P32] (a.k.a. tiling or Locally Parallel, Globally Sequential (LPGS) [60, 104]), and (3) hierarchical loop parallelization [P19,

---

<sup>6</sup>*Atomic execution of a tile* denotes that all iterations belonging to that tile are computed first before other tiles depending on these computations might be executed.—This may lead to cyclic dependences between adjacent tiles, and thus unschedulability, Whereas, a finer-grained execution and communication of results per iteration might be schedulable.

<sup>7</sup>*Atomic iteration* denotes that a single iteration of a given loop nest is executed in one unit of time (e.g., clock cycle).

J18]. Moreover, we generalized our symbolic approach to modulo scheduling [P15], including multi-cycle instructions under resource constraints (i.e., a limited number of Functional Units (FUs)). Here, we formally and experimentally show that if the number of processor elements to map onto is known at compile time, the resulting schedules are latency optimal. Otherwise, they are negligibly suboptimal. Summing-up, symbolic loop parallelization techniques complement resource-aware computing on TCPAs perfectly.

Our TCPA code generator produces highly competitive code [J17]. Compared with the Trimaran compiler infrastructure [27, 144], the generated codes of our approach achieve 56–88% higher throughputs. The code generator even more drastically outperforms the VEX compiler framework [38, 59]. Here, the generated code runs 4–12 times faster, although the code size is 2–4 times smaller.

### 2.3.4 Key Papers

In the following, I briefly classify the role of the key papers related to resource-aware computing, which are part of this cumulative habilitation treatise. Reprints of these papers are available in Appendix C.

**ACM TECS '14**      Hannig, Lari, Boppu, Tanase, and Reiche. “Invasive tightly-coupled processor arrays: A domain-specific architecture/compiler co-design approach”      [J18]

page 109ff.

The ACM TECS article summarizes our research on invasive TCPAs, a class of programmable accelerators with built-in hardware for very fast resource management. The article covers the entire technology stack, i.e., architectural design, programming, compilation, and simulation. By considering the design as a whole—that is, both architecture and compiler design—we could demonstrate a substantial performance and gains in energy efficiency in comparison with a state-of-the-art embedded GPU. Preserving application knowledge and combining it with architectural support, such as zero-overhead looping enabled us to generate code that is as fast as fully unrolled (across all loop levels) code, while the code size is kept minimal.


**RSP '17**      Witterauf, Hannig, and Teich. “Constructing fast and cycle-accurate simulators for configurable accelerators using C++ templates”      [P5]

page 139ff.


The RSP '17 paper deals with constructing fast and cycle-accurate simulators for processors and processor arrays, such as TCPAs. The entire architecture model is specified using C++ templates. All parameters of a hardware component are considered as C++ type and construct a class templated on this type. Eventually, the simulator is built as a hierarchical composition of the component classes. The proposed ap-




proach is evaluated by modeling and simulating a VLIW PE of a TCPA, and comparing it against a non-templated simulator.

**Springer JSPS '14** Teich, Tanase, and Hannig. "Symbolic mapping of loop programs onto processor [J15]   
page 147ff. arrays"

This JSPS '14 article is a rigorous extension of our award-winning paper [P48] "Symbolic Parallelization of Loop Programs for Massively Parallel Processor Arrays" presented at ASAP '13. It presents a method for the symbolic parallelization of nested loop programs with uniform data dependences using LSGP partitioning. Parametrized linear functions are used for assigning and scheduling iterations of such loop programs on processor arrays of unknown size. Latency-optimal schedules may be derived statically using two program transformations: First, the iteration space is partitioned symbolically using parametrized tile sizes. From this symbolically tiled code, latency-optimal symbolic schedules are determined. Further, an upper bound for the number of different optimal schedules as well as a pruning algorithm to reduce the search space efficiently is presented.

**MEMOCODE '14** Tanase, Witterauf, Teich, and Hannig. "Symbolic inner loop parallelisation for [P32]   
page 177ff. massively parallel processor arrays"

The MEMOCODE '14 paper presents a first solution to the unsolved problem of symbolically scheduling a given symbolically tiled loop nest according to the LPGS partitioning method. A mixed compile-/runtime approach is proposed. First, a unique intra-tile schedule is determined, which is responsible for scheduling the iteration points within one tile. Subsequently, all feasible inter-tile schedule candidates are determined based on sequential scanning orders given by stride matrices. The result is a parametrized latency formula that is used in a compiler-generated prolog to select the latency-minimal schedule candidate at runtime, once the size of the available processor array becomes known. The prolog also ensures feasibility of the schedules by evaluating simple guard expressions and repairing the schedule vector candidates accordingly.

**ACM TECS '17** Tanase, Witterauf, Teich, and Hannig. "Symbolic multi-level loop mapping of [J1]   
page 187ff. loop programs for massively parallel processor arrays"

LSGP partitioning is well-suited for tuning the I/O demand of a tile to the given bandwidth. However, this concept cannot be used to obtain a mapping independent of the iteration space size of the algorithm.

Whereas, LPGS partitioning may handle constant as well as minimal local memory requirements on an unknown number of processors at compile time. But, the required I/O bandwidth is more significant than in the case of LSGP and might exceed the existent I/O capacities. To remedy this dichotomy, the ACM TECS article combines the best of both worlds by proposing a symbolic multi-level tiling/scheduling approach that balances the I/O bandwidth and local memory requirements.

**ASAP '16**  
page 215ff.

Witterauf, Tanase, Hannig, and Teich. "Modulo scheduling of symbolically tiled loops for tightly coupled processor arrays" [P15]

The ASAP '16 paper combines symbolic tiling techniques with resource-constrained modulo scheduling [119, 118, 146, T1] to parallelize nested loop programs not only at loop level but also at instruction level (i.e., Instruction-Level Parallelism (ILP)). For this, the dependence constraints are partitioned into a parametric and non-parametric subset. A solution to the modulo scheduling problem can be found using only the non-parametric constraints. To still satisfy the parametric dependence constraints, a minimum tile size is determined from the found solution. If the minimum tile size is not satisfied at runtime, a fallback schedule is selected alternately.

**Springer JSPS '14**  
page 225ff.

Boppu, Hannig, and Teich. "Compact code generation for tightly-coupled processor arrays" [J17]

This JSPS '14 article is a thoroughgoing extended version of the ASAP '13 paper [P46]. It presents a design methodology for the mapping of nested loop programs onto TCPAs with an emphasis on code compaction and code generation. TCPAs and the corresponding code generator support zero-overhead looping not only for innermost loops but also for arbitrarily nested loops, i.e., the generated code is as fast as the equivalent entirely unrolled (across all loop levels) code while having a minimal code size. For selected benchmarks, our code generator is assessed and compared to the well-known Trimaran [27, 144] and VEX [38, 59] compiler frameworks.

The research on invasive TCPA architectures was mainly conducted with Vahid Lari during his Ph.D. [82, 83], helped by Jürgen Teich. Already, in my Ph.D. [T1], I investigated the problem of resource-constrained loop program scheduling for TCPAs. Consequently, this formed the basis for the research and development on TCPA code generation that was conducted with Srinivas Boppu during his Ph.D. [19]. Similarly, fundamentals such as the theoretic number of possible schedule candidates as well as

the concept of *path strides* arose out of my Ph.D. [T1] and were used as fundamentals in the research on symbolic parallelization of nested loop programs that was carried out together with Alexandru Tanase, Jürgen Teich, and Michael Witterauf. During his Ph.D. [137], Alexandru Tanase contributed principally to the symbolic LSGP, LPGS, and hierarchical partitioning methods, as well as symbolic scheduling techniques. The work on symbolic modulo scheduling and TCPA simulation was undertaken with doctoral researcher Michael Witterauf.



### 3 Domain-specific Computing

As introduced and motivated in Chapter 1, processor systems not only contain more and more cores but also specialized cores of different types due to performance and energy efficiency reasons. However, programming such parallel, heterogeneous systems is not an easy task since there exist too many different programming models and accordingly languages [95, 96]. Most of them require in-depth knowledge of both the application and target architecture, which is commonly referred to as the *programmability gap*. Naturally, the question arises how this gap can be closed and whether there is a one-fits-all solution, sort of a Jack or Jill of all trades of parallel programming languages?

On closer examination, programming languages might be rated with respect to three properties: (1) *performance*, (2) *generality*, and (3) *productivity*. At this point, we do not quantify these properties but rather rate them according to their general perception as follows:

**Performance** denotes the run time of a program executed on a certain architecture.

That is, which performance can be achieved using a certain programming language and how it is processed, i.e., whether it is compiled or interpreted? A compiler analyses an entire program and typically generates machine code that is directly executable by a processor, whereas an interpreter takes only a single instruction as input at a time and execute that instruction. An approach based on compilation is thus typically faster than one based on interpretation.

**Generality**, sometimes also denoted as *expressiveness*, denotes how rich the vocabulary and how powerful the syntax of a language is with respect to its versatility. Put differently, is it a *general-purpose* programming language applicable for a broad range of applications or is the language specialized for a certain application area? Even very specialized and restricted programming languages might be Turing complete<sup>8</sup>—however, it is questionable if it makes sense to spend great programming effort in order to describe an application of a different domain, and it goes along with the next property (productivity).

**Productivity** denotes the required programming effort to obtain a functionally correct, executable implementation. It corresponds to the variety of available high-level language constructs and programming abstractions. That is, productivity does not only include the development time or number of Lines of Code (LoC) but also the

---

<sup>8</sup> *Turing completeness* or *computational universality* denotes in the context of a programming language that the language can be used to simulate any single-taped Turing machine.

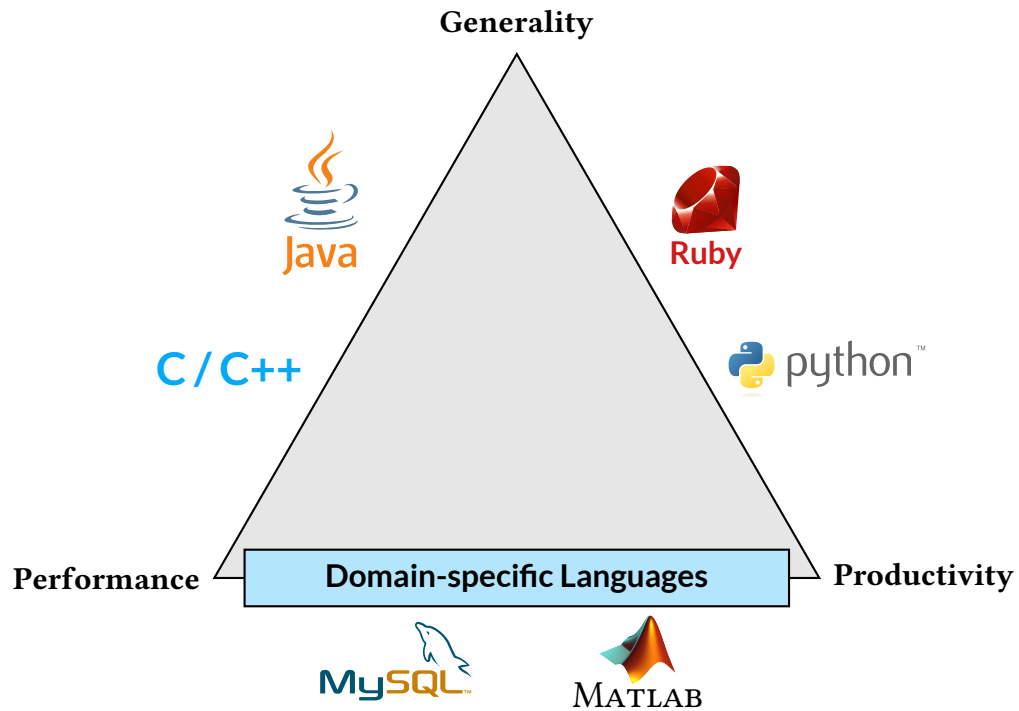


Figure 3.1: Triangle of the programming language landscape—a compromise between performance, generality, and productivity. A few popular programming languages are classified exemplarily.

time spent for finding errors in a program. Obviously, the probability for errors, and thus the debugging effort, might increase with the length of a program.

The three aforementioned properties span a triangle of the programming language landscape. It is visualized in Figure 3.1 and a few well-known programming languages are classified.

Next, I give a brief introduction to domain-specific languages, followed by three domain-specific approaches that I have investigated in the context of this habilitation treatise.

## 3.1 Domain-specific Languages

Already in the 1960s, researchers, such as Mark I. Halpern [50], James Martin [93], and Jean E. Sammet [126], discussed the desire for *machine independence*—in the context of writing computer programs in machine or assembly languages accompanied by portability issues (e.g., number formats and precision, integration of storage and I/O devices)—as well as *problem orientation* of programming languages.

In 1967, James Martin wrote in his textbook *Design of Real-Time Computer Systems* in the context of man-machine interfaces about the future of programming languages:

“We must develop languages that the scientist, the architect, the teacher, and the layman can use without being computer experts. The language for each user must be as *natural* as possible to him. The statistician must talk to his terminal in the language of statistics. The civil engineer must use the language of civil engineering. When a man learns his profession he must learn the *problem-oriented languages* to go with that profession.

If we give teachers throughout the world the right computer language for them, they will build *libraries* of teaching programs. If we give circuit designers the right language for them, they will make computers design circuits. If we give the medical profession the right language, they will give a doctor far more information at his fingertips than one individual could ever have today.” ([93, pp. 89f.])

Shortly afterward, Jean E. Sammet wrote in her textbook *Programming Languages: History and Fundamentals*:

“The term *problem-oriented* ... any language which is easier for writing solutions to a particular problem ... ” ([126, p. 21])

From these postulations, the idea follows to take advantage of the *knowledge* being inherent in a particular problem area or field of application, i.e., a certain *domain*, in a well-directed manner and thus, to master the complexity of the before-mentioned heterogeneous systems. Such *domain knowledge* can be captured by reasonable abstractions, augmentations, and notations, e.g., libraries, *Domain-Specific programming Languages (DSLs)*, or combinations of both (e.g., embedded DSLs implemented via template metaprogramming [131, 148]). On this basis, patterns can be utilized to transform and optimize the input description in a goal-oriented way during compilation, and, finally, to generate code for a specific target architecture. Thus, DSLs have a high productivity plus typically also a high performance (see Figure 3.1).

While more useful than ever before, DSLs are not new in the programming language landscape—and probably already came across any of us. Widely-used examples include query languages for accessing and manipulating relational databases (e.g., Structured Query Language (SQL) [31]), application programming interfaces for drawing 2D and 3D graphics like OpenGL [132], languages such as MATLAB [105, 53] for doing math,<sup>9</sup> hardware description languages like VHDL [7], or languages for the preparation of text documents, such as L<sup>A</sup>T<sub>E</sub>X [81] with which this treatise was typeset. But also away

---

<sup>9</sup>The association of MATLAB with a *math* language seems naive. More precisely, it started as an *array programming language* in the late 1970s and evolved since then to a multi-paradigm language (incorporating concepts of array, functional, imperative, procedural, and object-oriented programming).

from engineering and computer sciences, there have been applications in other areas, e.g., pig farming [99], finances [114, 40], or landscape ecology [42]. In addition to the mentioned increased program development productivity, one can infer another valuable DSLs property from the above languages, namely to ease the communication with domain experts because a well-designed DSL offers the context and thus serves as communication platform between the domain expert (i.e., user) and the programmer (i.e., technology expert) [94, 39].

#### 3.1.1 Definition

In literature, several definitions have been proposed for the term DSL. For example, Arie van Deursen et al. propose the following definition:

“A *domain-specific language* (DSL) is a programming language or executable specification language that offers, through appropriate notations and abstractions, expressive power focused on, and usually restricted to, a particular problem domain.” ([149, p. 1])

Marjan Mernik et al. write in their survey:

“Domain-specific languages (DSLs) are languages tailored to a specific application domain. They offer substantial gains in expressiveness and ease of use compared with general-purpose programming languages in their domain of application.” ([100, p. 1])

Martin Fowler proposes in his textbook *Domain-Specific Languages* the following definition:

“Domain-specific language (noun): a computer programming language of limited expressiveness focused on particular domain.” ([39, p. 27])

All three definitions above have in common that they refer to programming languages tailored to a certain problem domain, and are of limited/restricted expressiveness. Typically, DSLs adhere to the following properties.

- A programming language for a *particular field of application* or *domain*,
- Offers appropriate *abstractions* and *notations*,
- Typically, *small* (i.e., restricted in number of notations and abstractions), does not necessarily have to be Turing complete,
- *Limited expressiveness*, i.e., less expressive than a general-purpose programming language,



- Rather *declarative* than imperative,
- *Nature of a programming language*, i.e., expressiveness should not be limited by the number of individual expressions but rather by how these expressions can be combined logically.

There are many synonyms for the term DSLs in the literature, they have also been called: *Specification languages* [29] in the context of application generators, *little languages* [12], *micro-languages*, *minilanguages* in the Unix world [120, pp. 183ff.], *task-specific programming languages* [109, p. 27, 75, 72], *Very High-Level programming Languages (VHLLs)* that were used often for rapid software prototyping and scripting<sup>10</sup> [24, 154], *special purpose languages* [153, p. xix], or *languages for specialized application areas* [153, p. xix, 13, p. 17], for instance in the context of the APT programming language [123] for programming CNC (Computer Numerical Control) machines.

### 3.1.2 Classification of DSLs

One distinctive feature of a DSL is whether it is *textual* or *graphical*. A textual DSL contains typical constructs of textual programming languages (expressions, operators, etc.), while geometric forms (such as lines, arrows, and shapes) are used in a graphical DSL in order to express intent. Examples of graphical DSLs [47, C2] include the Unified Modeling Language (UML) [97, 18], LabVIEW [3], MATLAB Simulink [16], as well as visual programming languages, such as Scratch [90] or Google’s Blockly [41] for educational purposes of pupils.

In the following, I focus on textual DSLs even though the next attribute is also applicable to graphical DSLs. According to this property, DSLs can be mainly grouped into two categories, namely *internal* and *external* ones.

An *external DSL* denotes a programming language defined entirely new. That is, its syntax<sup>11</sup> and semantics<sup>12</sup> can be freely defined—however, often it is inspired by existing programming languages. The effort for designing an external DSL is relatively high but there exist tools that aid the development process (e.g., parser generators [86, 61, 101, 2, 112], metaprogramming [65, 67, 57, 117] and transformation frameworks [64, 36]). Well-known examples of external DSLs are SQL [31], awk, and regular expressions.

An *internal DSL* uses in some manner a general-purpose programming language, referred to as *host language*. The DSL utilizes the same syntax as its host language, i.e., it inherits the generic language elements of the host language (conditionals, functions, loops, etc.). Thus, the existing compiler and interpreter of the host language can be

---

<sup>10</sup>Nowadays, scripting languages, such as Perl, Python, and Ruby, are simply denoted as *high-level programming languages*.

<sup>11</sup>Syntax denotes the structure/grammar of a programming language.

<sup>12</sup>The semantics of a programming language is about the *meaning*, i.e., there could be syntactically well formed programs but that are not semantically defined (e.g., using uninitialized variables).

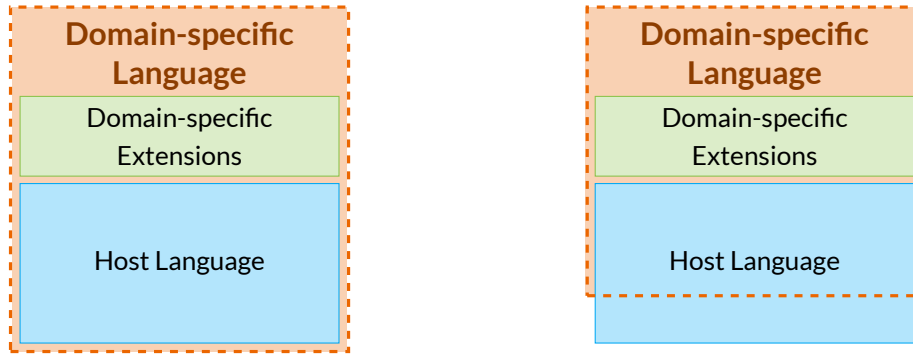


Figure 3.2: Common forms of internal DSLs. *Extension* (on the left side of the figure): The entire set of language features of a host language can be utilized and is augmented by domain-specific extensions. *Reduction* (shown on the right side): Here, also the host language is extended by domain-specific language constructs. However, only a certain subset of the host language’s elements can be used, i.e., the it is restricted.

employed for transforming and executing a DSL program, respectively. That is, the DSL is *embedded* into another language, therefore, the terms internal DSL and *embedded DSL* are interchangeable. An internal DSL may solely provide *extensions* to the employed host language, i.e., the entire host language with all its features is made available to the user. While such an extension is compelling (see Figure 3.2 left)—an acquired general-purpose language can be entirely used—it carries the inherent danger of expressing too many program parts by constructs that might be difficult to analyze by a compiler, or the generated target code might be inefficient. Internal DSLs, therefore, often hide constructs of the host language that are not relevant to the domain or difficult to analyze by a compiler. For instance, if C is considered as host language, often an embedded DSL forbids constructs such as pointers or recursions. This technique is called *reduction* and is illustrated in Figure 3.2 on the right. Finally, the DSL ends up being a restricted host language blended with new domain-specific augmentations. Common extension techniques to express domain knowledge in an internal DSL are libraries (data types, methods), annotations, or macros.

In comparison, external DSLs are typically more flexible and expressive than internal DSLs, however, at higher implementation cost. The richer expressiveness of external DSLs is not only reflected at the language level but also offers opportunities for domain-specific Intermediate Representations (IRs)—far beyond Abstract Syntax Trees (ASTs) as commonly used in compilers for general-purpose languages. Thus, powerful semantic models can be designed, e.g., a hierarchy of objects in the case of a principally declarative DSL, or finite state machines in the case of a DSL for automation technology.

## 3.2 Domain-specific High-level Synthesis

Research in the areas of loop parallelization (see also Section 2.3) and synthesis of nested loop programs to dedicated hardware accelerators is deeply rooted at the Chair of Hardware/Software Co-Design and also in my scientific career. In this course of research, we have developed PARO [P96], which is a High-Level Synthesis (HLS) tool for a particular class of loop algorithms, based on the mathematical foundation of the *polyhedron model* [37]. The development of PARO started<sup>13</sup> during my Ph.D. thesis [T1]. The considered loop programs belong basically to the class of *affine loop nests* [33], i.e., (1) all loop bounds and iteration-dependent control conditions must be expressible as an affine expression in the containing loop iteration variables, constants, or parameters (that is, fixed latest at loop entry), and (2) all array references (respective memory accesses) can be represented as affine functions in the loop iteration variables and fixed parameters. Furthermore, our considered algorithm class, the class of so-called *dynamic piecewise linear/regular algorithms* (DPLA/DPRA) [P118, T1], can also account for a specific type of dynamic data dependences. DPLAs can be seen as a descendant of the notion of *recurrence equations* introduced by Richard M. Karp et al. in the form of a *system of uniform recurrence equations* (SURE) [62] and a multitude of extensions thereof [116, 145, 158, 147, 143, 124, 34, 133], respectively. The class of DPLAs is implemented by the PAULA language [P97, T1] that serves as input to the PARO HLS tool [P96]. An abstract view of PARO's design flow is shown in Figure 3.3. Based on a given algorithm in PAULA notation, various source-to-source compiler transformations [108, 2] and optimizations can be applied using the design tool. Amongst others, these include *constant and variable propagation*, *common subexpression elimination*, *loop perfectization* [157], *dead code elimination*, *strength reduction of operators*, *(partial) unrolling of loops and reductions*, *affine transformations* [156] and *loop partitioning* based on the polyhedron model [37]. The heart of PARO is built on allocation and scheduling methods using mixed integer linear programming, as described in [T1]. Here, latency optimal schedules under resource constraints are derived before ultimately a dedicated hardware accelerator is synthesized.

PAULA [P97, T1] is not a DSL but a functional programming language, which is well suited for modeling iterative, multi-dimensional, data-flow dominated algorithms. In this sense, it is suitable for many application areas that can be expressed by systems of recurrence equations (linear algebra, digital signal processing, image processing,

<sup>13</sup>It should be mentioned that the terms *PARO* as a *design system* and *PARO methodology* are used considerably longer. A distinction from this older works follows: Marcus Bednara and Jürgen Teich [11, 10] present a design system, which is based on the CASPAR design system [138]. Around this system, a number of loosely-coupled tools for the parallelization of C code [15], scheduling and exploration [T2, P127], and hardware synthesis [10] were build. Scheduling is restricted to projection as global allocation. Also, the hardware generation is restricted to projection along a vector in the first direction (that means, the first iteration variable denotes the time axis).

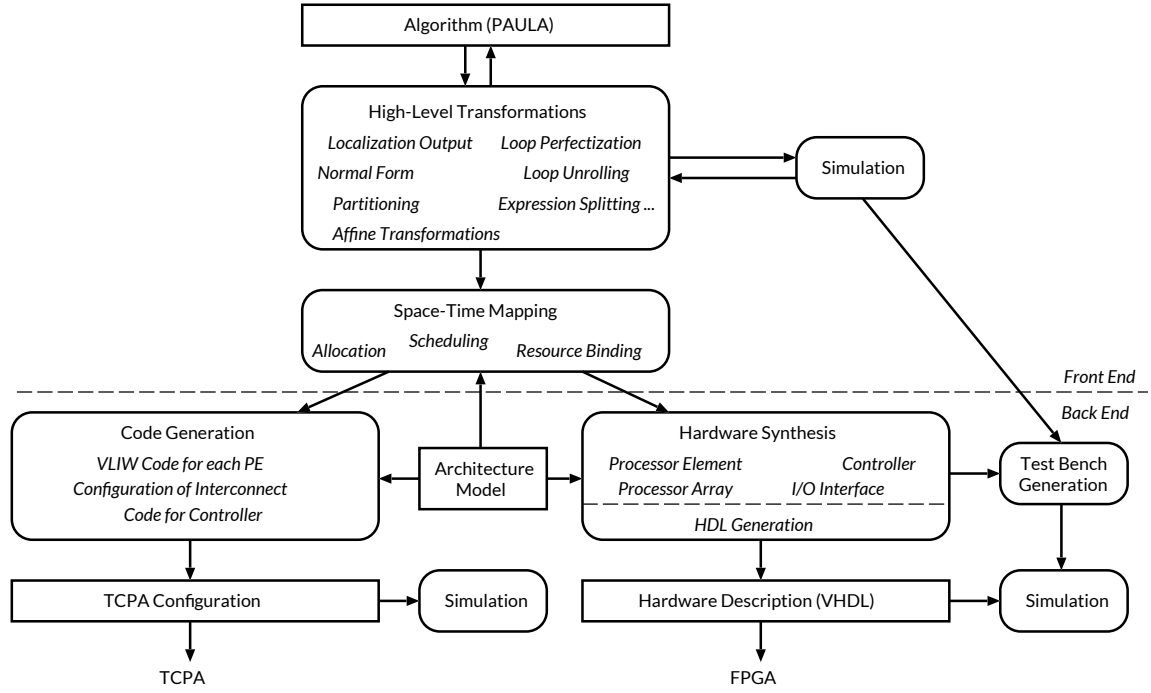


Figure 3.3: PARO design flow. Figure adapted from [T1, p. 151]. For more details on PARO, see [P96, T1]. Noteworthy, PARO’s front end also forms the basis for TCPA code generation [J18, J17] (back end on the lower left in the figure) as described in Section 2.3.

combinatorial problems, neural networks, etc.). Image processing was always a strong use case for PARO [T1, P74, P43]. Multi-dimensional reductions, such as summations, which are also handy when modeling two-dimensional convolutions in image processing, are one rationale for this. However, there were neither specific transformations in PARO nor language constructs in PAULA available for the domain of image processing. Similarly, albeit commercial HLS tools [C2] became recently very popular and versatile for FPGAs, they still require in-depth hardware design knowledge to obtain efficient implementations, and there exist hardly any domain-specific extensions. Exceptions are Xilinx Vivado HLS and the Intel/Altera FPGA SDK for Open Computing Language (OpenCL). The first offers a partial implementation of the Open Source Computer Vision (OpenCV) library for image processing and computer vision. The latter is per se—thanks to its OpenCL nature—suited for modeling image processing applications.

#### 3.2.1 Goals for Domain-specific HLS

In the context of domain-specific HLS, the primary goal is to analyze and provide programming abstractions to ease the specification of image processing applications.

Consequently, readability, error-proneness, and thus productivity can be improved significantly. The concepts should apply to both commercial tools, such as Xilinx Vivado HLS [C5], and academic frameworks, such as PARO [P96].

### 3.2.2 Approach

Other than image processing implementations in software following the control-driven von Neumann execution model [43], hardware designs have to be developed very differently to be efficient. Hardware designs should follow the dataflow-driven computing paradigm, where the entire ILP is explicitly represented, e.g., as a dataflow graph<sup>14</sup> [92, 102, 141]. Custom tailored hardware accelerators implemented as dataflow architecture [5, 113] can maximally benefit (within a space budget) from the width and height of a layered<sup>15</sup> data flow graph. Then, the width of a layer denotes the maximum number of operations that can be executed in parallel, i.e., the ILP. The height of the graph corresponds to the depth of a streaming pipeline of operations. In addition, the concept of streaming can be considered at the higher abstraction level of communicating tasks, represented again by a DAG, where nodes correspond to sub-algorithms (a.k.a. tasks, compute kernels), and edges denote data dependences (i.e., communication of data typically—but not necessarily—at higher granularity, e.g., entire frames in the case of image processing). GPU computing follows a buffer-wise execution concept [P31], where one compute kernel after the other is executed, and buffers serve as synchronization points (so-called host barriers) by reading from and writing to them sequentially. To achieve highly efficient FPGA implementations, the art here is to transform the buffer-wise execution model into a structural description suitable for streamed pipelining and to exploit the distributed on-chip storage capabilities (e.g., Block Random Access Memories (BRAMs), flip-flops) for data reuse in later iterations. In FPGAs, a widely used approach is *line buffering* [88, 87, P107], which is particularly beneficial for local operators in image processing, as each pixel is accessed more than once.

In PARO, thanks to the available loop transformations and parallelization techniques (e.g., partitioning) based on the polyhedron model as well as data flow analysis and modulo instruction scheduling [T1], the lifetime of input variables and corresponding line buffers are determined and generated automatically, respectively. In our previous work when using PARO in the image processing domain, a notable fraction of PAULA code was dedicated to image border treatment. For instance, in [P74], a 5-layered multiresolution filter for denoising X-ray images was designed. Here, in each filter

<sup>14</sup>A *dataflow graph* is a Directed Acyclic Graph (DAG). That is, in the context of data flow analysis, such as used in compiler design [108], only flow dependences exist. The nodes and edges of the DAG represent transformational actors (e.g., operations) and data dependences (i.e., the flow of data from one node to another one), respectively.

<sup>15</sup>A DAG is *layered* if its vertices can be arranged in horizontal rows, so-called layers, with the edges generally directed downwards [56].

kernel, about 300 lines of PAULA code, which correspond to half of the entire kernel code, were required to specify conditions at the image borders to realize *mirroring* [P74, P37] of pixels in the case of out-of-bound accesses.

As one contribution of [P37], we have introduced a new high-level transformation to facilitate border handling in high-level synthesis for FPGAs easily. Instead of writing dozens of code lines manually—which is potentially error-prone—just a single line specifying the input variable and type of border treatment can be used to produce the corresponding PAULA code automatically. Multi-dimensional reductions for median computations and sorting in general are another contribution. Both types of reductions are realized by generating throughput-optimized and highly parallel systolic arrays [79, 136].

In the case of Vivado HLS, a template-based approach is employed to specify memory components, such as line buffers and memory windows in dependence on the image width and size of local image operators [P33]. Moreover, architecture templates for loop coarsening and border handling are provided [P9].

#### 3.2.3 Results

In the case of PARO, we compared the required lines of PAULA code with the LoC of a functional equivalent implementation for C-based HLS. Here, we could demonstrate that the proposed domain-specific augmentations can increase productivity by one to two orders of magnitude [P37]. Furthermore, the generated implementations are in an equal range for Quality of Results (QoR) (i.e., FPGA resource utilization and clock frequency). The high productivity of our proposed domain-specific HLS approach was proven by utilizing it for the design of a novel algorithm for image impulse noise removal, which has a superior image quality compared to other state-of-the-art denoising methods [J2].

In the context of commercial C-based HLS, we designed a library to support the productive development of hardware accelerators for stream-based image processing applications and have shown that these accelerators can achieve a higher QoR at an equal coding effort than Xilinx OpenCV implementations [P33]. For the architecture templates (for loop coarsening in combination with border handling), which we also designed for Xilinx Vivado HLS, we could demonstrate an excellent scalability while I/O bandwidth and FPGA resources are available [P9]. Also, the synthesis results show that the proposed coarsening architecture uses 32% fewer registers for a 5-by-5 convolution with a coarsening factor of 64 compared to previous works, whereas the proposed border handling architectures facilitate a decrease in the Look-Up Table (LUT) usage by 36%.

### 3.2.4 Domain-specific HLS Key Papers

In the following, I briefly classify the role of the key papers related to domain-specific HLS, which are part of this cumulative habilitation treatise. Reprints of these papers are available in Appendix C.

- |   |   |
|---|---|
| <b>ASAP '14</b><br>page 251ff.          | Schmid, Tanase, Hannig, Teich, Bhadouria, and Ghoshal. "Domain-specific augmentations for high-level synthesis" [P37] <p>The paper introduces domain-specific augmentations to our PARO HLS tool [P96]. For the image processing domain, both a new high-level transformation to perform several types of border treatment (e.g., constant, clamp, mirror, mirror 101) is provided and domain-specific language extensions for sorting and median computations over polyhedral iteration domains are introduced to the PAULA language [P97]. The proposed approach is evaluated for several image processing algorithms and compared against C-based commercial HLS tool.</p>   |
| <b>FPL '14</b><br>page 257ff.           | Schmid, Apelt, Hannig, and Teich. "An image processing library for C-based high-level synthesis" [P33] <p>This paper presents a systematic approach for designing image processing accelerators in FPGA technology. For the domain of image processing filters, we developed concepts for (a) memory hierarchy, (b) causality and border handling, as well as (c) filter assembly (e.g., image pyramids). These concepts were implemented as a lightweight library to support productive design of FPGA accelerators for stream-based image processing using C-based HLS (e.g., Vivado HLS framework [C5] from Xilinx) and compared against state-of-the-art implementations in OpenCV [111] with respect to area cost, performance, and development effort (productivity).</p> |
| <b>Springer JSPS '17</b><br>page 261ff. | Bhadouria, Tanase, Schmid, Hannig, Teich, and Ghoshal. "A novel image impulse noise removal algorithm optimized for hardware accelerators" [J2] <p>The article in Springer's Journal of Signal Processing Systems underlines the powerfulness and productivity of the augmentations specific for image processing as introduced in our ASAP '14 paper [P37]. Here, we apply our approach to design a novel image impulse noise removal algorithm and synthesizing different FPGA implementations. The proposed noise removal algorithm relies on iteratively reducing</p>   |

the detection criteria, which refers to classifying a pixel as noisy pixel or noise-free pixel.

**ASAP '17**  
page 279ff.

Özkan, Reiche, Hannig, and Teich. "Hardware design and analysis of efficient loop coarsening and border handling for image processing" [P9]

The ASAP '17 paper presents two novel architectures for loop coarsening that use significantly less registers than previous work. Also, for the first time, we extended the problem of image border handling to loop coarsening. Depending on the input parameters (e.g., size of local operator, coarsening factor, border handling mode), we conducted a systematic analysis of latency and hardware costs. Based on these models, we provided an algorithm that selects the best coarsening and border handling architecture for the given parameters of a local operator. Finally, the paper is rounded off with implementation results obtained by Vivado HLS.

The work on domain-specific HLS was conducted with Moritz Schmid during his Ph.D. [128], helped by doctoral researcher Alexandru Tanase in the case of PARO and PAULA. This research was additionally spurred and complemented by Vivek Singh Bhadouria who contributed image processing knowledge during his four-month stay as visiting researcher in my group. Nicolas Apelt developed the library for stream-based image processing on top of Xilinx Vivado HLS during his master's thesis. The analysis and design of efficient loop coarsening and border handling architectures for the domain of image processing was mainly carried out together with doctoral researcher Akif Özkan, assisted by doctoral researcher Oliver Reiche.

### 3.3 HIPAcc: The Heterogeneous Image Processing Acceleration Framework

Modern medical image processing has come a long way since the first hazy "Röntgen-ray" radiograms as produced in 1895, yet X-rays are still one of the most important imaging techniques for a broad variety of applications. One exacting scenario is interventional angiography, which visualizes the inside of blood vessels (arteries, veins, etc.) and heart chambers by injecting a contrast agent and taking X-ray images. During an intervention, motion images in real time are required to guide catheters and minimal-invasive vessel treatments (stenosis, embolization, aneurysm,<sup>16</sup> etc.) interactively. The major problem still facing medical images is the poor Signal-to-Noise Ratio (SNR) due to limited dosage

---

<sup>16</sup>*Stenosis* denotes an abnormal narrowing of a blood vessel. An *aneurysm* is a balloon-like swelling of a vessel wall filled with blood. *Embolization* is a passage where a free mass travels through the bloodstream and may block a vessel ultimately.



and exposure for health reasons [80]. Therefore, the use of digital image processing algorithms to reduce the present noise along with preservation of visual structures is an essential field of research and just about to inhale the performance of available high-end systems. To meet the high arithmetic effort in combination with strict real-time requirements of such applications, sophisticated parallel implementations on multi-Digital Signal Processor (DSP) or FPGA systems have been used. The inhibition threshold to port an application to an entirely new class of target architectures, such as GPUs, is correspondingly high—and when is the appropriate time to make such a fundamental change of course? This tricky situation creates the desire to design algorithms only once in a domain-specific abstract way and to have compilers and generators that produce efficiently executable code for a broad variety of parallel architectures. And, in the case of new processors, *only* another compiler back end has to be added while the entire code base can remain untouched.

#### 3.3.1 HIPAcc Goals

The major goals of HIPAcc are to provide a DSL for image processing and a corresponding compiler framework for parallel processor architectures, including accelerators, such as GPUs and FPGAs. The DSL should offer means to specify image processing applications in an intuitive, expressive, and very compact manner to increase productivity (i.e., to reduce both development and debugging time). The DSL should not only be for the medical domain but also suitable for image processing and computer vision tasks, such as used in robotics or Advanced Driver Assistance Systems (ADASs). Further, the language should abstract completely from parallelization, low-level, and target-specific implementation details, hereby, algorithm development and implementation can be completely decoupled and existing DSL programs can be easily re-targeted to new processor architectures by *just* developing an appropriate compiler back end. Domain knowledge captured in the DSL as well as hardware knowledge of a target processor architecture should be utilized in the best possible way to generate highly efficient implementations.

#### 3.3.2 HIPAcc Approach

The design of HIPAcc's DSL started with a domain analysis, which includes a careful examination of the required algorithmic type of operations and operators as well as building blocks characteristic to the domain of digital image processing. A vast body of work for image processing algorithms and classifications exists in literature, e.g., [66, 20, 8, 125] to name only a few. John C. Russ et al. [125] classify image processing algorithms based on their different purposes, such as image acquisition, image correction, image enhancement, or measuring and filtering images in the frequency domain. Other authors [66, 20, 8] group image processing methods based on which input data (one

or multiple pixels,<sup>17</sup> the entire image or even multiple images) are used to produce an output (image). HIPAcc adopts the notion of Reinhard Klette et al. [66] to classify image processing in the spatial domain into *point operators* that compute one output pixel based on one input pixel (e.g., color-to-grayscale conversion, contrast enhancement), *local operators* that process also neighboring pixels to compute one output pixel (e.g., Gaussian blur for smoothing, Sobel filter for edge detection, median filter for noise removal, etc.), and *global operators* that consider the entire input (e.g., global reductions such determination of the minimum or maximum image intensity, image histogram calculation as representation of the tonal distribution). HIPAcc’s languages components were built upon this foundation and include objects for storing digital images and accessing them (by so-called *accessors*) or to operate on image pyramids (i.e., different resolutions of an image), declarative language constructs for boundary treatment (i.e., different modes for handling *out-of-bounds* accesses in the case of local operators), as well as interpolation modes in the case the computed and contributing images are of different size. Further, language elements, such as *masks* and *convolutions*, are offered to define point and local operators independent of the iteration space (i.e., size of an output image) as well as global reduction operators.

HIPAcc’s DSL is embedded into C++ and its components are implemented in form of C++ classes. Computations on images are as well encapsulated in C++ classes, which inherit from base classes provided by the framework. The decision for a C++-embedded DSL is twofold: (a) programs can be compiled with any C++ compiler, this allows to mix DSL programs with arbitrary C/C++ code as well as to port applications incrementally, and (b) no new compiler front end (parser, AST, etc.) has to be developed—HIPAcc utilizes Clang,<sup>18</sup> a C-based language family front end for the LLVM compiler [85].

Instead of directly generating machine code for each different target processor architecture, HIPAcc translates the DSL source program into parallel and optimized source code written again in a C-like (e.g., CUDA,<sup>19</sup> OpenCL, or OpenMP parallel C++, vector intrinsics) or other high-level programming language. This *source-to-source translation* is implemented within Clang, an overview of the HIPAcc framework is depicted in Figure 3.4. All analyses, optimizations, and target code generation are based on Clang’s IR [2, 108], the AST. The domain knowledge is captured in the nodes of the AST and further operations are triggered depending on their type, which correspond to (a) declarations and definitions of HIPAcc DSL classes, (b) statements that define objects in the DSL, and (c) expressions involving DSL objects [J9]. In addition to the domain knowledge, hardware knowledge of the target architectures is available, and thus, platform-specific optimization strategies can be applied. For instance, in the case of a certain GPU, a tailor-made adaptation of the generated code to the memory hierarchy.

---

<sup>17</sup>*Pixel*, a neologism for *picture element*.

<sup>18</sup><https://clang.llvm.org>

<sup>19</sup>Initially, CUDA was an acronym for *Compute Unified Device Architecture*, however, later Nvidia dropped its usage.

### 3.3. HIPAcc: The Heterogeneous Image Processing Acceleration Framework

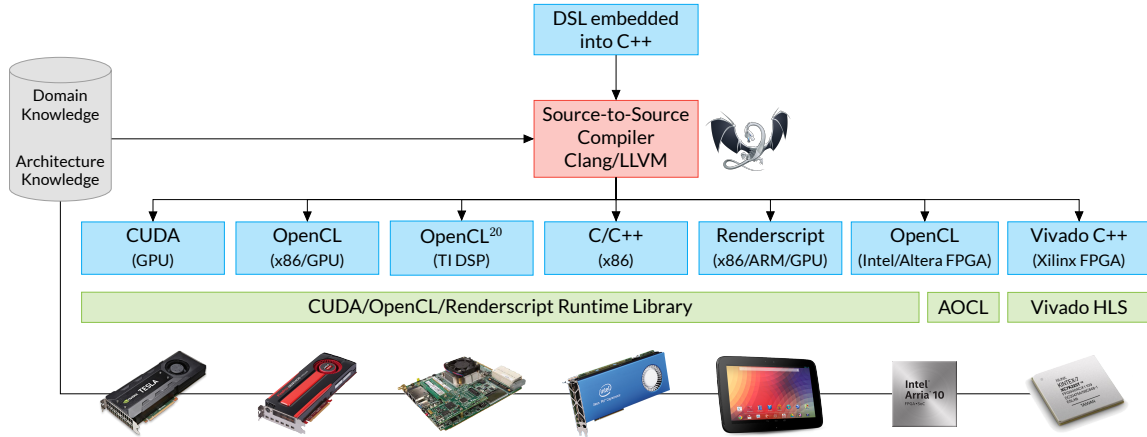


Figure 3.4: Overview of the HIPAcc framework and its target architectures.

Or, in the case of FPGAs as target architecture, the IR is transformed into a streaming pipeline (see Section 3.2.2) before HLS code (C++ tailored for Xilinx Vivado HLS or OpenCL for Intel/Altera FPGAs) is emitted.

#### 3.3.3 HIPAcc Results

We have applied domain-specific computing techniques to the area of image processing. The proposed techniques lead to significant advantages with respect to *productivity*, *portability*, and *performance*.

**Productivity:** HIPAcc’s DSL and source-to-source compilation framework can significantly increase productivity by one to two orders of magnitude [P61, P41, J12, J9] (e.g., quantified by LoC or other well-known software metrics [44, 51] when comparing DSL code with generated high-level code for a target architecture).

**Portability:** HIPAcc offers a wide range of code generators for different target architectures (see Figure 3.4), including manycore architectures, such as GPUs or Intel’s Xeon Phi [P61, P60, P56, J12, J9], Intel vector/SIMD instruction sets (up to SSE4.2 and AVX2) [P11], embedded GPUs, such as used in smartphones and tablet computers [P41, J9], and FPGAs [P31, P34, P26, P13, J5, P3]. Generating code for such a broad spectrum of architectures, starting from one and the same DSL program, is unrivaled. In addition, both the DSL and the source-to-source compiler are open-source,<sup>21</sup> and thus can be adapted by other researchers.

<sup>20</sup>HIPAcc’s OpenCL back end for DSPs from Texas Instruments (TI) was developed for an industrial cooperation partner to evaluate TI’s multicore DSP+ARM KeyStone II System-on-Chip (SoC) and the corresponding OpenCL compiler. This back end is not publicly available.

<sup>21</sup><http://hipacc-lang.org>

*Performance:* HIPAcc’s source-to-source compiler generates highly efficient parallel implementations. In the case of manycore architectures (i.e., high-end GPUs), we could demonstrate to beat hand-written low-level implementations (e.g., CUDA codes in the OpenCV library and in the Nvidia Performance Primitives (NPP) library<sup>22</sup>) [J12], [J9]. In the case of embedded GPUs, we demonstrated also that HIPAcc performs better than corresponding pre-implemented and hand-tuned versions [P41]. In the case of HIPAcc’s vectorization back end, we are on a par with other state-of-the-art compilers or even better for many benchmark algorithms [P11]. Regarding HIPAcc’s FPGA back ends, we could outperform Xilinx Vivado HLS OpenCV implementations [P31], and are close to those of hand-optimized Altera OpenCL examples [P13]. Beside pure performance numbers, we could also ascertain an excellent energy efficiency when comparing FPGA implementations against other parallel target architectures [J5].

#### 3.3.4 HIPAcc Key Papers

In the following, I briefly classify the role of the key papers related to HIPAcc, which are part of this habilitation treatise. Reprints of these papers are available in Appendix C.

**IEEE TPDS ’16**      Membarth, Reiche, Hannig, Teich, Körner, and Eckert. “HIPAcc: A domain-      [J9]

page 289ff.      specific language and compiler for image processing”

This article can be considered as the HIPAcc reference paper. It summarizes and unifies the core research results by providing a thorough description of HIPAcc’s language design and components, as well as of the compiler framework with an emphasis on kernel code generation and optimization for GPUs. Productivity gains are quantified using Halstead’s complexity measures [44, 51]. A variety of image processing applications is implemented in HIPAcc and for several manycore architectures, ranging from embedded to high-end GPUs from different vendors as well as Intel’s Xeon Phi, highly efficient code is generated. Finally, HIPAcc’s excellent performance results are underscored by comparing it against other state-of-the-art approaches (e.g., Halide [115], OpenCV [111]).

**DATE ’14**      Membarth, Reiche, Hannig, and Teich. “Code generation for embedded hetero-      [P41]

page 305ff.      geneous architectures on Android”

The DATE ’14 paper deals with code generation for embedded Heterogeneous System Architecture (HSA) platforms, such as MPSoCs as used in smartphones and tablet Personal Computers (PCs). Starting from an abstract high-level representation in HIPAcc, a code generator for

---

<sup>22</sup><https://developer.nvidia.com/npp>

both Renderscript and Filterscript on Android platforms is presented for the first time. With HSA, CPU and GPU share the same physical memory. This allows to avoid extensive memory transfers and enables the employment of heterogeneous resources where the same data has to be accessed frequently from different compute resources.

**CODES+ISSS '14**  
page 311ff.

Reiche, Schmid, Hannig, Membarth, and Teich. "Code generation from a domain-specific language for C-based HLS of hardware accelerators" [P31]

In the CODES+ISSS '14 paper, FPGAs are introduced as target to HIPAcc. To achieve this objective, we proposed a model transformation from the buffer-wise execution model as typically used in GPUs into a structural description in form of a streaming pipeline tailored for FPGAs. In addition, several FPGA-specific optimizations (e.g., conversion of floating to fixed point for mask coefficients, optimization of loop counter variables, mapping of vector data types) are proposed. Eventually, source code suited for C-based HLS is emitted instead of directly generating a structural description in a Hardware Description Language (HDL). The proposed approach is evaluated by assessing performance and power requirements of the generated FPGA designs in contrast to (embedded) GPUs.

**Elsevier JPDC '14**  
page 321ff.

Membarth, Reiche, Schmitt, Hannig, Teich, Stürmer, and Köstler. "Towards a performance-portable description of geometric multigrid algorithms using a domain-specific language" [J12]

This article extends HIPAcc's language range by facilitating multiscale modeling, i.e., image pyramids [25] and multiresolution approaches [80, P74] as used in image processing, but also known as two-dimensional geometric multigrid methods based on stencil computations [22, 49] in the domain of numerical analysis. Only image data for the finest pyramid level has to be provided, all other levels are managed automatically by the HIPAcc framework while the programmer can still specify how the pyramid is traversed and which operations are performed at different levels. That means, rather than just offering template objects, the nature of a programming language is preserved, and typical traversals for construction of pyramids in image processing as well as the V-cycle and W-cycle for multigrid stencil computations can be specified. Our proposed approach provides portability across different architectures and allows to achieve competitive performance compared to Halide [115] as well as hand-tuned implementations.

### 3. DOMAIN-SPECIFIC COMPUTING

---

**FPL '16**  
page 333ff.

Özkan, Reiche, Hannig, and Teich. "FPGA-based accelerator design from a domain-specific language" [P13]

Focus of the FPL '16 paper is Altera's Software Development Kit (SDK) for OpenCL in the context of image processing. First, it is shown that best programming practices and the data-parallel programming paradigm as used for GPUs are not well suited, and thus immense code modifications have to be applied in order to obtain efficient FPGA implementations. These findings are employed in the second major contribution of the paper, namely a HIPAcc back end for generating optimized OpenCL code specific to the peculiarities of Altera FPGAs. Implementation results for several image filters as well as a comparison of Altera's hand-optimized example designs with those generated by HIPAcc DSL programs of the same algorithms are provided and demonstrate that HIPAcc's generic approach can lead to results close to those of Altera. Furthermore, it is shown that server-grade GPUs can be outperformed in terms of throughput for a wide variety of image filter algorithms.

**Springer JSPS '17**  
page 343ff.

Reiche, Özkan, Hannig, Teich, and Schmid. "Loop parallelization techniques for FPGA accelerator synthesis" [J5]

This article summarizes, unifies, and extends FPGA-specific parallelization and optimization techniques as well as code generation for both Xilinx and Altera FPGAs. A generic method for loop tiling and *loop coarsening* [P20] as well as concrete back ends for generating C++ and OpenCL code are presented, which can be further synthesized with Xilinx Vivado HLS and Altera's SDK for OpenCL, respectively. A comparison of loop tiling and coarsening, in terms of hardware utilization and achieved throughput, for both HLS tools with varying parallelization factors is presented. Further, an evaluation of the throughput of HIPAcc-generated accelerators for Vivado HLS and Altera's SDK for OpenCL over an extensive application set is presented and compared against implementation results of embedded and discrete high-end GPUs.

**LCTES '17**  
page 369ff.

Reiche, Kobylko, Hannig, and Teich. "Auto-vectorization for image processing DSLs" [P11]

Based on the concept of whole-function vectorization [63], the LCTES '17 paper proposes auto-vectorization techniques for image processing DSLs, for the first time, in the context of source-to-source compilation. Thanks to the regular memory access patterns (i.e., relative indexing) of

the considered domain, the vectorization analysis can be significantly simplified. Another contribution is the handling of mixed bit-width data types. Here, an analysis that automatically selects the optimal Single Instruction, Multiple Data (SIMD) width for the specified target instruction set in order to pack native vectors into virtual vectors and to apply on-demand type promotion is proposed. Finally, the proposed techniques are implemented and integrated into HIPAcc, evaluated for several pre- and post-processing image filters, and compared against other state-of-the-art (semi-)automatically vectorizing compilers.

The work on the HIPAcc core framework, DSL definition, and code generation for (embedded) GPUs was carried out with Richard Membarth during his Ph.D. [98]. Doctoral researcher Oliver Reiche developed the Android integration, the model transformation and optimizations for efficient FPGA designs, while the work on HIPAcc's OpenCL back end for Intel/Altera FPGAs was conducted with doctoral researcher Akif Özkan. The research on domain-specific auto-vectorization was performed as well with Oliver Reiche, complemented by contributions from Christof Kobylko during his master's thesis. The research on image pyramids was a joined work that synergistically combines our findings on the automated solving of Partial Differential Equations (PDEs) as considered in project ExaStencils (together with doctoral researcher Christian Schmitt and project partner Harald Köstler) with HIPAcc's development efforts.

## 3.4 ExaStencils: Advanced Stencil-Code Engineering

Coming exascale computing<sup>23</sup> will require a close co-design of application, algorithm, and target-architecture-specific program development to unleash the tremendous performance of such supercomputers. However, before-mentioned HPC systems will only scale if the energy efficiency will considerably improve [69, 127]. One remedy for keeping up scalability are hardware components such as accelerators (e.g., GPUs, Intel Xeon Phi). Almost every fifth supercomputer in the TOP500<sup>24</sup> is already (as at June 2017) equipped with accelerator/co-processor technology. This trend will go on [69, 70, J3], and consequently, the node structure inside an exascale computer will become more heterogeneous, and thus challenging concerning programmability. Therefore, new software techniques and tools supporting both algorithm and architecture-aware program development will become vital, not only to ease application and program development, but also for performance analysis and tuning, to ensure short turn-around times, and for reasons of portability.

---

<sup>23</sup>Exascale computing refers to supercomputers that can carry out at least one exaFLOPS, i.e.,  $10^{18}$  Floating-point Operations Per Second (FLOPS).

<sup>24</sup><https://www.top500.org/>

Project “ExaStencils: Advanced Stencil-Code Engineering<sup>25</sup>” [R2, P35] tackles the challenges mentioned above for the important class of stencil computations, which are at the heart of many high-performance simulation applications in scientific computing. Examples are simulations in astrophysics [21], geophysics [4], oceanography [71], quantum chemistry [74], or the simulation of *viscoplastic materials* [78], i.e., suspensions of particles or macromolecules (e.g., foams, gels, pastes, bio-organic fluids such as blood, and food products such as fruit juices), as well as particle simulation in general [17]. Stencils are used to solve large systems of linear equations that may stem from the discretization of PDEs. They are regular computations on—usually multidimensional—structured or block-structured data grids, also known as multigrid methods [49], which involve a hierarchy of very fine to successively coarser grids.

#### 3.4.1 ExaStencils Goals

ExaStencils’ mission is to research and provide a unique, tool-assisted, co-design approach specific for the domain of multigrid methods based on stencil computations to reduce the foreseen performance/productivity gap of future exascale platforms. The project has three central objectives: (1) a substantial gain in productivity, (2) a high flexibility in the choices of the stencil algorithm and target platform, and (3) the provision and proof of scalability to reach exascale performance.

#### 3.4.2 ExaStencils Approach

Solving above-mentioned scientific problems requires the expertise of different specialists, including domain experts (i.e., natural scientists), mathematicians, software as well as hardware engineers. To best match the distinct interests and technical terminologies of these different groups, ExaStencils’ approach [P35] captures domain knowledge for each of these groups individually in the form of a hierarchy of tailor-made DSLs and corresponding compiler technology following the concept of stepwise refinement [155]. The *multi-layered* DSL is depicted in Figure 3.5, it is named ExaSlang [P29] and consists of four layers. Layer 1 is the most abstract layer, targeting natural scientists and engineers that have little or no experience in programming but want to define continuous mathematical problems in the form of an energy functional to be minimized or a partial differential equation to be solved, with a corresponding computational domain and boundary definitions. The second layer is slightly less abstract and allows to specify the problem in a discretized form. It is suitable for natural scientists and engineers with the appropriate knowledge as well as mathematicians. Based on the discretized

---

<sup>25</sup>Project ExaStencils is funded by the German Research Foundation (DFG) as part of the Priority Programme 1648 “Software for Exascale Computing”. The first funding period from January 2013 to December 2015 has been completed. The project is in its second funding period from January 2016 to December 2018.



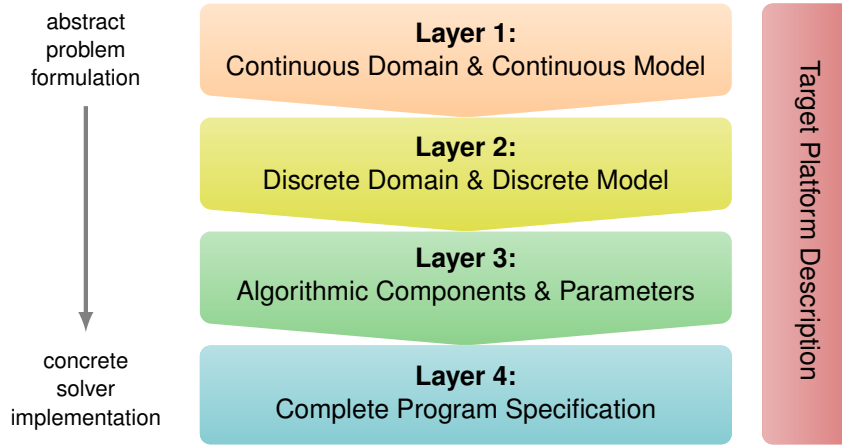


Figure 3.5: Multi-layered DSL approach of ExaSlang. Figure reprinted from [P29], p. 3].

problem of Layer 2, the third layer adds modeling of algorithmic components, settings, and parameter values. At this layer, the multigrid method becomes visible for the first time. Here, it is possible to define smoothers and to modify the multigrid cycle (e.g., different types of V-cycles or W-cycles [C1]). Computations are specified for the entire computational domain. Since this is already a very advanced layer regarding algorithm and discretization details, it targets mainly mathematicians and computer scientists. The fourth layer (ExaSlang 4) is most concrete, where user-relevant parts of the parallelization become accessible, data structures can be adapted for data exchange, and communication patterns can be specified. This layer can be considered as semi-explicitly parallel and is mainly intended for computer scientists.

While ExaSlang 4's syntax is partly inspired by the programming language Scala [110, P29], it and all other layers are external DSLs to offer highest expressiveness. Scala in turn is used as framework for DSL entry, transformation, optimization, and code generation. As a modern object-functional programming language [110], it offers powerful concepts, such as *parser combinators* and *pattern matching*. While the former allows the use of context-sensitive grammars, and thus, ease the specification of external DSLs, the latter offers possibilities to specify transformations shortly and expressively. The Target Platform Description Language (TPDL) is orthogonally available across all layers of the functional program description (i.e., ExaSlang 1 to ExaSlang 4). It specifies both hardware components of the target platform (e.g., CPUs, memory hierarchies, accelerators, cluster topology) and available software (e.g., compilers or Message Passing Interface (MPI) implementations).

On ExaSlang 4's IR, most of the compiler transformations are performed, i.e., parallelization efforts, such as domain partitioning, as well as high-level and low-level optimizations [J14], such as polyhedral optimizations [37, 76] and vectorization [77].

Since the order and parametrization of optimizations and compiler transformations are not always straightforward, modularity is paramount in the compiler framework to enable traceability and variant generation. Modularity and flexibility are realized by organizing transformations in strategies and transactions, and keeping track of the IR's state in a *StateManager* [P29], which allows to partly revert transformations in order to avoid re-generation and thus to speed up variant generation and exploration [J14, 46]. Afterward, the optimized IR is transformed to target source code, e.g., in C++. Finally, this generated code is transferred to the designated hardware platform to be compiled and executed.

#### 3.4.3 ExaStencils Results

As an essential result of having studied different modern frameworks and DSL technologies [P38], we decided to bank ExaStencils on the programming language Scala [110]. Right after, we quickly implemented a very limited prototype to turn ExaStencils' vision as a first proof-of-concept into reality [J16, P35, J6]. Another milestone of ExaStencils is the DSL design, especially the details on ExaSlang 4, presented in [P29]. Moreover, also in [P29], we presented our DSL-based optimization and transformation framework for generating code variants. Finally, we could prove a high productivity and scalability of our approach in [P29, C1]. Regarding productivity, we compared the LoC of ExaSlang 4 DSL code with the generated C++ code and obtained gains between a factor of 30 to 165. Concerning scalability, we could showcase in [P29] that our generated code scales up to the complete JUQUEEN cluster,<sup>26</sup> consisting of 28,672 nodes that correspond to a total of 458,752 cores.

#### 3.4.4 ExaStencils Key Papers

In the following, I briefly classify the role of the key papers related to ExaStencils, which are part of this cumulative habilitation treatise. Reprints of these papers are available in Appendix C.

ICCSA '14  
page 379ff.

Schmitt, Kuckuk, Köstler, Hannig, and Teich. "An evaluation of domain-specific language technologies for code generation" [P38]

The ICCSA '14 paper conducts a thorough evaluation of DSL technologies for code generation. It proposes several criteria for the assessment of frameworks for the design and compilation of textual DSLs. Four technologies are considered in the evaluation, namely the language workbench Spoofox/IMP [64], the metaprogramming language Rascal

---

<sup>26</sup>[http://www.fz-juelich.de/ias/jsc/EN/Expertise/Supercomputers/JUQUEEN/Configuration/Configuration\\_node.html](http://www.fz-juelich.de/ias/jsc/EN/Expertise/Supercomputers/JUQUEEN/Configuration/Configuration_node.html)

MPL [67], as well as two custom design approaches, one using C++ and the other Scala [110] as host programming language. This paper provides the rationale for selecting Scala as host language and technology for designing ExaStencils' multi-layered DSL and transformation framework, respectively.

**Euro-Par '14**  
page 389ff.

Lengauer, Apel, Bolten, Größlinger, Hannig, Köstler, Rüde, Teich, Grebhahn, Kronawitter, Kuckuk, Rittich, and Schmitt. "ExaStencils: Advanced stencil-code engineering" [P35]

The Euro-Par '14 presents thorough overview of project ExaStencils and can be considered as general reference. It introduces the considered application domain of stencil codes along with having a multi-layered domain-specific programming language and the corresponding concept of stepwise refinement [155] and model-driven software engineering [129]. In the ExaStencils workflow, at every refinement step, dedicated and highly automated optimizations are applied, which exploit the domain-specific knowledge available at this step. Since this paper summarizes the ideas at an early project stage, code generation is considered only as proof of concept with a preliminary prototypical implementation [J16, J6] in Scala [110].

**WOLFHPC '14**  
page 401ff.

Schmitt, Kuckuk, Hannig, Köstler, and Teich. "ExaSlang: A domain-specific language for highly scalable multigrid solvers" [P29]

The WOLFHPC '14 contribution is the key paper for ExaStencils multi-layered DSL technology. It summarizes the purpose of the four sorts of language layers (called ExaSlang 1 to ExaSlang 4), which are tailored for different target audiences, i.e., engineers and natural scientists, mathematicians, and computer scientists. The most concrete layer ExaSlang 4 follows the paradigm of procedural programming and is described in detail by presenting its language elements, such as simple (e.g., real, integer), aggregate (e.g., complex numbers, vectors) and algorithmic (e.g., fields, stencils) data types, level specifications to ease multigrid programming, control flow (e.g., if-statements and loops), functions, as well as communication. Furthermore, the structure and mechanisms of the transformation framework (compiler) that refine user input from one DSL layer to another, and finally emits C++ code, is presented. To enable both traceability and variant generation, modular concepts, such as transformations, strategies, and transactions, are introduced.

**Springer LNCSE '16** Schmitt, Kuckuk, Hannig, Teich, Köstler, Rüde, and Lengauer. "Systems of partial differential equations in ExaSlang" [C1] page 411ff.

The LNCSE '16 book chapter introduces additional data types such as vectors and matrices to ExaSlang 4, which further ease the description of solvers for systems of PDEs. Beside these new extensions at language level, the corresponding required modifications during the code generation are described. Finally, the paper is rounded off by an optical flow detection case study based on the multigrid approach [73]. Here, it is shown that the novel data types can reduce again ExaSlang 4 program sizes up to 28%. The productivity gains are up to a factor of 30 when the LoC of the generated C++ programs are set in proportion to LoC of ExaSlang 4 programs.

The work on ExaStencils was conducted with doctoral researcher Christian Schmitt. He significantly contributed to the evaluation of DSL technologies, the definition of ExaSlang, and to the transformation framework, helped by Sebastian Kuckuk, Harald Köstler, and Christian Lengauer.

## 4 Conclusions

This cumulative habilitation treatise compiles my research activities on how to tackle the design and programming complexity challenge of heterogeneous parallel systems. The presented design, simulation, parallelization, and compilation techniques pursue two strategies, namely resource-aware computing and domain-specific computing, which seem to be fundamentally different at first sight. Resource-aware computing provides a full control loop from hardware status information to the program level and back, whereas domain-specific computing drastically separates the concerns of algorithm development from parallelization and low-level implementation details. However, the two approaches have also commonalities: Both approaches scale very well for heterogeneous manycore systems and achieve high performance; in the case of invasive computing, through the implementation of program variants and symbolic mappings that can adapt to a certain number of resources at runtime. This adaptability gives programs the possibility to react flexibly to changes in the execution environment, such as the number of available resources, failures, power, or temperature. In the case of domain-specific computing, both domain knowledge and hardware knowledge are exploited to generate highly optimized implementations. In addition, an excellent productivity can be achieved thanks to highly abstract and predominantly declarative program specifications in the form of domain-specific augmentations (e.g., libraries or DSLs). But also in invasive computing, there has always been the motto: “As concrete as necessary, as abstract as possible.” That is, on the one hand, invasive computing provides instruments, such as resource-aware programming, that allow controlling a system in a very fine manner. On the other hand, once resource-aware programming and resource management strategies are adequately researched (e.g., by dint of system simulation), they can be moved progressively into compilation and runtime management, and that is exactly what happened in the second funding phase of invasive computing. Now, instead of specifying *how* many resources of which type should be claimed, the user can specify *what* should be achieved. For this purpose, she or he can specify requirements, e.g., in the case of image/video processing that the throughput should be at least 25 frames per second. But how to achieve this throughput requirement is completely up to the compiler and runtime system. For this purpose, design-time analysis/exploration of application mappings is combined with run-time management [151, 152]. By this transition from an imperative to a declarative programming concept, resource-aware computing approaches domain-specific computing. A second point of contact is the increasing investigation and determined exploitation of parallel patterns within invasive computing.

#### 4. CONCLUSIONS

---

Highly parallel accelerators in combination with domain-specific languages are more relevant than ever. For instance, Google's Pixel 2 smartphone has besides its high-end Qualcomm Snapdragon 835 MPSoC,<sup>27</sup> a domain-specific co-processor, the Pixel Visual Core that consists of 4096 tiny ALUs and will be programmable using the domain-specific library TensorFlow Lite<sup>28</sup> for machine intelligence (i.e., machine learning and deep neural networks) and the DSL Halide [115] for image processing applications. On the other extreme of computing, in data centers, accelerators are also increasingly investigated. Examples include interconnected FPGAs as computational accelerators in Microsoft's Project Catapult<sup>29</sup> [26], ASIC clouds [89], or even offered as commercial cloud services, e.g., Amazon EC2 Elastic GPUs<sup>30</sup> or Amazon EC2 F1 (FPGA) Instances.<sup>31</sup> Here, DSLs are appealing, especially, for non-hardware specialists [B3, C5] to unleash the full processing power of such accelerator-based large-scale systems.

What's next after Moore's law ends? Research in computing will not stop. Quite the opposite, new technologies and computational approaches, such as approximate computing [52, 103], neuromorphic computing [106], reversible and quantum computing<sup>32</sup> [150] will drive exciting scientific adventures and will foster research of corresponding computing paradigms, programming languages, and compilation techniques.

---

<sup>27</sup>The Snapdragon 835 has eight CPU cores, a GPU with 256 ALUs, several DSPs and other dedicated components.

<sup>28</sup><https://www.tensorflow.org/mobile/tflite/>

<sup>29</sup><https://www.microsoft.com/en-us/research/academic-program/project-catapult-academic-program/>

<sup>30</sup><https://aws.amazon.com/ec2/elastic-gpus/>

<sup>31</sup><https://aws.amazon.com/ec2/instance-types/f1/>

<sup>32</sup>LIQUi> <http://www.microsoft.com/en-us/research/publication/liqui-a-software-design-architecture-and-domain-specific-language-for-quantum-computing/>

# A Bibliography

## A.1 General Bibliography

- [1] Gul Abdulnabi Agha. *ACTORS: A Model of Concurrent Computation in Distributed Systems*. Tech. rep. AITR-844. MIT Artificial Intelligence Laboratory, June 1, 1985. HDL: 1721.1/6952.
- [2] Alfred V. Aho, Monica S. Lam, Ravi Sethi, and Jeffrey D. Ullman. *Compilers: Principles, Techniques, and Tools*. 2nd. Addison-Wesley, 2007. ISBN: 978-0-321-48681-3.
- [3] Hugo A. Andrade, Stephan Ahrends, and Simon Hogg. “Making FPGAs accessible with LabVIEW”. In: *FPGAs for Software Programmers*. Ed. by Dirk Koch, Frank Hannig, and Daniel Ziener. Springer, June 2016. Chap. 4, pp. 63–79. ISBN: 978-3-319-26406-6. DOI: 10.1007/978-3-319-26408-0\_4.
- [4] Mauricio Araya-Polo, Félix Rubio, Raúl de la Cruz, Mauricio Hanzich, José María Cela, and Daniele Paolo Scarpazza. “3D seismic imaging through reverse-time migration on homogeneous and heterogeneous multi-core processors”. In: *Scientific Programming* 17.1–2 (Jan. 2009), pp. 185–198. ISSN: 1058-9244. DOI: 10.1155/2009/382638.
- [5] Arvind and Stephen Brobst. “The evolution of dataflow architectures: From static dataflow to P-RISC”. In: *International Journal of High Speed Computing* 5.2 (June 1993), pp. 125–153. ISSN: 0129-0533. DOI: 10.1142/S0129053393000074.
- [6] Asen Asenov, Binjie Cheng, Xingsheng Wang, Andrew Robert Brown, Campbell Millar, Craig Alexander, Salvatore Maria Amoroso, Jente B. Kuang, and Sani R. Nassif. “Variability aware simulation based design-technology cooptimization (DTCO) flow in 14 nm FinFET/SRAM cooptimization”. In: *IEEE Transactions on Electron Devices* 62.6 (June 2015), pp. 1682–1690. ISSN: 0018-9383. DOI: 10.1109/TED.2014.2363117.
- [7] Peter J. Ashenden. *The Designer’s Guide to VHDL*. 3rd ed. Morgan Kaufmann, 2008. ISBN: 978-0-12-088785-9.
- [8] Isaac N. Bankman. *Handbook of Medical Image Processing and Analysis*. 2nd. Elsevier, 2009. ISBN: 978-0-12-373904-9. DOI: 10.1016/B978-0-12-373904-9.X0001-4.
- [9] Muthu Manikandan Baskaran, Albert Hartono, Sanket Tavarageri, Thomas Henretty, Jagannathan Ramanujam, and Ponnuswamy Sadayappan. “Parameterized tiling revisited”. In: *Proceedings of the 8th Annual IEEE/ACM International Symposium on Code Generation and Optimization (CGO)*. (Toronto, Ontario, Canada). Apr. 24–28, 2010, pp. 200–209. ISBN: 978-1-60558-635-9. DOI: 10.1145/1772954.1772983.
- [10] Marcus Bednara. “Design Automation for Massively Parallel Processor Arrays: Transforming Regular Algorithms to Reconfigurable Hardware”. Verlag Dr. Köster, Berlin, Germany. Dissertation. Friedrich-Alexander-Universität Erlangen-Nürnberg, Germany, 2004.
- [11] Marcus Bednara and Jürgen Teich. “Automatic synthesis of FPGA processor arrays from loop algorithms”. In: *The Journal of Supercomputing* 26.2 (2003), pp. 149–165. ISSN: 0920-8542. DOI: 10.1023/A:1024447517501.

## A. BIBLIOGRAPHY

---

- [12] Jon Bentley. “Programming pearls: Little languages”. In: *Communications of the ACM* 29.8 (Aug. 1986), pp. 711–721. ISSN: 0001-0782. DOI: 10.1145/6424.315691.
- [13] Thomas J. Bergin Jr. and Richard G. Gibson Jr., eds. *History of Programming Languages II*. ACM Press, 1996. ISBN: 978-0-201-89502-5.
- [14] Kerry Bernstein, David J. Frank, Anne E. Gattiker, Wilfried Haensch, Brian L. Ji, Sani R. Nassif, Edward J. Nowak, Dale J. Pearson, and Norman J. Rohrer. “High-performance CMOS variability in the 65-nm regime and beyond”. In: *IBM Journal of Research and Development* 50.4–5 (2006), pp. 433–450. DOI: 10.1147/rd.504.0433.
- [15] Oliver Beyer. “Implementierung eines Verfahrens zur Parallelisierung geschachtelter C-Schleifenprogramme”. Diploma Thesis. University of Paderborn, Department of Electrical Engineering and Information Technology, Computer Engineering Laboratory, Mar. 2002.
- [16] Robert H. Bishop. *Modern Control Systems Analysis and Design Using MATLAB and SIMULINK*. 1st. Boston, MA, USA: Addison-Wesley, 1996. ISBN: 0-201-49846-4.
- [17] Matthias Bolten. “Multigrid methods for long-range interactions”. In: *Fast Methods for Long-Range Interactions in Complex Systems*. Ed. by Godehard Sutmann, Paul Gibbon, and Thomas Lippert. Vol. 6. Schriften des Forschungszentrums Jülich: IAS Series. Forschungszentrum, Zentralbibliothek, 2011. ISBN: 978-3-89336-714-6.
- [18] Grady Booch, James Rumbaugh, and Ivar Jacobson. *Unified Modeling Language User Guide*. 2nd. Addison-Wesley, 2005. ISBN: 978-0-321-26797-9.
- [19] Srinivas Boppu. “Code Generation for Tightly Coupled Processor Arrays”. Dissertation. Friedrich-Alexander-Universität Erlangen-Nürnberg, Germany, Dec. 18, 2015. URN: urn:nbn:de:bvb:29-opus4-68678.
- [20] Alan C. Bovik, ed. *Handbook of Image and Video Processing (Communications, Networking and Multimedia)*. 2nd. Orlando, FL, USA: Academic Press, 2005. ISBN: 978-0-12-119792-6.
- [21] Axel Brandenburg. “Computational aspects of astrophysical MHD and turbulence”. In: *The Fluid Mechanics of Astrophysics and Geophysics*. Vol. 9. Advances in Nonlinear Dynamos. CRC Press, 2003. Chap. 9, pp. 269–344. ISBN: 978-0-415-28788-3. DOI: 10.1201/9780203493137.ch9.
- [22] Achi Brandt. “Multi-level adaptive solutions to boundary-value problems”. In: *Mathematics of Computation* 31.138 (Apr. 1977), pp. 333–390. ISSN: 0025-5718. DOI: 10.1090/S0025-5718-1977-0431719-X.
- [23] Kirk M. Bresniker, Sharad Singhal, and R. Stanley Williams. “Adapting to thrive in a new economy of memory abundance”. In: *Computer* 48.12 (Dec. 29, 2015), pp. 44–53. ISSN: 0018-9162. DOI: 10.1109/MC.2015.368.
- [24] Reinhard Budde, Karlheinz Kautz, Karin Kuhlenkamp, and Heinz Züllighoven. “Very high level languages”. In: *Prototyping: An Approach to Evolutionary System Development*. Springer, 1992, pp. 131–143. ISBN: 978-3-642-76820-0. DOI: 10.1007/978-3-642-76820-0\_11.
- [25] Peter J. Burt and Edward H. Adelson. “The Laplacian pyramid as a compact image code”. In: *IEEE Transactions on Communications* 31.4 (Apr. 1983), pp. 532–540. ISSN: 0090-6778. DOI: 10.1109/TCOM.1983.1095851.
- [26] Adrian M. Caulfield, Eric S. Chung, Andrew Putnam, Hari Angepat, Daniel Firestone, Jeremy Fowers, Michael Haselman, Stephen Heil, Matt Humphrey, Puneet Kaur, Joo-Young Kim, Daniel Lo, Todd Massengill, Kalin Ovtcharov, Michael Papamichael, Lisa Woods, Sitaram Lanka, Derek Chiou, and Doug Burger. “Configurable clouds”. In: *IEEE Micro* 37.3 (2017), pp. 52–61. DOI: 10.1109/MM.2017.51.



- [27] Lakshmi N. Chakrapani, John Gyllenhaal, Wen-mei W. Hwu, Scott A. Mahlke, Krishna V. Palem, and Rodric M. Rabbah. “Trimaran: An infrastructure for research in instruction-level parallelism”. In: *Languages and Compilers for High Performance Computing: 17th International Workshop, LCPC 2004, West Lafayette, IN, USA, September 22-24, 2004, Revised Selected Papers*. Ed. by Rudolf Eigenmann, Zhiyuan Li, and Samuel P. Midkiff. Vol. 3602. Lecture Notes in Computer Science (LNCS). Springer, 2005, pp. 32–41. ISBN: 978-3-540-31813-2. DOI: 10.1007/11532378\_4.
- [28] Philippe Charles, Christian Grothoff, Vijay Saraswat, Christopher Donawa, Allan Kielstra, Kemal Ebcioglu, Christoph von Praun, and Vivek Sarkar. “X10: An object-oriented approach to non-uniform cluster computing”. In: *Proceedings of the 20th Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA)*. (San Diego, CA, USA). ACM, Oct. 16–20, 2005, pp. 519–538. ISBN: 1-59593-031-0. DOI: 10.1145/1094811.1094852.
- [29] J. Craig Cleaveland. “Building application generators”. In: *IEEE Software* 5.4 (July 1988), pp. 25–33. ISSN: 0740-7459. DOI: 10.1109/52.17799.
- [30] Henk Corporaal. “TTAs: Missing the ILP complexity wall”. In: *Journal of Systems Architecture* 45.12–13 (June 1999), pp. 949–973. ISSN: 1383-7621. DOI: 10.1016/S1383-7621(98)00046-0.
- [31] C. J. Date and Hugh Darwen. *A Guide to the SQL Standard*. 4th. Addison-Wesley, 1997. ISBN: 0-201-96426-0.
- [32] Robert H. Dennard, Fritz H. Gaensslen, Hwa-Nien Yu, V. Leo Rideout, Ernest Bassous, and Andre R. LeBlanc. “Design of ion-implanted MOSFET’s with very small physical dimensions”. In: *IEEE Journal of Solid-State Circuits* 9.5 (Oct. 1974), pp. 256–268. ISSN: 0018-9200. DOI: 10.1109/JSSC.1974.1050511.
- [33] Michèle Dion and Yves Robert. “Mapping affine loop nests”. In: *Parallel Computing* 22.10 (Dec. 1996), pp. 1373–1397. ISSN: 0167-8191. DOI: 10.1016/S0167-8191(96)00049-X.
- [34] Uwe Eckhardt. “Algorithmus-Architektur-Codesign für den Entwurf digitaler Systeme mit eingebettetem Prozessorarray und Speicherhierarchie”. Dissertation. Technische Universität Dresden, Germany, June 2001.
- [35] Hadi Esmaeilzadeh, Emily R. Blem, Renée St. Amant, Karthikeyan Sankaralingam, and Doug Burger. “Dark silicon and the end of multicore scaling”. In: *IEEE Micro* 32.3 (2012), pp. 122–134. DOI: 10.1109/MM.2012.17.
- [36] Moritz Eysholdt and Heiko Behrens. “Xtext: Implement your language faster than the quick and dirty way”. In: *Proceedings of the ACM International Conference Companion on Object Oriented Programming Systems Languages and Applications Companion (OOPSLA)*. (Reno/Tahoe, NV, USA). ACM, Oct. 17–21, 2010, pp. 307–309. ISBN: 978-1-4503-0240-1. DOI: 10.1145/1869542.1869625.
- [37] Paul Feautrier and Christian Lengauer. “Polyhedron model”. In: *Encyclopedia of Parallel Computing*. Ed. by David Padua. Springer, 2011, pp. 1581–1592. ISBN: 978-0-387-09766-4. DOI: 10.1007/978-0-387-09766-4\_502.
- [38] Joseph A. Fisher, Paolo Faraboschi, and Cliff Young. *Embedded Computing – A VLIW approach to architecture, compilers, and tools*. Morgan Kaufmann, 2005. ISBN: 978-1-55860-766-8.
- [39] Martin Fowler. *Domain-Specific Languages*. 1st. Addison-Wesley Professional, 2010. ISBN: 978-0-321-71294-3.

## A. BIBLIOGRAPHY

---

- [40] Simon Frankau, Diomidis Spinellis, Nick Nassuphis, and Christoph Burgard. “Commercial uses: Going functional on exotic trades”. In: *Journal of Functional Programming* 19.1 (Jan. 2009), pp. 27–45. ISSN: 0956-7968. DOI: 10.1017/S0956796808007016.
- [41] Neil Fraser. *Google Blockly – A library for building visual programming editors*. <https://developers.google.com/blockly/>. 2017.
- [42] Cédric Gaucherel, Nathalie Giboire, Valérie Viaud, Thomas Houet, Jacques Baudry, and Françoise Burel. “A domain-specific language for patchy landscape modelling: The Brittany agricultural mosaic as a case study”. In: *Ecological Modelling* 194.1–3 (2006), pp. 233–243. ISSN: 0304-3800. DOI: 10.1016/j.ecolmodel.2005.10.026.
- [43] Michael D. Godfrey and David F. Hendry. “The computer as von Neumann planned it”. In: *IEEE Annals of the History of Computing* 15.1 (Jan. 1993), pp. 11–21. ISSN: 1058-6180. DOI: 10.1109/85.194088.
- [44] Ronald David Gordon and Maurice H. Howard Halstead. “An experiment comparing Fortran programming times with the software physics hypothesis”. In: *Proceedings of the National Computer Conference and Exposition (AFIPS)*. (New York, NY, USA). ACM. June 7–10, 1976, pp. 935–937. DOI: 10.1145/1499799.1499927.
- [45] Nathan Goulding-Hotta, Jack Sampson, Ganesh Venkatesh, Saturnino Garcia, Joe Auricchio, Po-Chao Huang, Manish Arora, Siddhartha Nath, Vikram Bhatt, Jonathan Babb, Steven Swanson, and Michael Bedford Taylor. “The GreenDroid mobile application processor: An architecture for silicon’s dark future”. In: *IEEE Micro* 31.2 (Mar.–Apr. 2011), pp. 86–95. ISSN: 0272-1732. DOI: 10.1109/MM.2011.18.
- [46] Alexander Grebhahn, Norbert Siegmund, Harald Köstler, and Sven Apel. “Performance prediction of multigrid-solver configurations”. In: *Software for Exascale Computing – SPPEXA 2013–2015*. Ed. by Hans-Joachim Bungartz, Philipp Neumann, and Wolfgang E. Nagel. Vol. 113. Lecture Notes in Computational Science and Engineering (LNCSE). Springer, Aug. 2016, pp. 69–88. ISBN: 978-3-319-40526-1. DOI: 10.1007/978-3-319-40528-5\_4.
- [47] John Grundy, John Hosking, Nianping Zhu, and Na Liu. “Generating domain-specific visual language editors from high-level tool specifications”. In: *Proceedings of the 21st IEEE/ACM International Conference on Automated Software Engineering (ASE)*. (Tokyo, Japan). Sept. 18–22, 2006, pp. 25–36. DOI: 10.1109/ASE.2006.39.
- [48] Suyash Gupta and V. Krishna Nandivada. “IMSuite: A benchmark suite for simulating distributed algorithms”. In: *The Computing Research Repository (CoRR)* (2013). arXiv: 1310.2814 [cs.DC].
- [49] Wolfgang Hackbusch. *Multi-Grid Methods and Applications*. Vol. 4. Series in Computational Mathematics. Springer, 1985. ISBN: 978-3-540-12761-1. DOI: 10.1007/978-3-662-02427-0.
- [50] Mark I. Halpern. “Machine independence: Its technology and economics”. In: *Communications of the ACM* 8.12 (Dec. 1965), pp. 782–785. ISSN: 0001-0782. DOI: 10.1145/365691.365943.
- [51] Maurice H. Howard Halstead. *Elements of Software Science*. Operating and Programming Systems. Elsevier, May 1977. ISBN: 978-0-444-00205-1.
- [52] Jie Han and Michael Orshansky. “Approximate computing: An emerging paradigm for energy-efficient design”. In: *Proceedings of the 8th IEEE European Test Symposium (ETS)*. (Avignon, France). IEEE Computer Society, May 27–30, 2013, pp. 1–6. DOI: 10.1109/ETS.2013.6569370.

- 
- [53] Duane C. Hanselman and Bruce L. Littlefield. *Mastering MATLAB*. 1st. Prentice Hall, 2011. ISBN: 978-0-13-601330-3.
- [54] Albert Hartono, Muthu Manikandan Baskaran, Cédric Bastoul, Albert Cohen, Sriram Krishnamoorthy, Boyana Norris, Jagannathan Ramanujam, and Ponnuswamy Sadayappan. “Parametric multi-level tiling of imperfectly nested loops”. In: *Proceedings of the 23rd International Conference on Supercomputing (ICS)*. (Yorktown Heights, NY, USA). ACM, June 8–12, 2009, pp. 147–157. ISBN: 978-1-60558-498-0. DOI: 10.1145/1542275.1542301.
- [55] Albert Hartono, Muthu Manikandan Baskaran, Jagannathan Ramanujam, and Ponnuswamy Sadayappan. “DynTile: Parametric tiled loop generation for parallel execution on multicore processors”. In: *Proceedings of the IEEE International Symposium on Parallel Distributed Processing (IPDPS)*. (Atlanta, GA, USA). Apr. 19–23, 2010, 12 pp. ISBN: 978-1-4244-6442-5. DOI: 10.1109/IPDPS.2010.5470459.
- [56] Patrick Healy and Nikola S. Nikolov. “How to layer a directed acyclic graph”. In: *Graph Drawing: 9th International Symposium, GD 2001, Vienna, Austria, September 23–26, 2001, Revised Papers*. Ed. by Petra Mutzel, Michael Jünger, and Sebastian Leipert. Vol. 2265. Lecture Notes in Computer Science (LNCS). Springer, 2002, pp. 16–30. ISBN: 978-3-540-45848-7. DOI: 10.1007/3-540-45848-4\_2.
- [57] Florian Heidenreich, Jendrik Johannes, Sven Karol, Mirko Seifert, and Christian Wende. “Derivation and refinement of textual syntax for models”. In: *Proceedings of 5th European Conference on Model Driven Architecture – Foundations and Applications (ECMDA-FA)*. (Enschede, The Netherlands). Ed. by Richard F. Paige, Alan Hartman, and Arend Rensink. Vol. 5562. Lecture Notes in Computer Science (LNCS). Springer, June 23–26, 2009, pp. 114–129. ISBN: 978-3-642-02674-4. DOI: 10.1007/978-3-642-02674-4\_9.
- [58] Carl Hewitt, Peter Bishop, and Richard Steiger. “A universal modular ACTOR formalism for artificial intelligence”. In: *Proceedings of the 3rd International Joint Conference on Artificial Intelligence (IJCAI)*. (Stanford, CA, USA). Morgan Kaufmann Publishers Inc., Aug. 20–23, 1973, pp. 235–245.
- [59] Hewlett-Packard Laboratories. *VEX toolchain*. <http://www.hpl.hp.com/downloads/vex>. 2009.
- [60] Kishan Jainandunsing. “Optimal partitioning scheme for wavefront/systolic array processors”. In: *Proceedings of the IEEE International Symposium on Circuits and Systems (ISCAS)*. (San Jose, CA, USA). May 5–7, 1986, pp. 940–943.
- [61] Stephen C. Johnson. *Yacc: Yet Another Compiler-Compiler*. Computing Science Tech. rep. No. 32. Bell Laboratories, Murray Hill, NJ 07974, 1975.
- [62] Richard M. Karp, Raymond E. Miller, and Shmuel Winograd. “The organization of computations for uniform recurrence equations”. In: *Journal of the Association for Computing Machinery* 14.3 (July 1967), pp. 563–590. ISSN: 0004-5411. DOI: 10.1145/321406.321418.
- [63] Ralf Karrenberg and Sebastian Hack. “Whole-function vectorization”. In: *Proceedings of the 9th Annual IEEE/ACM International Symposium on Code Generation and Optimization (CGO)*. (Chamonix, France). IEEE Computer Society, Apr. 2–6, 2011, pp. 141–150. ISBN: 978-1-61284-356-8. DOI: 10.1109/CGO.2011.5764682.
- [64] Lennart C. L. Kats and Eelco Visser. “The Spoofox language workbench: Rules for declarative specification of languages and IDEs”. In: *ACM SIGPLAN Notices* 45.10 (Oct. 2010), pp. 444–463. ISSN: 0362-1340. DOI: 10.1145/1932682.1869497.

## A. BIBLIOGRAPHY

---

- [65] Steven Kelly, Kalle Lyytinen, and Matti Rossi. “MetaEdit+: A fully configurable multi-user and multi-tool CASE and CAME environment”. In: *Proceedings of the 8th International Conference on Advances Information System Engineering (CAiSE)*. (Heraklion, Crete, Greece). Vol. 1080. Lecture Notes in Computer Science (LNCS). Springer, May 20–24, 1996, pp. 1–21. ISBN: 978-3-540-68451-0. DOI: 10.1007/3-540-61292-0\_1.
- [66] Reinhard Klette and Piero Zamperoni. *Handbook of Image Processing Operators*. John Wiley & Sons, 1996. ISBN: 0-471-95642-2.
- [67] Paul Klint, Tijs van der Storm, and Jurgen Vinju. “RASCAL: A domain specific language for source code analysis and manipulation”. In: *Proceedings of the Ninth IEEE International Working Conference on Source Code Analysis and Manipulation (SCAM)*. (Edmonton, Alberta, Canada). IEEE Computer Society, Sept. 20–21, 2009, pp. 168–177. ISBN: 978-0-7695-3793-1. DOI: 10.1109/SCAM.2009.28.
- [68] Sebastian Kobbe, Lars Bauer, Daniel Lohmann, Wolfgang Schröder-Preikschat, and Jörg Henkel. “DistRM: Distributed resource management for on-chip many-core systems”. In: *Proceedings of the Seventh IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*. (Taipei, Taiwan). ACM, Oct. 9–14, 2011, pp. 119–128. ISBN: 978-1-4503-0715-4. DOI: 10.1145/2039370.2039392.
- [69] Peter Kogge, Keren Bergman, Shekhar Borkar, Dan Campbell, William Carlson, William Dally, Monty Denneau, Paul Franzon, William Harrod, Kerry Hill, Jon Hiller, Sherman Karp, Stephen Keckler, Dean Klein, Robert Lucas, Mark Richards, Al Scarpelli, Steven Scott, Allan Snively, Thomas Sterling, R. Stanley Williams, and Katherine Yelick. *ExaScale computing study: Technology challenges in achieving exascale systems*. Sept. 2008. URL: <http://www.cse.nd.edu/Reports/2008/TR-2008-13.pdf>.
- [70] Peter Kogge and John Shalf. “Exascale computing trends: Adjusting to the “new normal” for computer architecture”. In: *Computing in Science Engineering* 15.6 (Nov. 2013), pp. 16–26. ISSN: 1521-9615. DOI: 10.1109/MCSE.2013.95.
- [71] Jean Kormann, Pedro Cobo, and Andrés Prieto. “Perfectly matched layers for modelling seismic oceanography experiments”. In: *Journal of Sound and Vibration* 317.1 (2008), pp. 354–365. ISSN: 0022-460X. DOI: 10.1016/j.jsv.2008.03.024.
- [72] Tomaz Kosar, Sudev Bohra, and Marjan Mernik. “Domain-specific languages: A systematic mapping study”. In: *Information & Software Technology* 71 (2016), pp. 77–91. ISSN: 0950-5849. DOI: 10.1016/j.infsof.2015.11.001.
- [73] Harald Köstler. “A Multigrid Framework for Variational Approaches in Medical Image Processing and Computer Vision”. Verlag Dr. Hut, Munich, Germany. Dissertation. Friedrich-Alexander-Universität Erlangen-Nürnberg, Germany, 2008, 285 pp. ISBN: 978-3-89963-761-8.
- [74] Harald Köstler, Rochus Schmid, Ulrich Rüde, and Ch. Scheit. “A parallel multigrid accelerated poisson solver for ab initio molecular dynamics applications”. In: *Computing and Visualization in Science* 11.2 (Mar. 2008), pp. 115–122. ISSN: 1433-0369. DOI: 10.1007/s00791-007-0062-0.
- [75] Filip Křikava, Philippe Collet, and Robert B. France. “Manipulating models using internal domain-specific languages”. In: *Proceedings of the 29th Annual ACM Symposium on Applied Computing (SAC)*. (Gyeongju, Republic of Korea). Mar. 24–28, 2014, pp. 1612–1614. ISBN: 978-1-4503-2469-4. DOI: 10.1145/2554850.2555127.

- [76] Stefan Kronawitter and Christian Lengauer. *Optimizations Applied by the ExaStencils Code Generator*. Tech. rep. MIP-1502. Faculty of Computer Science and Mathematics, University of Passau, Jan. 2015.
- [77] Stefan Kronawitter, Holger Stengel, Georg Hager, and Christian Lengauer. “Domain-specific optimization of two Jacobi smoother kernels and their evaluation in the ECM performance model”. In: *Parallel Processing Letters* 24.3 (Sept. 30, 2014), 18 pp. ISSN: 0129-6264. DOI: 10.1142/S0129626414410047.
- [78] Sebastian Kuckuk, Gundolf Haase, Diego A. Vasco, and Harald Köstler. “Towards generating efficient flow solvers with the ExaStencils approach”. In: *Concurrency and Computation: Practice and Experience* 29.17 (2017), e4062–n/a. ISSN: 1532-0634. DOI: 10.1002/cpe.4062.
- [79] H. T. Kung and Charles E. Leiserson. “Systolic arrays (for VLSI)”. In: *SIAM Sparse Matrix Proceedings*. (Philadelphia, PA, USA). Ed. by Iain S. Duff and G. W. Stewart. SIAM. 1978, pp. 245–282.
- [80] Dietmar Kunz, Kai Eck, Holger Fillbrandt, and Til Aach. “Nonlinear multiresolution gradient adaptive filter for medical images”. In: *Proceedings SPIE 5032, Medical Imaging 2003: Image Processing*. May 16, 2003, pp. 732–742. DOI: 10.1117/12.481323.
- [81] Leslie Lamport. *LaTeX – A Document Preparation System: User’s Guide and Reference Manual*. 2nd. Addison-Wesley, 1994. ISBN: 978-0-201-52983-8.
- [82] Vahid Lari. “Invasive Tightly Coupled Processor Arrays”. Dissertation. Friedrich-Alexander-Universität Erlangen-Nürnberg, Germany, Nov. 18, 2015.
- [83] Vahid Lari. *Invasive Tightly Coupled Processor Arrays*. Springer, 2016. ISBN: 978-981-10-1058-3. DOI: 10.1007/978-981-10-1058-3.
- [84] Vahid Lari, Andreas Weichslgartner, Alex Tanase, Michael Witterauf, Faramarz Khosravi, Jürgen Teich, Jürgen Becker, Jan Heißwolf, and Stephanie Friederich. “Providing fault tolerance through invasive computing”. In: *it – Information Technology* 58.6 (Oct. 19, 2016), pp. 309–328. ISSN: 1611-2776. DOI: 10.1515/itit-2016-0022.
- [85] Chris Lattner and Vikram Adve. “LLVM: A compilation framework for lifelong program analysis & transformation”. In: *Proceedings of the International Symposium on Code Generation and Optimization (CGO)*. (Palo Alto, CA, USA). IEEE Computer Society, Mar. 20–24, 2004, pp. 75–88. ISBN: 0-7695-2102-9. DOI: 10.1109/CGO.2004.1281665.
- [86] Michael E. Lesk and Eric Schmidt. *Lex – A Lexical Analyzer Generator*. Computing Science Tech. rep. No. 39. Bell Laboratories, Murray Hill, NJ 07974, 1975.
- [87] Xuejun Liang and Jack Shiann-Ning Jean. “Mapping of generalized template matching onto reconfigurable computers”. In: *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 11.3 (June 2003), pp. 485–498. ISSN: 1063-8210. DOI: 10.1109/TVLSI.2003.812306.
- [88] Xuejun Liang, Jack Jean, and Karen Tomko. “Data buffering and allocation in mapping generalized template matching on reconfigurable systems”. In: *The Journal of Supercomputing* 19.1 (May 2001), pp. 77–91. ISSN: 1573-0484. DOI: 10.1023/A:1011196613858.
- [89] Ikuo Magaki, Moein Khazraee, Luis Vega Gutierrez, and Michael Bedford Taylor. “ASIC clouds: Specializing the datacenter”. In: *ACM SIGARCH Computer Architecture News* 44.3 (June 2016), pp. 178–190. ISSN: 0163-5964. DOI: 10.1145/3007787.3001156.
- [90] John Maloney, Mitchel Resnick, Natalie Rusk, Brian Silverman, and Evelyn Eastmond. “The scratch programming language and environment”. In: *ACM Transactions on Computing Education (TOCE)* 10.4 (Nov. 2010), 16:1–16:15. ISSN: 1946-6226. DOI: 10.1145/1868358.1868363.

## A. BIBLIOGRAPHY

---

- [91] Andrea Marongiu, Alessandro Capotondi, Giuseppe Tagliavini, and Luca Benini. “Simplifying many-core-based heterogeneous SoC programming with offload directives”. In: *IEEE Transactions Industrial Informatics* 11.4 (2015), pp. 957–967. DOI: 10.1109/TII.2015.2449994.
- [92] David Martin and Gerald Estrin. “Models of computations and systems—Evaluation of vertex probabilities in graph models of computations”. In: *Journal of the Association for Computing Machinery* 14.2 (Apr. 1967), pp. 281–299. ISSN: 0004-5411. DOI: 10.1145/321386.321391.
- [93] James Martin. *Design of Real-Time Computer Systems*. Prentice Hall, 1967. ISBN: 0-13-201400-9.
- [94] Robert C. Martin. *Clean Code – A Handbook of Agile Software Craftsmanship*. Prentice Hall, 2009. ISBN: 978-0-13-235088-4.
- [95] Paul E. McKenney, Maged M. Michael, Manish Gupta, Phil Howard, Joshua Triplett, and Jonathan Walpole. *Is Parallel Programming Hard, And If So, Why?* Tech. rep. TR-09-02. Portland State University, Computer Science Department, Feb. 2009.
- [96] Paul E. McKenney, Maged M. Michael, Manish Gupta, Phil Howard, Joshua Triplett, and Jonathan Walpole. *Is Parallel Programming Hard, And If So, Why?* Tech. rep. TR-09-02. Portland State University, Computer Science Department, Feb. 2009.
- [97] Nenad Medvidovic, David S. Rosenblum, David F. Redmiles, and Jason E. Robbins. “Modeling software architectures in the unified modeling language”. In: *ACM Transactions on Software Engineering and Methodology (TOSEM)* 11.1 (Jan. 2002), pp. 2–57. ISSN: 1049-331X. DOI: 10.1145/504087.504088.
- [98] Richard Membarth. “Code Generation for GPU Accelerators from a Domain-Specific Language for Medical Imaging”. Verlag Dr. Hut, Munich, Germany. Dissertation. Friedrich-Alexander-Universität Erlangen-Nürnberg, Germany, 2013, 215 pp. ISBN: 978-3-8439-1074-3.
- [99] Tim Menzies, John Black, Joel Fleming, and Murray Dean. “An expert system for raising pigs”. In: *Proceedings of the First International Conference on Practical Applications of Prolog*. (London, UK). Apr. 1992.
- [100] Marjan Mernik, Jan Heering, and Anthony M. Sloane. “When and how to develop domain-specific languages”. In: *ACM Computing Surveys* 37.4 (Dec. 2005), pp. 316–344. ISSN: 0360-0300. DOI: 10.1145/1118890.1118892.
- [101] Marjan Mernik, Mitja Lenic, Enis Avdicausevic, and Viljem Zumer. “Compiler/interpreter generator system LISA”. In: *Proceedings of the 33rd Annual Hawaii International Conference on System Sciences (HICSS)*. (Maui, HI, USA). Jan. 4–7, 2000, 10 pp. ISBN: 0-7695-0493-0. DOI: 10.1109/HICSS.2000.927021.
- [102] Giovanni De Micheli. *Synthesis and Optimization of Digital Circuits*. 1st. McGraw-Hill Higher Education, 1994. ISBN: 0-07-016333-2.
- [103] Sparsh Mittal. “A survey of techniques for approximate computing”. In: *ACM Computing Surveys (CSUR)* 48.4 (Mar. 2016), 62:1–62:33. ISSN: 0360-0300. DOI: 10.1145/2893356.
- [104] Dan I. Moldovan and Jose A. B. Fortes. “Partitioning and mapping algorithms into fixed size systolic arrays”. In: *IEEE Transactions on Computers* C-35.1 (Jan. 1986), pp. 1–12. ISSN: 0018-9340. DOI: 10.1109/TC.1986.1676652.
- [105] Cleve B. Moler. *Numerical Computing with MATLAB*. SIAM, 2004. ISBN: 978-0-89871-660-3. DOI: 10.1137/1.9780898717952.
- [106] Don Monroe. “Neuromorphic computing gets ready for the (really) big time”. In: *Communications of the ACM* 57.6 (June 2014), pp. 13–15. ISSN: 0001-0782. DOI: 10.1145/2601069.

- [107] Gordon E. Moore. “Cramming more components onto integrated circuits”. In: *Electronics* 38.8 (Apr. 19, 1965), pp. 114–117. ISSN: 0013-5070.
- [108] Steven S. Muchnick. *Advanced Compiler Design and Implementation*. Morgan Kaufmann Publishers Inc., 1997. ISBN: 1-55860-320-4.
- [109] Bonnie A. Nardi. *A Small Matter of Programming: Perspectives on End User Computing*. MIT Press, July 1993. ISBN: 978-0-262-14053-9.
- [110] Martin Odersky, Lex Spoon, and Bill Venners. *Programming in Scala: A comprehensive step-by-step guide*. 3rd. Artima Inc., 2016, 888 pp. ISBN: 978-0-9815316-8-7.
- [111] OpenCV User Site. *Open source computer vision library (OpenCV)*. <http://www.opencv.org/>. June 2017.
- [112] Terence Parr. *The Definitive ANTLR 4 Reference*. 2nd. Pragmatic Bookshelf, 2013. ISBN: 978-1-934356-99-9.
- [113] Oliver Pell, Oskar Mencer, Kuen Hung Tsoi, and Wayne Luk. “Maximum performance computing with dataflow engines”. In: *High-Performance Computing Using FPGAs*. Ed. by Wim Vanderbauwhede and Khaled Benkrid. Springer, 2013, pp. 747–774. ISBN: 978-1-4614-1791-0. DOI: 10.1007/978-1-4614-1791-0\_25.
- [114] Simon Peyton Jones, Jean-Marc Eber, and Julian Seward. “Composing contracts: An adventure in financial engineering (Functional Pearl)”. In: *ACM SIGPLAN Notices* 35.9 (Sept. 2000), pp. 280–292. ISSN: 0362-1340. DOI: 10.1145/357766.351267.
- [115] Jonathan Ragan-Kelley, Andrew Adams, Sylvain Paris, Marc Levoy, Saman Amarasinghe, and Frédo Durand. “Decoupling algorithms from schedules for easy optimization of image processing pipelines”. In: *ACM Transactions on Graphics (TOG)* 31.4 (July 2012), 32:1–32:12. ISSN: 0730-0301. DOI: 10.1145/2185520.2185528.
- [116] Sailesh K. Rao. “Regular Iterative Algorithms and their Implementations on Processor Arrays”. PhD Thesis. Stanford University, CA, USA, 1985.
- [117] Daniel Ratiu, Markus Voelter, Zaur Molotnikov, and Bernhard Schaetz. “Implementing modular domain specific languages and analyses”. In: *Proceedings of the Workshop on Model-Driven Engineering, Verification and Validation (MoDeVVA)*. (Innsbruck, Austria). Sept. 30, 2012, pp. 35–40. ISBN: 978-1-4503-1801-3. DOI: 10.1145/2427376.2427383.
- [118] B. Ramakrishna Rau. “Iterative modulo scheduling: An algorithm for software pipelining loops”. In: *Proceedings of the 27th Annual International Symposium on Microarchitecture (MICRO)*. (San Jose, CA, USA). ACM, Nov. 30–Dec. 2, 1994, pp. 63–74. ISBN: 0-89791-707-3. DOI: 10.1145/192724.192731.
- [119] B. Ramakrishna Rau, Michael S. Schlansker, and Partha P. Tirumalai. *Code Generation Schema for Modulo Scheduled DO-Loops and WHILE-Loops*. Tech. rep. HPL-92-47. Palo Alto, CA, USA: Hewlett-Packard Laboratories, Apr. 1992.
- [120] Eric Steven Raymond. *The Art of UNIX Programming*. Pearson Education, 2003. ISBN: 0-13-142901-9.
- [121] Lakshminarayanan Renganarayanan, DaeGon Kim, Sanjay Rajopadhye, and Michelle Mills Strout. “Parameterized tiled loops for free”. In: *ACM SIGPLAN Notices* 42.6 (June 2007), pp. 405–414. ISSN: 0362-1340. DOI: 10.1145/1273442.1250780.
- [122] Lakshminarayanan Renganarayanan, DaeGon Kim, Michelle Mills Strout, and Sanjay Rajopadhye. “Parameterized loop tiling”. In: *ACM Transactions on Programming Languages and Systems (TOPLAS)* 34.1 (May 2012), 3:1–3:41. ISSN: 0164-0925. DOI: 10.1145/2160910.2160912.

## A. BIBLIOGRAPHY

---

- [123] Douglas T. Ross. “Origins of the APT language for automatically programmed tools”. In: *SIGPLAN Notices* 13.8 (Aug. 1978), pp. 61–99. ISSN: 0362-1340. DOI: 10 . 1145 / 960118 . 808374.
- [124] Vwani Prasad Roychowdhury, Lothar Thiele, Sailesh K. Rao, and Thomas Kailath. “On the localization of algorithms of VLSI processor arrays”. In: *Proceedings of the Workshop on VLSI Signal Processing III*. Nov. 1988, pp. 459–470. ISBN: 978-0-87942-248-6.
- [125] John C. Russ and F. Brent Neal. *The Image Processing Handbook*. 7th. Boca Raton, FL, USA: CRC Press, 2015. ISBN: 978-1-4987-4026-5.
- [126] Jean E. Sammet. *Programming Languages: History and Fundamentals*. Prentice-Hall, Inc., 1969. ISBN: 0-13-729988-5.
- [127] Vinay Saripalli, Guangyu Sun, Asit K. Mishra, Yuan Xie, Suman Datta, and Vijaykrishnan Narayanan. “Exploiting heterogeneity for energy efficiency in chip multiprocessors”. In: *IEEE Journal on Emerging and Selected Topics in Circuits and Systems* 1.2 (June 2011), pp. 109–119. ISSN: 2156-3357. DOI: 10 . 1109 / JETCAS . 2011 . 2158343.
- [128] Moritz Schmid. “Rapid Prototyping for Hardware Accelerators in the Medical Imaging Domain”. Dissertation. Friedrich-Alexander-Universität Erlangen-Nürnberg, Germany, 2015. URN: urn : nbn : de : bvb : 29-opus4-65733.
- [129] Douglas C. Schmidt. “Guest editor’s introduction: model-driven engineering”. In: *Computer* 39.2 (Feb. 2006), pp. 25–31. ISSN: 0018-9162. DOI: 10 . 1109 / MC . 2006 . 58.
- [130] Tobias Schwarzer, Andreas Weichslgartner, Michael Glaß, Stefan Wildermann, Peter Brand, and Jürgen Teich. “Symmetry-eliminating design space exploration for hybrid application mapping on many-core architectures”. In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* PP.99 (2017), 14 pp. ISSN: 0278-0070. DOI: 10 . 1109 / TCAD . 2017 . 2695894.
- [131] Tim Sheard. “Accomplishments and research challenges in meta-programming”. In: *Proceedings of the Second International Workshop Semantics, Applications, and Implementation of Program Generation (SAIG)*. (Florence, Italy). Ed. by Walid Taha. Springer, Sept. 6, 2001, pp. 2–44. ISBN: 978-3-540-44806-8. DOI: 10 . 1007 / 3-540-44806-3\_2.
- [132] Dave Shreiner, Graham Sellers, John M. Kessenich, and Bill M. Licea-Kane. *OpenGL Programming Guide: The Official Guide to Learning OpenGL, Version 4.3*. 8th. Addison-Wesley Professional, 2013. ISBN: 978-0-321-77303-6.
- [133] Todor Stefanov and Ed F. Deprettere. “Deriving process networks from weakly dynamic applications in system-level design”. In: *Proceedings of the 1st IEEE/ACM/IFIP International Conference on Hardware/Software Codesign & System synthesis (CODES+ISSS)*. (Newport Beach, CA, USA). ACM Press, 2003, pp. 90–96. ISBN: 1-58113-742-7. DOI: 10 . 1145 / 944645 . 944673.
- [134] Herb Sutter. “A fundamental turn toward concurrency in software”. In: *Dr. Dobb’s Journal* 30.3 (Mar. 1, 2005). <http://www.drdobbs.com/web-development/a-fundamental-turn-toward-concurrency-in/184405990>.
- [135] Herb Sutter. *The free lunch is over: A fundamental turn toward concurrency in software*. <http://www.gotw.ca/publications/concurrency-ddj.htm>. 2009.
- [136] Earl E. Swartzlander. *Systolic Signal Processing Systems*. Marcel Dekker, 1987. ISBN: 0-8247-7717-4.
- [137] Alexandru Tanase. “Symbolic Parallelization of Nested Loop Programs”. Dissertation. Friedrich-Alexander-Universität Erlangen-Nürnberg, Germany, Sept. 19, 2017.



- [138] Jürgen Teich. “A Compiler for Application-Specific Processor Arrays”. Shaker Verlag, Aachen, Germany. Dissertation. Universität des Saarlandes, 1993. ISBN: 3-86111-701-0.
- [139] Jürgen Teich. “Invasive algorithms and architectures”. In: *it – Information Technology* 50.5 (Sept. 2008), pp. 300–310. ISSN: 1611-2776. DOI: 10.1524/itit.2008.0499.
- [140] Jürgen Teich, Michael Glaß, Sascha Roloff, Wolfgang Schröder-Preikschat, Gregor Snelting, Andreas Weichslgartner, and Stefan Wildermann. “Language and compilation of parallel programs for \*-predictable MPSoC execution using invasive computing”. In: *Proceedings of the 10th IEEE International Symposium on Embedded Multicore/Many-core Systems-on-Chip (MCSoc)*. Lyon, France, Sept. 21–23, 2016, pp. 313–320. ISBN: 978-1-5090-3531-1. DOI: 10.1109/MCSoc.2016.30.
- [141] Jürgen Teich and Christian Haubelt. *Digitale Hardware/Software-Systeme: Synthese und Optimierung*. 2nd ed. Springer, 2007. ISBN: 978-3-540-46822-6. DOI: 10.1007/978-3-540-46822-0.
- [142] Jürgen Teich, Jörg Henkel, Andreas Herkersdorf, Doris Schmitt-Landsiedel, Wolfgang Schröder-Preikschat, and Gregor Snelting. “Invasive computing: An overview”. In: *Multiprocessor System-on-Chip – Hardware Design and Tool Integration*. Ed. by Michael Hübner and Jürgen Becker. Springer, 2011, pp. 241–268. ISBN: 978-1-4419-6459-5. DOI: 10.1007/978-1-4419-6460-1\_11.
- [143] Jürgen Teich and Lothar Thiele. “Control generation in the design of processor arrays”. In: *Journal of VLSI Signal Processing Systems* 3.1-2 (1991), pp. 77–92. ISSN: 0922-5773. DOI: 10.1007/BF00927836.
- [144] The Trimaran Consortium. *An infrastructure for research in backend compilation and architecture exploration*. <http://www.trimaran.org>. 2010.
- [145] Lothar Thiele. “On the hierarchical design of VLSI processor arrays”. In: *Proceedings of the IEEE International Symposium on Circuits and Systems (ISCAS)*. Vol. 3. June 1988, pp. 2517–2520. DOI: 10.1109/ISCAS.1988.15454.
- [146] Lothar Thiele. “Resource Constrained Scheduling of Uniform Algorithms”. In: *Journal of VLSI Signal Processing* 10 (1995), pp. 295–310.
- [147] Lothar Thiele and Vwani Prasad Roychowdhury. “Systematic design of local processor arrays for numerical algorithms”. In: *Proceedings of the International Workshop on Algorithms and Parallel VLSI Architectures*. (Amsterdam, The Netherlands). Ed. by Ed F. Deprettere and A. J. van der Veen. Vol. A: Tutorials. Elsevier, 1991, pp. 329–339.
- [148] Laurence Tratt. “Domain specific language implementation via compile-time meta-programming”. In: *ACM Transactions on Programming Languages and Systems (TOPLAS)* 30.6 (Oct. 2008), 31:1–31:40. ISSN: 0164-0925. DOI: 10.1145/1391956.1391958.
- [149] Arie van Deursen, Paul Klint, and Joost Visser. “Domain-specific languages: An annotated bibliography”. In: *ACM SIGPLAN Notices* 35.6 (June 2000), pp. 26–36. ISSN: 0362-1340. DOI: 10.1145/352029.352035.
- [150] Dave Wecker and Krysta M. Svore. “LIQUI|>: A software design architecture and domain-specific language for quantum computing”. In: *The Computing Research Repository (CoRR)* (Feb. 18, 2014), 14 pp. arXiv: 1402.4467 [quant-ph].

## A. BIBLIOGRAPHY

---

- [151] Andreas Weichslgartner, Deepak Gangadharan, Stefan Wildermann, Michael Glaß, and Jürgen Teich. “DAARM: Design-time application analysis and run-time mapping for predictable execution in many-core systems”. In: *Proceedings of the International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*. (New Dehli, India). ACM, Oct. 12–17, 2014, 34:1–34:10. ISBN: 978-1-4503-3051-0. DOI: 10.1145/2656075.2656083.
- [152] Andreas Weichslgartner, Stefan Wildermann, Deepak Gangadharan, Michael Glaß, and Jürgen Teich. “A design-time/run-time application mapping methodology for predictable execution time in MPSoCs”. In: *The Computing Research Repository (CoRR)* (Nov. 16, 2017), 30 pp. arXiv: 1711.05932 [cs.DC].
- [153] Richard L. Wexelblat, ed. *History of Programming Languages*. Academic Press, 1981. ISBN: 0-12-745040-8.
- [154] Greg Wilson. *Are VHLLs really high-level?* [http://www.oreilly.com/news/vhll\\_1299.html](http://www.oreilly.com/news/vhll_1299.html). Dec. 1999.
- [155] Niklaus Wirth. “Program development by stepwise refinement”. In: *Communications of the ACM* 14.4 (Apr. 1971), pp. 221–227. ISSN: 0001-0782. DOI: 10.1145/362575.362577.
- [156] Michael Joseph Wolfe. *High performance compilers for parallel computing*. Ed. by Carter Shanklin and Leda Ortega. Addison-Wesley, 1995. ISBN: 0-8053-2730-4.
- [157] Jingling Xue. “Unimodular transformations of non-perfectly nested loops”. In: *Parallel Computing* 22.12 (1997), pp. 1621–1645. ISSN: 0167-8191. DOI: 10.1016/S0167-8191(96)00063-4.
- [158] Yoav Yaacoby and Peter R. Cappello. “Scheduling a system of affine recurrence equations onto a systolic array”. In: *Proceedings of the International Conference on Systolic Arrays*. (San Diego, CA, USA). May 25–27, 1988, pp. 373–382. DOI: 10.1109/ARRAYS.1988.18077.

## A.2 Personal Bibliography (186)


All journal, conference, and workshop papers listed in the following were selected for publication by peers or international program committees in a formal review process. They have been published in printed journals, proceedings, or widely recognized online archives (ACM Digital Library, IEEE Xplore, SpringerLink, etc.). Different publication categories are denoted by capital prefixes, e.g., “B” for books, “C” for chapters in books, “J” for journal articles, “P” for papers in proceedings, and so on. References to key papers of this habilitation treatise are additionally marked by a book symbol, such as [J9📖].

### Books and Proceedings (10)

- [B1] Alexandru Tanase, Frank Hannig, and Jürgen Teich. *Symbolic Parallelization of Nested Loop Programs*. Springer, Feb. 2018. ISBN: 978-3-319-73908-3. In press.
- [B2] Frank Hannig, João M. P. Cardoso, Thilo Pionteck, Dietmar Fey, Wolfgang Schröder-Preikschat, and Jürgen Teich, eds. *Proceedings of the 29th International Conference on Architecture of Computing Systems (ARCS)*. Vol. 9637. Lecture Notes in Computer Science (LNCS). Springer, 2016, 402 pp. ISBN: 978-3-319-30694-0. DOI: 10.1007/978-3-319-30695-7.
- [B3] Dirk Koch, Frank Hannig, and Daniel Ziener, eds. *FPGAs for Software Programmers*. Springer, June 2016, 327 pp. ISBN: 978-3-319-26406-6. DOI: 10.1007/978-3-319-26408-0.

- [B4] Frank Hannig, Dirk Koch, and Daniel Ziener, eds. *Proceedings of the Second International Workshop on FPGAs for Software Programmers (FSP 2015)*. (London, United Kingdom). Sept. 1, 2015, 104 pp. arXiv: 1508.06320 [cs.AR].
- [B5] Frank Hannig, Dietmar Fey, and Anton Lokhmotov, eds. *Proceedings of the DATE Friday Workshop on Heterogeneous Architectures and Design Methods for Embedded Image Systems (HIS 2015)*. (Grenoble, France). Mar. 13, 2015, 35 pp. arXiv: 1502.07241 [cs.AR].
- [B6] Frank Hannig, Dirk Koch, and Daniel Ziener, eds. *Proceedings of the First International Workshop on FPGAs for Software Programmers (FSP 2014)*. (Munich, Germany). Sept. 1, 2014, 82 pp. arXiv: 1408.4423 [cs.AR].
- [B7] Frank Hannig and Jürgen Teich, eds. *Proceedings of the First Workshop on Resource Awareness and Adaptivity in Multi-Core Computing (Racing 2014)*. (Paderborn, Germany). May 29–30, 2014, 62 pp. arXiv: 1405.2281 [cs.DC].
- [B8] Joseph R. Cavallaro, Milos D. Ercegovac, Frank Hannig, Paolo Ienne, Earl E. Swartzlander, Jr., and Alexandre F. Tenca, eds. *Proceedings of the 22nd IEEE International Conference on Application-specific Systems, Architectures and Processors (ASAP)*. IEEE Computer Society, 2011. ISBN: 978-1-4577-1292-0. DOI: 10.1109/ASAP.2011.6043219.
- [B9] François Charot, Frank Hannig, Jürgen Teich, and Christophe Wolinski, eds. *Proceedings of the 21st IEEE International Conference on Application-specific Systems, Architectures and Processors (ASAP)*. IEEE Computer Society, 2010. ISBN: 978-1-4244-6967-3. DOI: 10.1109/ASAP.2010.5540766.
- [B10] Frank Hannig. “Scheduling Techniques for High-Throughput Loop Accelerators”. Verlag Dr. Hut, Munich, Germany. Dissertation. Friedrich-Alexander-Universität Erlangen-Nürnberg, Germany, Aug. 11, 2009, 307 pp. ISBN: 978-3-86853-220-3.

## Book Chapters (11)

- [C1]  Christian Schmitt, Sebastian Kuckuk, Frank Hannig, Jürgen Teich, Harald Köstler, Ulrich Rüde, and Christian Lengauer. “Systems of partial differential equations in ExaSlang”. In: *Software for Exascale Computing – SPPEXA 2013–2015*. Ed. by Hans-Joachim Bungartz, Philipp Neumann, and Wolfgang E. Nagel. Vol. 113. Lecture Notes in Computational Science and Engineering (LNCSE). Springer, Aug. 2016, pp. 47–67. ISBN: 978-3-319-40526-1. DOI: 10.1007/978-3-319-40528-5\_3.
- [C2] Frank Hannig. “A quick tour of high-level synthesis solutions for FPGAs”. In: *FPGAs for Software Programmers*. Ed. by Dirk Koch, Frank Hannig, and Daniel Ziener. Springer, June 2016. Chap. 3, pp. 49–59. ISBN: 978-3-319-26406-6. DOI: 10.1007/978-3-319-26408-0\_3.
- [C3] Dirk Koch, Daniel Ziener, and Frank Hannig. “FPGA versus software programming: why, when, and how?” In: *FPGAs for Software Programmers*. Ed. by Dirk Koch, Frank Hannig, and Daniel Ziener. Springer, June 2016. Chap. 1, pp. 1–21. ISBN: 978-3-319-26406-6. DOI: 10.1007/978-3-319-26408-0\_1.
- [C4] Moritz Schmid, Oliver Reiche, Frank Hannig, and Jürgen Teich. “HIPAcc”. In: *FPGAs for Software Programmers*. Ed. by Dirk Koch, Frank Hannig, and Daniel Ziener. Springer, June 2016. Chap. 12, pp. 205–223. ISBN: 978-3-319-26406-6. DOI: 10.1007/978-3-319-26408-0\_12.

## A. BIBLIOGRAPHY


---

- [C5] Moritz Schmid, Christian Schmitt, Frank Hannig, Gorker Alp Malazgirt, Nehir Sönmez, Arda Yurdakul, and Adrián Cristal. “Big data and HPC acceleration with Vivado HLS”. In: *FPGAs for Software Programmers*. Ed. by Dirk Koch, Frank Hannig, and Daniel Ziener. Springer, June 2016. Chap. 7, pp. 115–136. ISBN: 978-3-319-26406-6. DOI: 10.1007/978-3-319-26408-0\_7.
- [C6] Jürgen Teich, Srinivas Boppu, Frank Hannig, and Vahid Lari. “Compact code generation and throughput optimization for coarse-grained reconfigurable arrays”. In: *Transforming Reconfigurable Systems: A Festschrift Celebrating the 60th Birthday of Professor Peter Cheung*. Ed. by Wayne Luk and George A. Constantinides. London, UK: Imperial College Press, Apr. 2015. Chap. 10, pp. 167–206. ISBN: 978-1-78326-696-8. DOI: 10.1142/9781783266975\_0010.
- [C7] Moritz Schmid, Frank Hannig, Alexandru Tanase, and Jürgen Teich. “High-level synthesis revised – Generation of FPGA accelerators from a domain-specific language using the polyhedron model”. In: *Parallel Computing: Accelerating Computational Science and Engineering (CSE)*. Ed. by Michael Bader, Arndt Bode, Hans-Joachim Bungartz, Michael Gerndt, Gerhard R. Joubert, and Frans J. Peters. Vol. 25. Advances in Parallel Computing. Amsterdam, The Netherlands: IOS Press, Apr. 2014, pp. 497–506. ISBN: 978-1-61499-380-3. DOI: 10.3233/978-1-61499-381-0-497.
- [C8] Alexandru Tanase, Vahid Lari, Frank Hannig, and Jürgen Teich. “Exploitation of quality/-throughput tradeoffs in image processing through invasive computing”. In: *Parallel Computing: Accelerating Computational Science and Engineering (CSE)*. Ed. by Michael Bader, Arndt Bode, Hans-Joachim Bungartz, Michael Gerndt, Gerhard R. Joubert, and Frans J. Peters. Vol. 25. Advances in Parallel Computing. Amsterdam, The Netherlands: IOS Press, Apr. 2014, pp. 53–62. ISBN: 978-1-61499-380-3. DOI: 10.3233/978-1-61499-381-0-53.
- [C9] Alexey Kupriyanov, Frank Hannig, Dmitrij Kissler, and Jürgen Teich. “MAML: An ADL for designing single and multiprocessor architectures”. In: *Processor Description Languages*. Ed. by Prabhat Mishra and Nikil Dutt. Systems on Silicon. Morgan Kaufmann, June 2008. Chap. 12, pp. 295–327. ISBN: 978-0-12-374287-2.
- [C10] Frank Hannig and Jürgen Teich. “Energy estimation and optimization for piecewise regular processor arrays”. In: *Domain-Specific Processors: Systems, Architectures, Modeling, and Simulation*. Ed. by Shuvra S. Bhattacharyya et al. Signal Processing and Communications 20. New York, USA: Marcel Dekker, Jan. 2004. Chap. 6, pp. 107–126. ISBN: 0-8247-4711-9.
- [C11] Marcus Bednara, Frank Hannig, and Jürgen Teich. “Generation of distributed loop control”. In: *Embedded Processor Design Challenges: Systems, Architectures, Modeling, and Simulation – SAMOS*. Ed. by Ed F. Deprettere, Jürgen Teich, and Stamatis Vassiliadis. Vol. 2268. Lecture Notes in Computer Science (LNCS). Springer, Mar. 2002, pp. 154–170. ISBN: 3-540-43322-8. DOI: 10.1007/3-540-45874-3\_9.

## Editorials (2)





- [E1] Frank Hannig, João M. P. Cardoso, and Dietmar Fey. “Introduction to the special issue on architecture of computing systems”. In: *Journal of Systems Architecture* 77 (June 1, 2017), pp. 1–2. ISSN: 1383-7621. DOI: 10.1016/j.sysarc.2017.04.003.
- [E2] Frank Hannig and Andreas Herkersdorf. “Introduction to the special issue on testing, prototyping, and debugging of multi-core architectures”. In: *Journal of Systems Architecture* 61.10 (Nov. 7, 2015), p. 600. ISSN: 1383-7621. DOI: 10.1016/j.sysarc.2015.11.003.

## Journal Articles (27)

- [J1]  Alexandru Tanase, Michael Witterauf, Jürgen Teich, and Frank Hannig. “Symbolic multi-level loop mapping of loop programs for massively parallel processor arrays”. In: *ACM Transactions on Embedded Computing Systems (TECS)* 17.2 (Dec. 2017), 31:1–31:27. ISSN: 1539-9087. DOI: 10.1145/3092952.
- [J2]  Vivek Singh Bhadouria, Alexandru Tanase, Moritz Schmid, Frank Hannig, Jürgen Teich, and Dibyendu Ghoshal. “A novel image impulse noise removal algorithm optimized for hardware accelerators”. In: *Journal of Signal Processing Systems* 89.2 (Nov. 1, 2017), pp. 225–242. ISSN: 1939-8018. DOI: 10.1007/s11265-016-1187-5.
- [J3] Didem Unat, Anshu Dubey, Torsten Hoefler, John Shalf, Mark Abraham, Mauro Bianco, Bradford L. Chamberlain, Romain Cledat, H. Carter Edwards, Hal Finkel, Karl Furlinger, Frank Hannig, Emmanuel Jeannot, Amir Kamil, Jeff Keasler, Paul H. J. Kelly, Vitus J. Leung, Hatem Ltaief, Naoya Maruyama, Chris J. Newburn, and Miquel Pericàs. “Trends in data locality abstractions for HPC systems”. In: *IEEE Transactions on Parallel and Distributed Systems* 28.10 (Oct. 2017), pp. 3007–3020. ISSN: 1045-9219. DOI: 10.1109/TPDS.2017.2703149.
- [J4] Heba Khdr, Santiago Pagani, Éricles R. Sousa, Vahid Lari, Anuj Pathania, Frank Hannig, Muhammad Shafique, Jürgen Teich, and Jörg Henkel. “Power density-aware resource management for heterogeneous tiled multicores”. In: *IEEE Transactions on Computers (TC)* 66.3 (Mar. 1, 2017), pp. 488–501. ISSN: 0018-9340. DOI: 10.1109/TC.2016.2595560.
- [J5]  Oliver Reiche, M. Akif Özkan, Frank Hannig, Jürgen Teich, and Moritz Schmid. “Loop parallelization techniques for FPGA accelerator synthesis”. In: *Journal of Signal Processing Systems* (Feb. 2017), 25 pp. ISSN: 1939-8018. DOI: 10.1007/s11265-017-1229-7. In press.
- [J6] Harald Köstler, Christian Schmitt, Sebastian Kuckuk, Stefan Kronawitter, Frank Hannig, Jürgen Teich, Ulrich Rüde, and Christian Lengauer. “A Scala prototype to generate multigrid solver implementations for different problems and target multi-core platforms”. In: *International Journal of Computational Science and Engineering* 14.2 (Jan. 2017), pp. 150–163. ISSN: 1742-7185. DOI: 10.1504/IJCSE.2017.082879.
- [J7] Santiago Pagani, Lars Bauer, Qingqing Chen, Elisabeth Glocker, Frank Hannig, Andreas Herkersdorf, Heba Khdr, Anuj Pathania, Ulf Schlichtmann, Doris Schmitt-Landsiedel, Mark Sagi, Éricles R. Sousa, Philipp Wagner, Volker Wenzel, Thomas Wild, and Jörg Henkel. “Dark silicon management: An integrated and coordinated cross-layer approach”. In: *it – Information Technology* (2016). ISSN: 1611-2776. DOI: 10.1515/itit-2016-0028.
- [J8] Christian Schmitt, Moritz Schmid, Sebastian Kuckuk, Harald Köstler, Jürgen Teich, and Frank Hannig. “Reconfigurable hardware generation of multigrid solvers with conjugate gradient coarse-grid solution”. In: *Parallel Processing Letters* (2016). ISSN: 0129-6264. In press.
- [J9]  Richard Membarth, Oliver Reiche, Frank Hannig, Jürgen Teich, Mario Körner, and Wieland Eckert. “HIPAcc: A domain-specific language and compiler for image processing”. In: *IEEE Transactions on Parallel and Distributed Systems* 27.1 (Jan. 1, 2016), pp. 210–224. ISSN: 1045-9219. DOI: 10.1109/TPDS.2015.2394802.
- [J10] Johny Paul, Walter Stechele, Benjamin Oechslein, Christoph Erhardt, Jens Schedel, Daniel Lohmann, Wolfgang Schröder-Preikschat, Manfred Kröhnert, Tamim Asfour, Éricles R. Sousa, Vahid Lari, Frank Hannig, Jürgen Teich, Artjom Grudnitsky, Lars Bauer, and Jörg Henkel. “Resource awareness on heterogeneous mpsoes for image processing”. In: *Journal of Systems Architecture* 61.10 (Nov. 6, 2015), pp. 668–680. ISSN: 1383-7621. DOI: 10.1016/j.sysarc.2015.09.002.

## A. BIBLIOGRAPHY

---

- [J11] Oliver Reiche, Konrad Häublein, Marc Reichenbach, Moritz Schmid, Frank Hannig, Jürgen Teich, and Dietmar Fey. “Synthesis and optimization of image processing accelerators using domain knowledge”. In: *Journal of Systems Architecture* 61.10 (Oct. 9, 2015), pp. 646–658. ISSN: 1383-7621. DOI: 10.1016/j.sysarc.2015.09.004.
- [J12]  Richard Membarth, Oliver Reiche, Christian Schmitt, Frank Hannig, Jürgen Teich, Markus Stürmer, and Harald Köstler. “Towards a performance-portable description of geometric multi-grid algorithms using a domain-specific language”. In: *Journal of Parallel and Distributed Computing* 74.12 (Dec. 2014), pp. 3191–3201. ISSN: 0743-7315. DOI: 10.1016/j.jpdc.2014.08.008.
- [J13] Sven Apel, Matthias Bolten, Armin Größlinger, Frank Hannig, Harald Köstler, Christian Lengauer, Ulrich Rüde, and Jürgen Teich. “ExaStencils: Advanced stencil-code engineering”. In: *inSiDE* 12.2 (Nov. 2014), pp. 60–63.
- [J14] Alexander Grebhahn, Sebastian Kuckuk, Christian Schmitt, Harald Köstler, Norbert Siegmund, Sven Apel, Frank Hannig, and Jürgen Teich. “Experiments on optimizing the performance of stencil codes with SPL Conqueror”. In: *Parallel Processing Letters* 24.3 (Sept. 30, 2014), 19 pp. ISSN: 0129-6264. DOI: 10.1142/S0129626414410011.
- [J15]  Jürgen Teich, Alexandru Tanase, and Frank Hannig. “Symbolic mapping of loop programs onto processor arrays”. In: *Journal of Signal Processing Systems* 77.1–2 (July 11, 2014), pp. 31–59. ISSN: 1939-8018. DOI: 10.1007/s11265-014-0905-0.
- [J16] Harald Köstler, Christian Schmitt, Sebastian Kuckuk, Frank Hannig, Jürgen Teich, and Ulrich Rüde. “A Scala prototype to generate multigrid solver implementations for different problems and target multi-core platforms”. In: *The Computing Research Repository (CoRR)* (June 20, 2014), 18 pp. arXiv: 1406.5369 [cs.MS].
- [J17]  Srinivas Boppu, Frank Hannig, and Jürgen Teich. “Compact code generation for tightly-coupled processor arrays”. In: *Journal of Signal Processing Systems* 77.1–2 (May 31, 2014), pp. 5–29. ISSN: 1939-8018. DOI: 10.1007/s11265-014-0891-2.
- [J18]  Frank Hannig, Vahid Lari, Srinivas Boppu, Alexandru Tanase, and Oliver Reiche. “Invasive tightly-coupled processor arrays: A domain-specific architecture/compiler co-design approach”. In: *ACM Transactions on Embedded Computing Systems (TECS)* 13.4s (Mar. 2014), 133:1–133:29. ISSN: 1539-9087. DOI: 10.1145/2584660.
- [J19] Vahid Lari, Shravan Muddasani, Srinivas Boppu, Frank Hannig, Moritz Schmid, and Jürgen Teich. “Hierarchical power management for adaptive tightly-coupled processor arrays”. In: *ACM Transactions on Design Automation of Electronic Systems (TODAES)* 18.1 (Jan. 2013), 2:1–2:25. ISSN: 1084-4309. DOI: 10.1145/2390191.2390193.
- [J20] Richard Membarth, Hritam Dutta, Frank Hannig, and Jürgen Teich. “Efficient mapping of streaming applications for image processing on graphics cards”. In: *Transactions on High-Performance Embedded Architectures and Compilers (Transactions on HiPEAC)* 5.3 (2011).
- [J21] Dmitriy Kissler, Daniel Gran, Zoran A. Salcic, Frank Hannig, and Jürgen Teich. “Scalable many-domain power gating in coarse-grained reconfigurable processor arrays”. In: *IEEE Embedded Systems Letters* 3.2 (June 2011), pp. 58–61. ISSN: 1943-0663. DOI: 10.1109/LES.2011.2124438.
- [J22] Dmitriy Kissler, Frank Hannig, and Jürgen Teich. “Efficient evaluation of power/area/latency design trade-offs for coarse-grained reconfigurable processor arrays”. In: *Journal of Low Power Electronics* 7.1 (Feb. 2011), pp. 29–40. ISSN: 1546-1998. DOI: 10.1166/jolpe.2011.1114.






- [J23] Dmitrij Kissler, Andreas Strawetz, Frank Hannig, and Jürgen Teich. “Power-efficient reconfiguration control in coarse-grained dynamically reconfigurable architectures”. In: *Journal of Low Power Electronics* 5.1 (Apr. 2009), pp. 96–105. ISSN: 1546-1998. DOI: 10.1166/jolpe.2009.1008.
- [J24] Hritam Dutta, Dmitrij Kissler, Frank Hannig, Alexey Kupriyanov, Jürgen Teich, and Bernard Pottier. “A holistic approach for tightly coupled reconfigurable parallel processors”. In: *Microprocessors and Microsystems* 33.1 (Feb. 2009), pp. 53–62. ISSN: 0141-9331. DOI: 10.1016/j.micpro.2008.08.007.
- [J25] Dmitrij Kissler, Frank Hannig, and Jürgen Teich. “Schwach programmiert macht stark – Massiv parallele Prozessorfelder”. In: *Design & Elektronik* 4 (2007), pp. 34–39.
- [J26] Hritam Dutta, Frank Hannig, Holger Ruckdeschel, and Jürgen Teich. “Efficient control generation for mapping nested loop programs onto processor arrays”. In: *Journal of Systems Architecture* 53.5–6 (May–June 2007), pp. 300–309. ISSN: 1383-7621. DOI: 10.1016/j.sysarc.2006.10.009.
- [J27] Frank Hannig, Hritam Dutta, and Jürgen Teich. “Mapping a class of dependence algorithms to coarse-grained reconfigurable arrays: architectural parameters and methodology”. In: *International Journal of Embedded Systems* 2.1/2 (2006), pp. 114–127. ISSN: 1741-1068. DOI: 10.1504/IJES.2006.010170.

### Papers in Conference, Symposia, and Workshop Proceedings (127)


- [P1] Jörg Fickenscher, Jens Schlumberger, Frank Hannig, Mohamed Essayed Bouzouraa, and Jürgen Teich. “Cell-based update algorithm for occupancy grid maps and hybrid map for ADAS on embedded GPUs”. In: *Proceedings of the Conference on Design, Automation and Test in Europe (DATE)*. (Dresden, Germany). Mar. 19–23, 2018. Forthcoming.
- [P2] Éricles R. Sousa, Alexandru Tanase, Frank Hannig, and Jürgen Teich. “A reconfigurable memory architecture for system integration of coarse-grained reconfigurable arrays”. In: *Proceedings of the International Conference on Reconfigurable Computing and FPGAs (ReConFig)*. (Cancun, Mexico). IEEE, Dec. 4–6, 2017.
- [P3] Oliver Reiche, M. Akif Özkan, Richard Membarth, Jürgen Teich, and Frank Hannig. “Generating FPGA-based image processing accelerators with Hipacc”. In: *Proceedings of the International Conference On Computer Aided Design (ICCAD)*. (Irvine, CA, USA). IEEE, Nov. 13–16, 2017, pp. 1012–1019. ISBN: 978-1-5386-3093-8.
- [P4]  Sascha Roloff, Frank Hannig, and Jürgen Teich. “High performance network-on-chip simulation by interval-based timing predictions”. In: *Proceedings of the 15th IEEE/ACM Symposium on Embedded Systems for Real-time Multimedia (ESTIMedia)*. (Seoul, Republic of Korea). ACM, Oct. 15–20, 2017, pp. 2–11. ISBN: 978-1-4503-5117-1. DOI: 10.1145/3139315.3139320.
- [P5]  Michael Witterauf, Frank Hannig, and Jürgen Teich. “Constructing fast and cycle-accurate simulators for configurable accelerators using C++ templates”. In: *Proceedings of the 28th International Symposium on Rapid System Prototyping (RSP)*. (Seoul, Republic of Korea). ACM, Oct. 19–20, 2017, pp. 9–15. ISBN: 978-1-4503-5418-9. DOI: 10.1145/3130265.3130324.
- [P6] Marcel Brand, Frank Hannig, Alexandru Tanase, and Jürgen Teich. “Orthogonal instruction processing: An alternative to lightweight VLIW processors”. In: *Proceedings of the IEEE 11th International Symposium on Embedded Multicore/Many-core Systems-on-Chip (MCSoc)*. (Seoul, Republic of Korea). IEEE Computer Society, Sept. 18–20, 2017, pp. 5–12. ISBN: 978-1-5386-3441-7. DOI: 10.1109/MCSoc.2017.17.

## A. BIBLIOGRAPHY

---

- [P7] M. Akif Özkan, Oliver Reiche, Frank Hannig, and Jürgen Teich. “A highly efficient and comprehensive image processing library for C++-based high-level synthesis”. In: *Proceedings of the Fourth International Workshop on FPGAs for Software Programmers (FSP)*. (Ghent, Belgium). VDE, Sept. 7, 2017, pp. 23–32. ISBN: 978-3-8007-4443-5.
- [P8] Marcel Brand, Frank Hannig, Alexandru Tanase, and Jürgen Teich. “Efficiency in ILP processing by using orthogonality”. In: *Proceedings of the 28th Annual IEEE International Conference on Application-specific Systems, Architectures and Processors (ASAP)*. (Seattle, WA, USA). IEEE, July 10–12, 2017, p. 207. ISBN: 978-1-5090-4825-0. DOI: 10.1109/ASAP.2017.7995282.
- [P9]  M. Akif Özkan, Oliver Reiche, Frank Hannig, and Jürgen Teich. “Hardware design and analysis of efficient loop coarsening and border handling for image processing”. In: *Proceedings of the 28th Annual IEEE International Conference on Application-specific Systems, Architectures and Processors (ASAP)*. (Seattle, WA, USA). IEEE, July 10–12, 2017, pp. 155–163. ISBN: 978-1-5090-4825-0. DOI: 10.1109/ASAP.2017.7995273.
- [P10] Jörg Fickenscher, Sebastian Reinhart, Mohamed Essayed Bouzouraa, Frank Hannig, and Jürgen Teich. “Convoy tracking for ADAS on embedded GPUs”. In: *Proceedings of the IEEE Intelligent Vehicles Symposium (IV)*. (Redondo Beach, CA, USA). IEEE, June 11–14, 2017, pp. 959–965. ISBN: 978-1-5090-4804-5. DOI: 10.1109/IVS.2017.7995839.
- [P11]  Oliver Reiche, Christof Kobylko, Frank Hannig, and Jürgen Teich. “Auto-vectorization for image processing DSLs”. In: *Proceedings of the 18th ACM SIGPLAN/SIGBED Conference on Languages, Compilers, and Tools for Embedded systems (LCTES)*. (Barcelona, Spain). ACM, June 21–22, 2017, pp. 21–30. ISBN: 978-1-4503-5030-3. DOI: 10.1145/3078633.3081039.
- [P12] Jörg Fickenscher, Oliver Reiche, Jens Schlumberger, Frank Hannig, and Jürgen Teich. “Modeling, programming and performance analysis of automotive environment map representations on embedded GPUs”. In: *Proceedings of the 18th IEEE International High-Level Design Validation and Test Workshop (HLDVT)*. (Santa Cruz, CA, USA). IEEE, Oct. 7–8, 2016, pp. 70–77. ISBN: 978-1-5090-4270-8. DOI: 10.1109/HLDVT.2016.7748257.
- [P13]  M. Akif Özkan, Oliver Reiche, Frank Hannig, and Jürgen Teich. “FPGA-based accelerator design from a domain-specific language”. In: *Proceedings of the 26th International Conference on Field-Programmable Logic and Applications (FPL)*. (Lausanne, Switzerland). IEEE, Aug. 29–Sept. 2, 2016, 9 pp. ISBN: 978-2-8399-1844-2. DOI: 10.1109/FPL.2016.7577357.
- [P14] Konrad Häublein, Marc Reichenbach, Oliver Reiche, M. Akif Özkan, Dietmar Fey, Frank Hannig, and Jürgen Teich. “Hybrid code description for developing fast and resource efficient image processing architectures”. In: *Proceedings of the 16th International Conference on Embedded Computer Systems: Architectures, Modeling and Simulation (SAMOS)*. (Island of Samos, Greece). July 18–21, 2016.
- [P15]  Michael Witterauf, Alexandru Tanase, Frank Hannig, and Jürgen Teich. “Modulo scheduling of symbolically tiled loops for tightly coupled processor arrays”. In: *Proceedings of the 27th IEEE International Conference on Application-specific Systems, Architectures and Processors (ASAP)*. (London, United Kingdom). IEEE, July 6–8, 2016.
- [P16]  Sascha Roloff, Alexander Pöpl, Tobias Schwarzer, Stefan Wildermann, Michael Bader, Michael Glaß, Frank Hannig, and Jürgen Teich. “ActorX10: An actor library for X10”. In: *Proceedings of the Sixth ACM SIGPLAN X10 Workshop (X10)*. (Santa Barbara, CA, USA). ACM, June 14, 2016, pp. 24–29. ISBN: 978-1-4503-4386-2. DOI: 10.1145/2931028.2931033.





- [P17] Éricles R. Sousa, Frank Hannig, and Jürgen Teich. “Reconfigurable buffer structures for coarse-grained reconfigurable arrays”. In: *Proceedings of the 5th IFIP International Embedded Systems Symposium (IESS)*. (Foz do Iguaçu, Brazil). Lecture Notes in Computer Science (LNCS). Springer, Nov. 3–6, 2015, 10 pp.
- [P18] Sascha Roloff, Stefan Wildermann, Frank Hannig, and Jürgen Teich. “Invasive computing for predictable stream processing: A simulation-based case study”. In: *Proceedings of the 13th IEEE Symposium on Embedded Systems for Real-time Multimedia (ESTIMedia)*. (Amsterdam, The Netherlands). IEEE, Oct. 8–9, 2015, 2 pp. ISBN: 978-1-4673-8164-2. DOI: 10.1109/ESTIMedia.2015.7351761.
- [P19] Alexandru Tanase, Michael Witterauf, Jürgen Teich, and Frank Hannig. “Symbolic loop parallelization for balancing I/O and memory accesses on processor arrays”. In: *Proceedings of the 13th ACM-IEEE International Conference on Formal Methods and Models for System Design (MEMOCODE)*. (Austin, TX, USA). IEEE, Sept. 21–23, 2015, pp. 188–197. ISBN: 978-1-5090-0237-5. DOI: 10.1109/MEMCOD.2015.7340486.
- [P20] Moritz Schmid, Oliver Reiche, Frank Hannig, and Jürgen Teich. “Loop coarsening in C-based high-level synthesis”. In: *Proceedings of the 26th IEEE International Conference on Application-specific Systems, Architectures and Processors (ASAP)*. (Toronto, Canada). IEEE, July 27–29, 2015, pp. 166–173. ISBN: 978-1-4799-1925-3. DOI: 10.1109/ASAP.2015.7245730.
- [P21] Alexandru Tanase, Michael Witterauf, Jürgen Teich, Frank Hannig, and Vahid Lari. “On-demand fault-tolerant loop processing on massively parallel processor arrays”. In: *Proceedings of the 26th IEEE International Conference on Application-specific Systems, Architectures and Processors (ASAP)*. (Toronto, Canada). IEEE, July 27–29, 2015, pp. 194–201. ISBN: 978-1-4799-1925-3. DOI: 10.1109/ASAP.2015.7245734.
- [P22] Michael Witterauf, Alexandru Tanase, Frank Hannig, and Jürgen Teich. “Adaptive fault tolerance in tightly coupled processor arrays with invasive computing”. In: *Proceedings of ACACES 2015 Poster Abstracts: Advanced Computer Architecture and Compilation for Embedded Systems*. (Fiuggi, Italy). HiPEAC, July 12–18, 2015, pp. 205–208. ISBN: 978-88-905806-3-5.
- [P23] Vahid Lari, Alexandru Tanase, Jürgen Teich, Michael Witterauf, Faramarz Khosravi, Frank Hannig, and Brett H. Meyer. “A co-design approach for fault-tolerant loop execution on coarse-grained reconfigurable arrays”. In: *Proceedings of the NASA/ESA Conference on Adaptive Hardware and Systems (AHS)*. (Montréal, Quebec, Canada). IEEE, June 15–18, 2015.
- [P24]  Sascha Roloff, David Schafhauser, Frank Hannig, and Jürgen Teich. “Execution-driven parallel simulation of PGAS applications on heterogeneous tiled architectures”. In: *Proceedings of the 52nd ACM/EDAC/IEEE Design Automation Conference (DAC)*. (San Francisco, CA, USA). **HiPEAC Paper Award**. ACM, June 7–11, 2015, 44:1–44:6. ISBN: 978-1-4503-3520-1. DOI: 10.1145/2744769.2744840.
- [P25] Éricles R. Sousa, Frank Hannig, Jürgen Teich, Qingqing Chen, and Ulf Schlichtmann. “Runtime adaptation of application execution under thermal and power constraints in massively parallel processor arrays”. In: *Proceedings of the 18th International Workshop on Software and Compilers for Embedded Systems (SCOPES)*. (St. Goar, Germany). ACM, June 1–3, 2015, pp. 121–124. ISBN: 978-1-4503-3593-5. DOI: 10.1145/2764967.2771933.
- [P26] Oliver Reiche, Konrad Häublein, Marc Reichenbach, Frank Hannig, Jürgen Teich, and Dietmar Fey. “Automatic optimization of hardware accelerators for image processing”. In: *Proceedings of the DATE Friday Workshop on Heterogeneous Architectures and Design Methods for Embedded Image Systems (HIS)*. (Grenoble, France). Mar. 13, 2015, pp. 10–15. arXiv: 1502.07448 [cs.PL].

## A. BIBLIOGRAPHY

---

- [P27] Christian Schmitt, Moritz Schmid, Frank Hannig, Jürgen Teich, Sebastian Kuckuk, and Harald Köstler. “Generation of multigrid-based numerical solvers for FPGA accelerators”. In: *Proceedings of the 2nd International Workshop on High-Performance Stencil Computations (HiStencils)*. (Amsterdam, The Netherlands). Ed. by Armin Größlinger and Harald Köstler. Jan. 20, 2015, pp. 9–15.
- [P28] Deepak Gangadharan, Éricles R. Sousa, Vahid Lari, Frank Hannig, and Jürgen Teich. “Application-driven reconfiguration of shared resources for timing predictability of MPSoC platforms”. In: *Proceedings of Asilomar Conference on Signals, Systems, and Computers (ACSSC)*. (Pacific Grove, CA, USA). IEEE, Nov. 2–5, 2014, pp. 398–403. ISBN: 978-1-4799-8297-4. DOI: 10.1109/ACSSC.2014.7094471.
- [P29]  Christian Schmitt, Sebastian Kuckuk, Frank Hannig, Harald Köstler, and Jürgen Teich. “ExaSlang: A domain-specific language for highly scalable multigrid solvers”. In: *Proceedings of the 4th International Workshop on Domain-Specific Languages and High-Level Frameworks for High Performance Computing (WOLFHPC)*. (New Orleans, LA, USA). IEEE Computer Society, Nov. 17, 2014, pp. 42–51. ISBN: 978-1-4799-7020-9. DOI: 10.1109/WOLFHPC.2014.11.
- [P30] Johnny Paul, Walter Stechele, Éricles R. Sousa, Vahid Lari, Frank Hannig, Jürgen Teich, Manfred Kröhnert, and Tamim Asfour. “Self adaptive harris corner detection on heterogeneous many-core processor”. In: *Proceedings of the Conference on Design and Architectures for Signal and Image Processing (DASIP)*. (Madrid, Spain). IEEE, Oct. 8–10, 2014, 8 pp. ISBN: 979-1-09-227905-4. DOI: 10.1109/DASIP.2014.7115616.
- [P31]  Oliver Reiche, Moritz Schmid, Frank Hannig, Richard Membarth, and Jürgen Teich. “Code generation from a domain-specific language for C-based HLS of hardware accelerators”. In: *Proceedings of the International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*. (New Dehli, India). ACM, Oct. 12–17, 2014, 17:1–17:10. ISBN: 978-1-4503-3051-0. DOI: 10.1145/2656075.2656081.
- [P32]  Alexandru Tanase, Michael Witterauf, Jürgen Teich, and Frank Hannig. “Symbolic inner loop parallelisation for massively parallel processor arrays”. In: *Proceedings of the 12th ACM/IEEE International Conference on Formal Methods and Models for System Design (MEMOCODE)*. (Lausanne, Switzerland). IEEE, Oct. 19–21, 2014, pp. 219–228. ISBN: 978-1-4799-5338-7. DOI: 10.1109/MEMCOD.2014.6961865.
- [P33]  Moritz Schmid, Nicolas Apelt, Frank Hannig, and Jürgen Teich. “An image processing library for C-based high-level synthesis”. In: *Proceedings of the 24th International Conference on Field Programmable Logic and Applications (FPL)*. (Munich, Germany). IEEE, Sept. 2–4, 2014, 4 pp. DOI: 10.1109/FPL.2014.6927424.
- [P34] Moritz Schmid, Oliver Reiche, Christian Schmitt, Frank Hannig, and Jürgen Teich. “Code generation for high-level synthesis of multiresolution applications on FPGAs”. In: *Proceedings of the First International Workshop on FPGAs for Software Programmers (FSP)*. (Munich, Germany). Sept. 1, 2014, pp. 21–26. arXiv: 1408.4721 [cs.CV].
- [P35]  Christian Lengauer, Sven Apel, Matthias Bolten, Armin Größlinger, Frank Hannig, Harald Köstler, Ulrich Rüde, Jürgen Teich, Alexander Grebhahn, Stefan Kronawitter, Sebastian Kuckuk, Hannah Rittich, and Christian Schmitt. “ExaStencils: Advanced stencil-code engineering”. In: *Proceedings of Euro-Par 2014: Parallel Processing Workshops*. (Porto, Portugal). Vol. 8806. Lecture Notes in Computer Science (LNCS). Springer, Aug. 25–29, 2014, pp. 553–564. ISBN: 978-3-319-14312-5. DOI: 10.1007/978-3-319-14313-2\_47.

- [P36] Éricles R. Sousa, Deepak Gangadharan, Frank Hannig, and Jürgen Teich. “Runtime reconfigurable bus arbitration for concurrent applications on heterogeneous MPSoC architectures”. In: *Proceedings of the 17th Euromicro Conference on Digital Systems Design (DSD)*. (Verona, Italy). IEEE, Aug. 27–29, 2014, pp. 74–81. ISBN: 978-1-4799-5793-4. DOI: 10.1109/DSD.2014.105.
- [P37]  Moritz Schmid, Alexandru Tanase, Frank Hannig, Jürgen Teich, Vivek Singh Bhadouria, and Dibyendu Ghoshal. “Domain-specific augmentations for high-level synthesis”. In: *Proceedings of the 25th IEEE International Conference on Application-specific Systems, Architectures and Processors (ASAP)*. (Zurich, Switzerland). IEEE, June 18–20, 2014, pp. 173–177. ISBN: 978-1-4799-3609-0. DOI: 10.1109/ASAP.2014.6868653.
- [P38]  Christian Schmitt, Sebastian Kuckuk, Harald Köstler, Frank Hannig, and Jürgen Teich. “An evaluation of domain-specific language technologies for code generation”. In: *Proceedings of the 14th International Conference on Computational Science and its Applications (ICCSA)*. (Guimaraes, Portugal). IEEE Computer Society, June 30–July 3, 2014, pp. 18–26. ISBN: 978-1-4799-4264-0. DOI: 10.1109/ICCSA.2014.16.
- [P39] Vahid Lari, Alexandru Tanase, Frank Hannig, and Jürgen Teich. “Massively parallel processor architectures for resource-aware computing”. In: *Proceedings of the First Workshop on Resource Awareness and Adaptivity in Multi-Core Computing (Racing 2014)*. (Paderborn, Germany). May 29–30, 2014, pp. 1–7. arXiv: 1405.2907 [cs.AR].
- [P40] Deepak Gangadharan, Alexandru Tanase, Frank Hannig, and Jürgen Teich. “Timing analysis of a heterogeneous architecture with massively parallel processor arrays”. In: *Proceedings of the DATE Friday Workshop on Performance, Power and Predictability of Many-Core Embedded Systems (3PMCES)*. (Dresden, Germany). ECSI, Mar. 28, 2014.
- [P41]  Richard Membarth, Oliver Reiche, Frank Hannig, and Jürgen Teich. “Code generation for embedded heterogeneous architectures on Android”. In: *Proceedings of the Conference on Design, Automation and Test in Europe (DATE)*. (Dresden, Germany). European Design and Automation Association (EDAA), Mar. 24–28, 2014, 86:1–86:6. DOI: 10.7873/DATE.2014.099.
- [P42] Sascha Roloff, Frank Hannig, and Jürgen Teich. “Towards actor-oriented programming on PGAS-based multicore architectures”. In: *Workshop Proceedings of the 27th International Conference on Architecture of Computing Systems (ARCS)*. (Lübeck, Germany). VDE Verlag, Feb. 25–28, 2014. ISBN: 978-3-8007-3579-2.
- [P43] Moritz Schmid, Markus Blocherer, Frank Hannig, and Jürgen Teich. “Real-time range image preprocessing on FPGAs”. In: *Proceedings of the International Conference on Reconfigurable Computing and FPGAs (ReConFig)*. (Cancun, Mexico). Dec. 9–11, 2013. ISBN: 978-1-4799-2078-5. DOI: 10.1109/ReConFig.2013.6732325.
- [P44] Éricles R. Sousa, Alexandru Tanase, Frank Hannig, and Jürgen Teich. “A prototype of an adaptive computer vision algorithm on an MPSoC architecture”. In: *Proceedings of the Conference on Design and Architectures for Signal and Image Processing (DASIP)*. (Cagliari, Italy). IEEE, Oct. 8–10, 2013, pp. 353–354. ISBN: 979-1-09-227901-6.
- [P45] Éricles R. Sousa, Alexandru Tanase, Frank Hannig, and Jürgen Teich. “Accuracy and performance analysis of harris corner computation on tightly-coupled processor arrays”. In: *Proceedings of the Conference on Design and Architectures for Signal and Image Processing (DASIP)*. (Cagliari, Italy). IEEE, Oct. 8–10, 2013, pp. 88–95. ISBN: 979-1-09-227901-6.

## A. BIBLIOGRAPHY

---

- [P46] Srinivas Boppu, Frank Hannig, and Jürgen Teich. “Loop program mapping and compact code generation for programmable hardware accelerators”. In: *Proceedings of the 24th IEEE International Conference on Application-specific Systems, Architectures and Processors (ASAP)*. (Washington, DC, USA). IEEE, June 5–7, 2013, pp. 10–17. ISBN: 978-1-4799-0493-8. DOI: 10.1109/ASAP.2013.6567544.
- [P47] Sascha Roloff, Andreas Weichslgartner, Jan Heißwolf, Frank Hannig, and Jürgen Teich. “NoC simulation in heterogeneous architectures for PGAS programming model”. In: *Proceedings of the 16th International Workshop on Software and Compilers for Embedded Systems (M-SCOPES)*. (St. Goar, Germany). ACM, June 19–21, 2013, pp. 77–85. ISBN: 978-1-4503-2142-6. DOI: 10.1145/2463596.2463606.
- [P48] Jürgen Teich, Alexandru Tanase, and Frank Hannig. “Symbolic parallelization of loop programs for massively parallel processor arrays”. In: *Proceedings of the 24th IEEE International Conference on Application-specific Systems, Architectures and Processors (ASAP)*. (Washington, DC, USA). **Best Paper Award**. IEEE, June 5–7, 2013, pp. 1–9. ISBN: 978-1-4799-0493-8. DOI: 10.1109/ASAP.2013.6567543.
- [P49] Srinivas Boppu, Vahid Lari, Frank Hannig, and Jürgen Teich. “Transactor-based prototyping of heterogeneous multiprocessor system-on-chip architectures”. In: *Proceedings of the Synopsys Users Group Conference (SNUG)*. (Munich, Germany). May 14, 2013.
- [P50] Frank Hannig, Moritz Schmid, Vahid Lari, Srinivas Boppu, and Jürgen Teich. “System integration of tightly-coupled processor arrays using reconfigurable buffer structures”. In: *Proceedings of the ACM International Conference on Computing Frontiers (CF)*. (Ischia, Italy). ACM, May 14–16, 2013, 2:1–2:4. ISBN: 978-1-4503-2053-5. DOI: 10.1145/2482767.2482770.
- [P51] Éricles R. Sousa, Alexandru Tanase, Vahid Lari, Frank Hannig, Jürgen Teich, Johny Paul, Walter Stechele, Manfred Kröhnert, and Tamim Asfour. “Acceleration of optical flow computations on tightly-coupled processor arrays”. In: *Proceedings of the 25th Workshop on Parallel Systems and Algorithms (PARS)*. (Erlangen, Germany). Vol. 30. Mitteilungen – Gesellschaft für Informatik e. V., Parallel-Algorithmen und Rechnerstrukturen. Gesellschaft für Informatik e. V., Apr. 11–12, 2013, pp. 80–89.
- [P52] Frank Hannig. “Resource-aware computing on domain-specific accelerators”. In: *Proceedings of the 10th Workshop on Optimizations for DSP and Embedded Systems (ODES)*. (Shenzhen, China). **Keynote Speech**. ACM, Feb. 24, 2013, p. 35. ISBN: 978-1-4503-1905-8. DOI: 10.1145/2443608.2443616.
- [P53] Richard Membarth, Frank Hannig, Jürgen Teich, and Harald Köstler. “Towards domain-specific computing for stencil codes in HPC”. In: *Proceedings of the 2nd International Workshop on Domain-Specific Languages and High-Level Frameworks for High Performance Computing (WOLFHPC)*. (Salt Lake City, UT, USA). Nov. 16, 2012, pp. 1133–1138. ISBN: 978-1-4673-6218-4. DOI: 10.1109/SC.Companion.2012.136.
- [P54] Shravan Muddasani, Srinivas Boppu, Frank Hannig, Boris Kuzmin, Vahid Lari, and Jürgen 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.
- [P55] Michael Gerndt, Frank Hannig, Andreas Herkersdorf, Andreas Hollmann, Marcel Meyer, Sascha Roloff, Josef Weidendorfer, Thomas Wild, and Aurang Zaib. “An integrated simulation framework for invasive computing”. In: *Proceedings of the Forum on Specification and Design Languages (FDL)*. (Vienna, Austria). IEEE, Sept. 18–20, 2012, pp. 209–216. ISBN: 978-1-4673-1240-0.

- [P56] Richard Membarth, Frank Hannig, Jürgen Teich, Mario Körner, and Wieland Eckert. “Mastering software variant explosion for gpu accelerators”. In: *Proceedings of the International Workshop on Algorithms, Models and Tools for Parallel Computing on Heterogeneous Platforms (HeteroPar) in Euro-Par 2012: Parallel Processing Workshops*. (Rhodes Island, Greece). Ed. by Ioannis Caragiannis, Michael Alexander, Rosa Maria Badia, Mario Cannataro, Alexandru Costan, Marco Danelutto, Frédéric Desprez, Bettina Krammer, Julio Sahuquillo, Stephen L. Scott, and Josef Weidendorfer. Vol. 7640. Lecture Notes in Computer Science (LNCS). Springer, Aug. 27–27, 2012, pp. 123–132. ISBN: 978-3-642-36948-3. DOI: 10.1007/978-3-642-36949-0\_15.
- [P57] Vahid Lari, Shravan Muddasani, Srinivas Boppu, Frank Hannig, and Jürgen 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.
- [P58] Sascha Roloff, Frank Hannig, and Jürgen Teich. “Simulation of resource-aware applications on heterogeneous architectures”. In: *Proceedings of ACACES 2012 Poster Abstracts: Advanced Computer Architecture and Compilation for Embedded Systems*. (Fiuggi, Italy). Academia Press, Ghent, July 8–14, 2012, pp. 127–130. ISBN: 978-90-382-1987-5.
- [P59] Alexandru Tanase, Frank Hannig, and Jürgen Teich. “Symbolic loop parallelization of static control programs”. In: *Proceedings of ACACES 2012 Poster Abstracts: Advanced Computer Architecture and Compilation for Embedded Systems*. (Fiuggi, Italy). Academia Press, Ghent, July 8–14, 2012, pp. 33–36. ISBN: 978-90-382-1987-5.
- [P60] Richard Membarth, Frank Hannig, Jürgen Teich, Mario Körner, and Wieland Eckert. “Automatic optimization of in-flight memory transactions for GPU accelerators based on a domain-specific language for medical imaging”. In: *Proceedings of the 11th International Symposium on Parallel and Distributed Computing (ISPDC)*. (Munich, Germany). IEEE, June 25–29, 2012, pp. 211–218. DOI: 10.1109/ISPDC.2012.36.
- [P61] Richard Membarth, Frank Hannig, Jürgen Teich, Mario Körner, and Wieland Eckert. “Generating device-specific GPU code for local operators in medical imaging”. In: *Proceedings of the 26th IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. (Shanghai, China). IEEE, May 21–25, 2012, pp. 569–581. ISBN: 978-0-7695-4675-9. DOI: 10.1109/IPDPS.2012.59.
- [P62] Sascha Roloff, Frank Hannig, and Jürgen 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. ISBN: 978-1-4503-1336-0. DOI: 10.1145/2236576.2236582.
- [P63] Moritz Schmid, Frank Hannig, and Jürgen Teich. “Power management strategies for serial RapidIO endpoints in FPGAs”. In: *Proceedings of the 20th Annual IEEE International Symposium on Field-Programmable Custom Computing Machines (FCCM)*. (Toronto, Canada). **HiPEAC Paper Award**. IEEE, Apr. 29–May 1, 2012, pp. 101–108. ISBN: 978-0-7695-4699-5. DOI: 10.1109/FCCM.2012.26.
- [P64] Richard Membarth, Jan-Hugo Lupp, Frank Hannig, Jürgen Teich, Mario Körner, and Wieland Eckert. “Dynamic task-scheduling and resource management for GPU accelerators in medical imaging”. In: *Proceedings of the 25th International Conference on Architecture of Computing Systems (ARCS)*. (Munich, Germany). Ed. by Andreas Herkersdorf, Kay Römer, and Uwe Brinkschulte. Vol. 7179. Lecture Notes in Computer Science (LNCS). Springer, Feb. 28–Mar. 2, 2012, pp. 147–159. ISBN: 978-3-642-28292-8. DOI: 10.1007/978-3-642-28293-5\_13.

## A. BIBLIOGRAPHY

---

- [P65] Sascha Roloff, Frank Hannig, and Jürgen 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.
- [P66] Srinivas Boppu, Frank Hannig, Jürgen Teich, and Roberto Perez-Andrade. “Towards symbolic run-time reconfiguration in tightly-coupled processor arrays”. In: *Proceedings of the International Conference on Reconfigurable Computing and FPGAs (ReConFig)*. (Cancun, Mexico). IEEE Computer Society, Nov. 30–Dec. 2, 2011, pp. 392–397. ISBN: 978-1-4577-1734-5. DOI: 10.1109/ReConFig.2011.91.
- [P67] Vahid Lari, Andriy Narovlyanskyy, Frank Hannig, and Jürgen Teich. “Decentralized dynamic resource management support for massively parallel processor arrays”. In: *Proceedings of the 22nd IEEE International Conference on Application-specific Systems, Architectures and Processors (ASAP)*. (Santa Monica, CA, USA). IEEE Computer Society, Sept. 11–14, 2011, pp. 87–94. ISBN: 978-1-4577-1291-3. DOI: 10.1109/ASAP.2011.6043240.
- [P68] Georgia Kouveli, Frank Hannig, Jan-Hugo Lupp, and Jürgen Teich. “Towards resource-aware programming on Intel’s single-chip cloud computer processor”. In: *3rd Many-core Applications Research Community (MARC) Symposium*. (Ettlingen, Germany). Vol. 7598. KIT Scientific Reports. KIT Scientific Publishing, July 5–6, 2011, pp. 111–114. ISBN: 978-3-86644-717-2.
- [P69] Frank Hannig, Sascha Roloff, Gregor Snelting, Jürgen Teich, and Andreas Zwinkau. “Resource-aware programming and simulation of MPSoC architectures through extension of X10”. In: *Proceedings of the 14th International Workshop on Software and Compilers for Embedded Systems (SCOPES)*. (St. Goar, Germany). ACM Press, June 27–28, 2011, pp. 48–55. ISBN: 978-1-4503-0763-5. DOI: 10.1145/1988932.1988941.
- [P70] Richard Membarth, Frank Hannig, Jürgen Teich, Mario Körner, and Wieland Eckert. “Frameworks for GPU accelerators: A comprehensive evaluation using 2d/3d image registration”. In: *Proceedings of the 9th IEEE Symposium on Application Specific Processors (SASP)*. (San Diego, CA, USA). June 5–6, 2011, pp. 78–81. ISBN: 978-1-4577-1211-1. DOI: 10.1109/SASP.2011.5941083.
- [P71] Vahid Lari, Frank Hannig, and Jürgen Teich. “Distributed resource reservation in massively parallel processor arrays”. In: *Proceedings of the International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*. (Anchorage, AK, USA). IEEE Computer Society, May 16–17, 2011, pp. 318–321. ISBN: 978-0-7695-4385-7. DOI: 10.1109/IPDPS.2011.157.
- [P72] Richard Membarth, Frank Hannig, Jürgen Teich, Mario Körner, and Wieland Eckert. “Frameworks for multi-core architectures: A comprehensive evaluation using 2d/3d image registration”. In: *Proceedings of the 24th International Conference on Architecture of Computing Systems (ARCS)*. (Lake Como, Italy). Vol. 6566. Lecture Notes in Computer Science (LNCS). Springer, Feb. 22–25, 2011, pp. 62–73. ISBN: 978-3-642-19136-7. DOI: 10.1007/978-3-642-19137-4\_6.
- [P73] Richard Membarth, Frank Hannig, Jürgen Teich, Gerhard Litz, and Heinz Hornegger. “Detector defect correction of medical images on graphics processors”. In: *Proceedings of SPIE Medical Imaging*. (Lake Buena Vista, FL, USA). Vol. 7962. Feb. 12–17, 2011, pp. 79624M 1–12. DOI: 10.1117/12.877656.
- [P74] Frank Hannig, Moritz Schmid, Jürgen Teich, and Heinz Hornegger. “A deeply pipelined and parallel architecture for denoising medical images”. In: *Proceedings of the IEEE International Conference on Field Programmable Technology (FPT)*. (Beijing, China). IEEE, Dec. 8–10, 2010, pp. 485–490. ISBN: 978-1-4244-8982-4. DOI: 10.1109/FPT.2010.5681464.

- [P75] Tom Vander Aa, Praveen Raghavan, Scott Mahlke, Bjorn De Sutter, Aviral Shrivastava, and Frank Hannig. "Compilation techniques for CGRAs: exploring all parallelization approaches". In: *Proceedings of the International Conference on Hardware-Software Codesign and System Synthesis (CODES+ISSS)*. (Scottsdale, AZ, USA). ACM, Oct. 24–29, 2010, pp. 185–186. ISBN: 978-1-60558-905-3. DOI: 10.1145/1878961.1878995.
- [P76] Hritam Dutta, Frank Hannig, Moritz Schmid, and Joachim Keinert. "Modeling and synthesis of communication subsystems for loop accelerator pipelines". In: *Proceedings of the 21st IEEE International Conference on Application-specific Systems, Architectures and Processors (ASAP)*. (Rennes, France). IEEE Computer Society, July 7–9, 2010, pp. 125–132. ISBN: 978-1-4244-6967-3. DOI: 10.1109/ASAP.2010.5540760.
- [P77] Richard Membarth, Frank Hannig, Jürgen Teich, Mario Körner, and Wieland Eckert. "Comparison of parallelization frameworks for shared memory multi-core architectures". In: *Proceedings of the Embedded World Conference*. (Nuremberg, Germany). Mar. 3–5, 2010.
- [P78] Moritz Schmid, Frank Hannig, Jürgen Teich, Ralf Diefenbach, Hartmut Pettendorf, and Heinz Hornegger. "Discourse on extending embedded medical image processing systems using the high speed serial RapidIO interconnect". In: *Proceedings of the Embedded World Conference*. (Nuremberg, Germany). Mar. 3–5, 2010.
- [P79] Amouri Abdulazim, Farhadur Arifin, Frank Hannig, and Jürgen Teich. "FPGA implementation of an invasive computing architecture". In: *Proceedings of the IEEE International Conference on Field Programmable Technology (FPT)*. (Sydney, Australia). IEEE, Dec. 9–11, 2009, pp. 135–142. ISBN: 978-1-4244-4376-5. DOI: 10.1109/FPT.2009.5377633.
- [P80] Farhadur Arifin, Richard Membarth, Amouri Abdulazim, Frank Hannig, and Jürgen Teich. "FSM-controlled architectures for linear invasion". In: *Proceedings of the 17th IFIP/IEEE International Conference on Very Large Scale Integration (VLSI-SoC)*. (Florianópolis, Brazil). IEEE, Oct. 12–14, 2009, pp. 59–64. ISBN: 978-1-4577-0237-2. DOI: 10.1109/VLSISOC.2009.6041331.
- [P81] Vahid Lari, Frank Hannig, and Jürgen Teich. "System integration of tightly-coupled reconfigurable processor arrays and evaluation of buffer size effects on their performance". In: *Proceedings of the 4th International Symposium on Embedded Multicore Systems-on-Chip (MCSoc)*. (Vienna, Austria). IEEE Computer Society, Sept. 22–25, 2009, pp. 528–534. ISBN: 978-1-4244-4923-1. DOI: 10.1109/ICPPW.2009.72.
- [P82] Hritam Dutta, Jiali Zhai, Frank Hannig, and Jürgen Teich. "Impact of loop tiling on the controller logic of hardware acceleration engines". In: *Proceedings of the 20th IEEE International Conference on Application-specific Systems, Architectures and Processors (ASAP)*. (Boston, MA, USA). IEEE Computer Society, July 7–9, 2009, pp. 161–168. ISBN: 978-0-7695-3732-0. DOI: 10.1109/ASAP.2009.21.
- [P83] Richard Membarth, Frank Hannig, Hritam Dutta, and Jürgen Teich. "Efficient mapping of multiresolution image filtering algorithms on graphics processors". In: *Proceedings of the 9th International Workshop on Systems, Architectures, Modeling, and Simulation (SAMOS)*. (Island of Samos, Greece). Ed. by Koen Bertels, Nikitas Dimopoulos, Christina Silvano, and Stephan Wong. Vol. 5657. Lecture Notes in Computer Science (LNCS). Springer, July 20–23, 2009, pp. 277–288. ISBN: 978-3-642-03137-3. DOI: 10.1007/978-3-642-03138-0\_31.
- [P84] Richard Membarth, Frank Hannig, Hritam Dutta, and Jürgen Teich. "Optimization flow for algorithm mapping on graphics cards". In: *Proceedings of ACACES 2009 Poster Abstracts: Advanced Computer Architecture and Compilation for Embedded Systems*. (Terrassa, Spain). Academia Press, Ghent, July 12–18, 2009, pp. 229–232. ISBN: 978-90-382-1467-2.



## A. BIBLIOGRAPHY

---

- [P85] Richard Membarth, Philipp Kutzer, Hritam Dutta, Frank Hannig, and Jürgen Teich. “Acceleration of multiresolution imaging algorithms: A comparative study”. In: *Proceedings of the 20th IEEE International Conference on Application-specific Systems, Architectures and Processors (ASAP)*. (Boston, MA, USA). IEEE Computer Society, July 7–9, 2009, pp. 211–214. ISBN: 978-0-7695-3732-0. DOI: 10.1109/ASAP.2009.8.
- [P86] Joachim Keinert, Hritam Dutta, Frank Hannig, Christian Haubelt, and Jürgen Teich. “Model-based synthesis and optimization of static multi-rate image processing algorithms”. In: *Proceedings of the Conference on Design, Automation and Test in Europe (DATE)*. (Nice, France). IEEE Computer Society, Apr. 20–24, 2009, pp. 135–140. ISBN: 978-1-4244-3781-8.
- [P87] Hritam Dutta, Frank Hannig, and Jürgen Teich. “Performance matching of hardware acceleration engines for heterogeneous MPSoC using modular performance analysis”. In: *Proceedings of the 22nd International Conference on Architecture of Computing Systems (ARCS)*. (Delft, The Netherlands). Vol. 5455. Lecture Notes in Computer Science (LNCS). Springer, Mar. 10–13, 2009, pp. 233–245. ISBN: 978-3-642-00453-7. DOI: 10.1007/978-3-642-00454-4\_23.
- [P88] Frank Hannig, Hritam Dutta, and Jürgen Teich. “Parallelization approaches for hardware accelerators – loop unrolling versus loop partitioning”. In: *Proceedings of the 22nd International Conference on Architecture of Computing Systems (ARCS)*. (Delft, The Netherlands). Vol. 5455. Lecture Notes in Computer Science (LNCS). Springer, Mar. 10–13, 2009, pp. 16–27. ISBN: 978-3-642-00453-7. DOI: 10.1007/978-3-642-00454-4\_5.
- [P89] Sven Eisenhardt, Thomas Schweizer, Julio A. de Oliveira Filho, Tobias Oppold, Wolfgang Rosenstiel, Alexander Thomas, Jürgen Becker, Frank Hannig, Dmitrij Kissler, Hritam Dutta, Jürgen Teich, Heiko Hinkelmann, Peter Zipf, and Manfred Glesner. “SPP1148 booth: coarse-grained reconfiguration”. In: *Proceedings of the International Conference on Field Programmable Logic and Applications (FPL)*. (Heidelberg, Germany). Sept. 8–10, 2008, p. 349. ISBN: 978-1-4244-1961-6. DOI: 10.1109/FPL.2008.4629957.
- [P90] Dmitrij Kissler, Andreas Strawetz, Frank Hannig, and Jürgen Teich. “Power-efficient reconfiguration control in coarse-grained dynamically reconfigurable architectures”. In: *Proceedings of the 18th International Workshop on Power and Timing Modeling, Optimization and Simulation (PATMOS)*. (Lisbon, Portugal). Vol. 5349. Lecture Notes in Computer Science (LNCS). Springer, Sept. 10–12, 2008, pp. 307–317. ISBN: 978-3-540-95947-2. DOI: 10.1007/978-3-540-95948-9\_31.
- [P91] Rainer Schaffer, Renate Merker, Frank Hannig, and Jürgen Teich. “Utilization of all levels of parallelism in a processor array with subword parallelism”. In: *Proceedings of the 11th Euromicro Conference on Digital System Design (DSD)*. (Parma, Italy). IEEE, Sept. 3–5, 2008, pp. 391–398. ISBN: 978-0-7695-3277-6. DOI: 10.1109/DSD.2008.24.
- [P92] Christophe Wolinski, Krzysztof Kuchcinski, Jürgen Teich, and Frank Hannig. “Area and reconfiguration time minimization of the communication network in regular 2d reconfigurable architectures”. In: *Proceedings of the International Conference on Field Programmable Logic and Applications (FPL)*. (Heidelberg, Germany). IEEE, Sept. 8–10, 2008, pp. 391–396. ISBN: 978-1-4244-1961-6. DOI: 10.1109/FPL.2008.4629969.
- [P93] Christophe Wolinski, Krzysztof Kuchcinski, Jürgen Teich, and Frank Hannig. “Communication network reconfiguration overhead optimization in programmable processor array architectures”. In: *Proceedings of the 11th Euromicro Conference on Digital System Design (DSD)*. (Parma, Italy). IEEE, Sept. 3–5, 2008, pp. 345–352. ISBN: 978-0-7695-3277-6. DOI: 10.1109/DSD.2008.1.



- [P94] Hritam Dutta, Frank Hannig, and Jürgen Teich. “PARO: A design tool for automatic generation of hardware accelerators”. In: *Proceedings of ACACES 2008 Poster Abstracts: Advanced Computer Architecture and Compilation for Embedded Systems*. (L’Aquila, Italy). Academia Press, Ghent, July 13–19, 2008, pp. 317–320. ISBN: 978-90-382-1288-3.
- [P95] Christophe Wolinski, Krzysztof Kuchcinski, Jürgen Teich, and Frank Hannig. “Optimization of routing and reconfiguration overhead in programmable processor array architectures”. In: *Proceedings of the 16th IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM)*. (Palo Alto, CA, USA). IEEE Computer Society, Apr. 14–15, 2008, pp. 306–309. ISBN: 978-0-7695-3307-0. DOI: 10.1109/FCCM.2008.16.
- [P96] Frank Hannig, Holger Ruckdeschel, Hritam Dutta, and Jürgen Teich. “PARO: synthesis of hardware accelerators for multi-dimensional dataflow-intensive applications”. In: *Proceedings of the Fourth International Workshop on Applied Reconfigurable Computing (ARC)*. (London, United Kingdom). Vol. 4943. Lecture Notes in Computer Science (LNCS). Springer, Mar. 26–28, 2008, pp. 287–293. ISBN: 978-3-540-78609-2. DOI: 10.1007/978-3-540-78610-8\_30.
- [P97] Frank Hannig, Holger Ruckdeschel, and Jürgen Teich. “The PAULA language for designing multi-dimensional dataflow-intensive applications”. In: *Proceedings of the GI/ITG/GMM-Workshop – Methoden und Beschreibungssprachen zur Modellierung und Verifikation von Schaltungen und Systemen*. (Freiburg, Germany). Shaker, Mar. 3–5, 2008, pp. 129–138. ISBN: 978-3-8322-6962-3.
- [P98] Frank Hannig, Hritam Dutta, Holger Ruckdeschel, and Jürgen Teich. “Quantitative evaluation of behavioral synthesis approaches for reconfigurable devices”. In: *Proceedings of the 2nd HiPEAC Workshop on Reconfigurable Computing (WRC)*. (Gothenburg, Sweden). Jan. 27, 2008, pp. 73–82.
- [P99] Hritam Dutta, Frank Hannig, Alexey Kupriyanov, Dmitrij Kissler, Jürgen Teich, Rainer Schaffer, Sebastian Siegel, Renate Merker, and Bernard Pottier. “Massively parallel processor architectures: A co-design approach”. In: *Proceedings of the 3rd International Workshop on Reconfigurable Communication Centric System-on-Chips (ReCoSoC)*. (Montpellier, France). Univ. Montpellier II, June 18–20, 2007, pp. 61–68. ISBN: 2-9517461-3-X.
- [P100] Jürgen Teich, Frank Hannig, Holger Ruckdeschel, Hritam Dutta, Dmitrij Kissler, and Andrej Stravet. “A unified retargetable design methodology for dedicated and re-programmable multiprocessor arrays: case study and quantitative evaluation”. In: *Proceedings of the International Conference on Engineering of Reconfigurable Systems and Algorithms (ERSA), Invited paper*. (Las Vegas, NV, USA). CSREA Press, June 25–28, 2007, pp. 14–24. ISBN: 1-60132-026-4.
- [P101] Alexey Kupriyanov, Dmitrij Kissler, Frank Hannig, and Jürgen Teich. “Efficient event-driven simulation of parallel processor architectures”. In: *Proceedings of the 10th International Workshop on Software and Compilers for Embedded Systems (SCOPES)*. (Nice, France). ACM Press, Apr. 20, 2007, pp. 71–80. DOI: 10.1145/1269843.1269854.
- [P102] Alexey Kupriyanov, Frank Hannig, Dmitrij Kissler, Jürgen Teich, Julien Lallet, Olivier Sentieys, and Sébastien Pillement. “Modeling of interconnection networks in massively parallel processor architectures”. In: *Proceedings of the 20th International Conference on Architecture of Computing Systems (ARCS)*. (Zurich, Switzerland). Ed. by Paul Lukowicz, Lothar Thiele, and Gerhard Tröster. Vol. 4415. Lecture Notes in Computer Science (LNCS). Springer, Mar. 12–15, 2007, pp. 268–282. ISBN: 978-3-540-71267-1. DOI: 10.1007/978-3-540-71270-1\_20.

## A. BIBLIOGRAPHY

---

- [P103] Dmitrij Kissler, Frank Hannig, Alexey Kupriyanov, and Jürgen Teich. “A highly parameterizable parallel processor array architecture”. In: *Proceedings of the IEEE International Conference on Field Programmable Technology (FPT)*. (Bangkok, Thailand). IEEE, Dec. 13–15, 2006, pp. 105–112. ISBN: 0-7803-9728-2. DOI: 10.1109/FPT.2006.270293.
- [P104] Dmitrij Kissler, Frank Hannig, Alexey Kupriyanov, and Jürgen Teich. “Hardware cost analysis for weakly programmable processor arrays”. In: *Proceedings of the International Symposium on System-on-Chip (SoC)*. (Tampere, Finland). IEEE, Nov. 14–16, 2006, pp. 179–182. ISBN: 1-4244-0621-8. DOI: 10.1109/ISSOC.2006.321996.
- [P105] Sebastian Siegel, Renate Merker, Frank Hannig, and Jürgen Teich. “Communication-conscious mapping of regular nested loop programs onto massively parallel processor arrays”. In: *Proceedings of the 18th International Conference on Parallel and Distributed Computing and Systems (PDCS)*. (Dallas, TX, USA). ACTA Press, Nov. 13–15, 2006, pp. 71–76. ISBN: 0-88986-638-4.
- [P106] Hritam Dutta, Frank Hannig, and Jürgen Teich. “Hierarchical partitioning for piecewise linear algorithms”. In: *Proceedings of the 5th International Conference on Parallel Computing in Electrical Engineering (PARELEC)*. (Bialystok, Poland). IEEE Computer Society, Sept. 13–17, 2006, pp. 153–160. ISBN: 978-0-7695-2554-9. DOI: 10.1109/PARELEC.2006.43.
- [P107] Hritam Dutta, Frank Hannig, Jürgen Teich, Benno Heigl, and Heinz Hornegger. “A design methodology for hardware acceleration of adaptive filter algorithms in image processing”. In: *Proceedings of the 17th IEEE International Conference on Application-specific Systems, Architectures and Processors (ASAP)*. (Steamboat Springs, CO, USA). IEEE, Sept. 11–13, 2006, pp. 331–337. ISBN: 978-0-7695-2682-9. DOI: 10.1109/ASAP.2006.4.
- [P108] Dmitrij Kissler, Frank Hannig, Alexey Kupriyanov, and Jürgen Teich. “A dynamically reconfigurable weakly programmable processor array architecture template”. In: *Proceedings of the 2nd International Workshop on Reconfigurable Communication Centric System-on-Chips (ReCoSoC)*. (Montpellier, France). July 3–5, 2006, pp. 31–37. ISBN: 2-9517461-2-1.
- [P109] Dmitrij Kissler, Alexey Kupriyanov, Frank Hannig, Dirk Koch, and Jürgen Teich. “A generic framework for rapid prototyping of system-on-chip designs”. In: *Proceedings of International Conference on Computer Design (CDES)*. (Las Vegas, NV, USA). June 26–29, 2006, pp. 189–195. ISBN: 1-60132-009-4.
- [P110] Hritam Dutta, Frank Hannig, and Jürgen Teich. “Controller synthesis for mapping partitioned programs on array architectures”. In: *Proceedings of the 19th International Conference on Architecture of Computing Systems (ARCS)*. (Frankfurt am Main, Germany). Ed. by Werner Grass, Bernhard Sick, and Klaus Waldschmidt. Vol. 3894. Lecture Notes in Computer Science (LNCS). Springer, Mar. 13–16, 2006, pp. 176–191. ISBN: 3-540-32765-7. DOI: 10.1007/11682127\_13.
- [P111] Alexey Kupriyanov, Frank Hannig, Dmitrij Kissler, Jürgen Teich, Rainer Schaffer, and Renate Merker. “An architecture description language for massively parallel processor architectures”. In: *Proceedings of the GLITG/GMM-Workshop – Methoden und Beschreibungssprachen zur Modellierung und Verifikation von Schaltungen und Systemen*. (Dresden, Germany). Shaker, Feb. 20–22, 2006, pp. 11–20.
- [P112] Hritam Dutta, Frank Hannig, and Jürgen Teich. “Mapping of nested loop programs onto massively parallel processor arrays with memory and I/O constraints”. In: *Proceedings of the 6th International Heinz Nixdorf Symposium, New Trends in Parallel & Distributed Computing*. (Paderborn, Germany). Ed. by Friedhelm Meyer auf der Heide and Burkhard Monien. Vol. 181. HNI-Verlagsschriftenreihe. Heinz Nixdorf Institut, Universität Paderborn, Jan. 17–18, 2006, pp. 97–119. ISBN: 3-939350-00-1.

- [P113] Holger Ruckdeschel, Hritam Dutta, Frank Hannig, and Jürgen Teich. “Automatic FIR filter generation for FPGAs”. In: *Embedded Computer Systems: Architectures, Modeling, and Simulation, 5th International Workshop, SAMOS, Proceedings*. (Island of Samos, Greece). Ed. by Timo D. Härmäläinen, Andy D. Pimentel, Jarmo Takala, and Stamatis Vassiliadis. Vol. 3553. Lecture Notes in Computer Science (LNCS). Springer, July 18–20, 2005, pp. 51–61. ISBN: 3-540-26969-X. DOI: 10.1007/11512622\_7.
- [P114] Thomas Schlichter, Christian Haubelt, Frank Hannig, and Jürgen Teich. “Using symbolic feasibility tests during design space exploration of heterogeneous multi-processor systems”. In: *Proceedings of the 16th IEEE International Conference on Application-specific Systems, Architectures, and Processors (ASAP)*. (Island of Samos, Greece). IEEE Computer Society, July 23–25, 2005, pp. 9–14. ISBN: 0-7695-2407-9. DOI: 10.1109/ASAP.2005.64.
- [P115] Frank Hannig, Hritam Dutta, Alexey Kupriyanov, Jürgen Teich, Rainer Schaffer, Sebastian Siegel, Renate Merker, Ronan Keryell, Bernard Pottier, Daniel Chillet, Daniel Ménard, and Olivier Sentieys. “Co-design of massively parallel embedded processor architectures”. In: *Proceedings of the first International Workshop on Reconfigurable Communication Centric System-on-Chips (ReCoSoC)*. (Montpellier, France). Univ. Montpellier II, June 27–29, 2005, pp. 27–34. ISBN: 2-9517461-1-3.
- [P116] Frank Hannig and Jürgen Teich. “Output serialization for FPGA-based and coarse-grained processor arrays”. In: *Proceedings of the International Conference on Engineering of Reconfigurable Systems and Algorithms (ERSA)*. (Las Vegas, NV, USA). CSREA Press, June 27–30, 2005, pp. 78–84. ISBN: 1-932415-74-2.
- [P117] Jan van der Veen, Sándor Fekete, Mateusz Majer, Ali Ahmadinia, Christophe Bobda, Frank Hannig, and Jürgen Teich. “Defragmenting the module layout of a partially reconfigurable device”. In: *Proceedings of the International Conference on Engineering of Reconfigurable Systems and Algorithms (ERSA)*. (Las Vegas, NV, USA). **Distinguished Paper**. CSREA Press, June 27–30, 2005, pp. 92–101. ISBN: 1-932415-74-2.
- [P118] Frank Hannig and Jürgen Teich. “Dynamic piecewise linear/regular algorithms”. In: *Proceedings of the Fourth International Conference on Parallel Computing in Electrical Engineering (PARELEC)*. (Dresden, Germany). IEEE Computer Society, Sept. 7–10, 2004, pp. 79–84. ISBN: 0-7695-2080-4. DOI: 10.1109/PCEE.2004.28.
- [P119] Frank Hannig and Jürgen Teich. “Resource constrained and speculative scheduling of an algorithm class with run-time dependent conditionals”. In: *Proceedings of the 15th IEEE International Conference on Application-specific Systems, Architectures, and Processors (ASAP)*. (Galveston, TX, USA). IEEE Computer Society, Sept. 27–29, 2004, pp. 17–27. ISBN: 0-7695-2226-2. DOI: 10.1109/ASAP.2004.1342455.
- [P120] Alexey Kupriyanov, Frank Hannig, and Jürgen Teich. “Automatic and optimized generation of compiled high-speed RTL simulators”. In: *Proceedings of Workshop on Compilers and Tools for Constrained Embedded Systems (CTCES)*. (Washington, DC, USA). Sept. 22, 2004.
- [P121] Alexey Kupriyanov, Frank Hannig, and Jürgen Teich. “High-speed event-driven RTL compiled simulation”. In: *Computer Systems: Architectures, Modeling, and Simulation, 4th International Samos Workshop (SAMOS), Proceedings*. (Island of Samos, Greece). Ed. by Andy D. Pimentel and Stamatis Vassiliadis. Vol. 3133. Lecture Notes in Computer Science (LNCS). Springer, July 19–21, 2004, pp. 519–529. ISBN: 3-540-22377-0. DOI: 10.1007/b98714.

## A. BIBLIOGRAPHY

---

- [P122] Frank Hannig, Hritam Dutta, and Jürgen Teich. “Regular mapping for coarse-grained reconfigurable architectures”. In: *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*. (Montréal, Québec, Canada). Vol. V. IEEE Signal Processing Society, May 17–21, 2004, pp. 57–60. ISBN: 0-7803-8484-9. DOI: 10.1109/ICASSP.2004.1327046.
- [P123] Frank Hannig, Hritam Dutta, and Jürgen Teich. “Mapping of regular nested loop programs to coarse-grained reconfigurable arrays – constraints and methodology”. In: *Proceedings of the 18th International Parallel and Distributed Processing Symposium (IPDPS)*. (Santa Fe, NM, USA). IEEE Computer Society, Apr. 26–30, 2004. ISBN: 0-7695-2132-0. DOI: 10.1109/IPDPS.2004.1303132.
- [P124] Frank Hannig and Jürgen Teich. “Energy estimation of nested loop programs”. In: *Proceedings 14th Annual ACM Symposium on Parallel Algorithms and Architectures (SPAA)*. (Winnipeg, Manitoba, Canada). ACM Press, Aug. 10–13, 2002, pp. 149–150. ISBN: 1-58113-529-7. DOI: 10.1145/564870.564895.
- [P125] Frank Hannig and Jürgen Teich. “Energy estimation for piecewise regular processor arrays”. In: *Proceedings of the Second International Samos Workshop on Systems, Architectures, Modeling, and Simulation (SAMOS)*. (Island of Samos, Greece). July 22–25, 2002.
- [P126] Marcus Bednara, Frank Hannig, and Jürgen Teich. “Boundary control: A new distributed control architecture for space-time transformed (VLSI) processor arrays”. In: *Proceedings of the 35th IEEE Asilomar Conference on Signals, Systems, and Computers*. (Pacific Grove, CA, USA). Vol. 2. IEEE Computer Society, Nov. 4–7, 2001, pp. 468–474. DOI: 10.1109/ACSSC.2001.986970.
- [P127] Frank Hannig and Jürgen Teich. “Design space exploration for massively parallel processor arrays”. In: *Proceedings of the 6th International Conference on Parallel Computing Technologies (PaCT)*. (Novosibirsk, Russia). Ed. by Victor Malyshev. Vol. 2127. Lecture Notes in Computer Science (LNCS). Springer, Sept. 3–7, 2001, pp. 51–65. ISBN: 3-540-42522-5. DOI: 10.1007/3-540-44743-1\_5.

## Theses (3)

- [T1] Frank Hannig. “Scheduling Techniques for High-Throughput Loop Accelerators”. Verlag Dr. Hut, Munich, Germany. Dissertation. Friedrich-Alexander-Universität Erlangen-Nürnberg, Germany, Aug. 11, 2009, 307 pp. ISBN: 978-3-86853-220-3.
- [T2] Frank Hannig. “Exploration von Raum- und Zeittransformationen für Algorithmen mit uniformen Datenabhängigkeiten”. Diplomarbeit. Universität Paderborn, Fachbereich Elektrotechnik und Informationstechnik, Fachgebiet Datentechnik, Oct. 31, 2000.
- [T3] Frank Hannig. “Eine Softwareumgebung für neuronale Assoziativspeicher”. Studienarbeit. Universität Paderborn, Fachbereich Elektrotechnik und Informationstechnik, Fachgebiet Schaltungstechnik, Apr. 6, 1999.

## Technical Reports and White Papers (7)

- [R1] Adrian Tate, Amir Kamil, Anshu Dubey, Armin Größlinger, Brad Chamberlain, Brice Goglin, H. Carter Edwards, Chris J. Newburn, David Padua, Didem Unat, Emmanuel Jeannot, Frank Hannig, Tobias Gysi, Hatem Ltaief, James Sexton, Jesus Labarta, John Shalf, Karl Furlinger, Kathryn O’Brien, Leonidas Linardakis, Maciej Besta, Marie-Christine Sawley, Mark Abraham, Mauro Bianco, Miquel Pericàs, Naoya Maruyama, Paul H. J. Kelly, Peter Messmer, Robert B. Ross, Romain Cledat, Satoshi Matsuoka, Thomas Schulthess, Torsten Hoefler, and Vitus

- J. Leung. *Programming Abstractions for Data Locality*. White Paper. PADAL Workshop 2014, April 28–29, Swiss National Supercomputing Center (CSCS), Lugano, Switzerland, Nov. 10, 2014, 54 pp.
- [R2] Christian Lengauer, Sven Apel, Matthias Bolten, Armin Größlinger, Frank Hannig, Harald Köstler, Ulrich Rüde, Jürgen Teich, Alexander Grebhahn, Stefan Kronawitter, Sebastian Kuckuk, Hannah Rittich, and Christian Schmitt. *ExaStencils: Advanced Stencil-Code Engineering – First Project Report*. Tech. rep. MIP-1401. Department of Computer Science and Mathematics, University of Passau, June 2014, 12 pp.
  - [R3] Alexey Kupriyanov, Frank Hannig, Dmitrij Kissler, Jürgen Teich, Julien Lallet, Olivier Sentieys, and Sébastien Pillement. *Modeling of Interconnection Networks in Massively Parallel Processor Architectures*. Tech. rep. 05–2006. Am Weichselgarten 3, 91058 Erlangen, Germany: University of Erlangen-Nuremberg, Department of CS 12, Hardware-Software-Co-Design, Aug. 15, 2006.
  - [R4] Hritam Dutta, Frank Hannig, and Jürgen Teich. *A Formal Methodology for Hierarchical Partitioning of Piecewise Linear Algorithms*. Tech. rep. 04–2006. Am Weichselgarten 3, 91058 Erlangen, Germany: University of Erlangen-Nuremberg, Department of CS 12, Hardware-Software-Co-Design, Apr. 3, 2006.
  - [R5] Alexey Kupriyanov, Frank Hannig, Dmitrij Kissler, Rainer Schaffer, and Jürgen Teich. *MAML – An Architecture Description Language for Modeling and Simulation of Processor Array Architectures, Part I*. Tech. rep. 03–2006. Am Weichselgarten 3, 91058 Erlangen, Germany: University of Erlangen-Nuremberg, Department of CS 12, Hardware-Software-Co-Design, Mar. 23, 2006.
  - [R6] Hritam Dutta, Frank Hannig, and Jürgen Teich. *Controller Synthesis for Mapping Partitioned Programs on Array Architectures*. Tech. rep. 03–2005. Am Weichselgarten 3, 91058 Erlangen, Germany: University of Erlangen-Nuremberg, Department of CS 12, Hardware-Software-Co-Design, Nov. 1, 2005.
  - [R7] Frank Hannig and Jürgen Teich. *Resource Constrained and Speculative Scheduling of Dynamic Piecewise Regular Algorithms*. Tech. rep. 01–2004. Am Weichselgarten 3, 91058 Erlangen, Germany: University of Erlangen-Nuremberg, Department of CS 12, Hardware-Software-Co-Design, June 8, 2004.



## B Image Credits



Advanced Micro Devices Inc.,

<http://www.amd.com/PublishingImages/photography/product/360px/AMD-FirePro-W9000-360W.png>



AMI GmbH,

[http://www.ami-gmbh.com/ami/img/Xilinx\\_Kintex\\_7\\_325T.jpg](http://www.ami-gmbh.com/ami/img/Xilinx_Kintex_7_325T.jpg)



Afrank99 (Wikipedia),

<https://de.wikipedia.org/w/index.php?curid=2095155>



Digi-Key Electronics,

[https://media.digikey.com/Photos/Texas%20Instr%20Photos/MFG\\_EVMK2H.jpg](https://media.digikey.com/Photos/Texas%20Instr%20Photos/MFG_EVMK2H.jpg)



Dave Gandy, \faBook from  $\LaTeX$  fontawesome package, SIL Open Font License,

<https://www.ctan.org/tex-archive/fonts/fontawesome>



Intel Corporation,

<https://www.intel.com/content/dam/www/public/us/en/images/product/badges-rwd/arria10-fpga-soc-device-chip.png>



Intel Corporation,

<https://www.intel.com/content/dam/www/public/us/en/images/product/RWD/xeon-phi-family-rwd.png>



Jarekt (Wikipedia), Public Domain,

<https://commons.wikimedia.org/w/index.php?curid=3769111>



Tam (Wikipedia),

<https://de.wikipedia.org/w/index.php?curid=2888467>



Reproduced from work created and shared by the Android Open Source Project (Wikipedia), CC BY-SA 2.5,

[https://upload.wikimedia.org/wikipedia/commons/f/fe/Nexus\\_10.png](https://upload.wikimedia.org/wikipedia/commons/f/fe/Nexus_10.png)



NVIDIA Corporation,

<http://www.nvidia.de/docs/IO/143902/tesla-k40.png>



www.llvm.org, ©Apple Inc.,

<http://llvm.org/img/DragonFull.png>



www.python.org (<https://www.python.org/community/logos/>), GPL,

<https://commons.wikimedia.org/w/index.php?curid=34991637>



Yukihiro Matsumoto, Ruby Visual Identity Team (Wikipedia), CC BY-SA 2.5,

<https://commons.wikimedia.org/w/index.php?curid=3239992>





## C Paper Reprints

From my 165 peer-reviewed publications listed in Appendix A.2 (page 60ff.), I have opted for the following 25 key contributions of my research. These form the chief part of my cumulative habilitation treatise.

### Resource-aware Computing

#### Modeling and System Simulation Papers

<b>DAC '15</b> page 87ff.	Roloff, Schafhauser, Hannig, and Teich. "Execution-driven parallel simulation of PGAS applications on heterogeneous tiled architectures"	[P24]
<b>X10 '16</b> page 93ff.	Roloff, Pöppel, Schwarzer, Wildermann, Bader, Glaß, Hannig, and Teich. "ActorX10: An actor library for X10"	[P16]
<b>ESTIMedia '17</b> page 99ff.	Roloff, Hannig, and Teich. "High performance network-on-chip simulation by interval-based timing predictions"	[P4]

#### Papers on Architecture/Compiler Co-Design of Invasive TCPAs

<b>ACM TECS '14</b> page 109ff.	Hannig, Lari, Boppu, Tanase, and Reiche. "Invasive tightly-coupled processor arrays: A domain-specific architecture/compiler co-design approach"	[J18]
<b>RSP '17</b> page 139ff.	Witterauf, Hannig, and Teich. "Constructing fast and cycle-accurate simulators for configurable accelerators using C++ templates"	[P5]
<b>Springer JSPS '14</b> page 147ff.	Teich, Tanase, and Hannig. "Symbolic mapping of loop programs onto processor arrays"	[J15]
<b>MEMOCODE '14</b> page 177ff.	Tanase, Witterauf, Teich, and Hannig. "Symbolic inner loop parallelisation for massively parallel processor arrays"	[P32]
<b>ACM TECS '17</b> page 187ff.	Tanase, Witterauf, Teich, and Hannig. "Symbolic multi-level loop mapping of loop programs for massively parallel processor arrays"	[J1]
<b>ASAP '16</b> page 215ff.	Witterauf, Tanase, Hannig, and Teich. "Modulo scheduling of symbolically tiled loops for tightly coupled processor arrays"	[P15]
<b>Springer JSPS '14</b> page 225ff.	Boppu, Hannig, and Teich. "Compact code generation for tightly-coupled processor arrays"	[J17]

## Domain-specific Computing

### Domain-specific HLS Papers




<b>ASAP '14</b> page 251ff.	Schmid, Tanase, Hannig, Teich, Bhadouria, and Ghoshal. "Domain-specific augmentations for high-level synthesis"	[P37]
<b>FPL '14</b> page 257ff.	Schmid, Apelt, Hannig, and Teich. "An image processing library for C-based high-level synthesis"	[P33]
<b>Springer JSPS '17</b> page 261ff.	Bhadouria, Tanase, Schmid, Hannig, Teich, and Ghoshal. "A novel image impulse noise removal algorithm optimized for hardware accelerators"	[J2]
<b>ASAP '17</b> page 279ff.	Özkan, Reiche, Hannig, and Teich. "Hardware design and analysis of efficient loop coarsening and border handling for image processing"	[P9]

### HIPAcc Papers

<b>IEEE TPDS '16</b> page 289ff.	Membarth, Reiche, Hannig, Teich, Körner, and Eckert. "HIPAcc: A domain-specific language and compiler for image processing"	[J9]
<b>DATE '14</b> page 305ff.	Membarth, Reiche, Hannig, and Teich. "Code generation for embedded heterogeneous architectures on Android"	[P41]
<b>CODES+ISSS '14</b> page 311ff.	Reiche, Schmid, Hannig, Membarth, and Teich. "Code generation from a domain-specific language for C-based HLS of hardware accelerators"	[P31]
<b>Elsevier JPDC '14</b> page 321ff.	Membarth, Reiche, Schmitt, Hannig, Teich, Stürmer, and Köstler. "Towards a performance-portable description of geometric multigrid algorithms using a domain-specific language"	[J12]
<b>FPL '16</b> page 333ff.	Özkan, Reiche, Hannig, and Teich. "FPGA-based accelerator design from a domain-specific language"	[P13]
<b>Springer JSPS '17</b> page 343ff.	Reiche, Özkan, Hannig, Teich, and Schmid. "Loop parallelization techniques for FPGA accelerator synthesis"	[J5]
<b>LCTES '17</b> page 369ff.	Reiche, Kobylko, Hannig, and Teich. "Auto-vectorization for image processing DSLs"	[P11]

---

## ExaStencils Papers

<b>ICCSA '14</b> page 379ff.	Schmitt, Kuckuk, Köstler, Hannig, and Teich. "An evaluation of domain-specific language technologies for code generation"	[P38 
<b>Euro-Par '14</b> page 389ff.	Lengauer, Apel, Bolten, Größlinger, Hannig, Köstler, Rüde, Teich, Grebhahn, Kronawitter, Kuckuk, Rittich, and Schmitt. "ExaStencils: Advanced stencil-code engineering"	[P35 
<b>WOLFHPC '14</b> page 401ff.	Schmitt, Kuckuk, Hannig, Köstler, and Teich. "ExaSlang: A domain-specific language for highly scalable multigrid solvers"	[P29 
<b>Springer LNCSE '16</b> page 411ff.	Schmitt, Kuckuk, Hannig, Teich, Köstler, Rüde, and Lengauer. "Systems of partial differential equations in ExaSlang"	[C1 