Proceedings of

# CGO 2016

## The 14th International Symposium on Code Generation and Optimization

*Sponsored by:*

IEEE computer society

Association for Computing Machinery

12-18 March 2016
Barcelona, Spain

The Association for Computing Machinery, Inc.
2 Penn Plaza, Suite 701
New York, NY 10121-0701

ACM ISBN: 978-1-4503-3778-6

Additional copies may be ordered prepaid from:

| | |
|---|---|
| | Phone: 1-800-342-6626 |
| ACM Order Department | (U.S.A. and Canada) |
| P.O. BOX 11405 | +1-212-626-0500 |
| Church Street Station | (All other countries) |
| New York, NY 10286-1405 | Fax: +1-212-944-1318 |
| | E-mail: acmhelp@acm.org |

Cover photo "Barcelona, Sagrada Familia by night, 2015" by Euku / Licensed under CC BY-SA 3.0 / Original at https://commons.wikimedia.org/wiki/File:Barcelona,_Sagrada_Familia_by_night,_2015.jpg

**Production:** Conference Publishing Consulting
D-94034 Passau, Germany, info@conference-publishing.com

## Message from the General Chair

On behalf of the Organizing Committee, welcome to the 14th Annual IEEE/ACM International Symposium on Code Generation and Optimization (CGO'16) held at Hotel Princesa Sofia, Barcelona, Spain between March 12th-18th, 2016. As in previous years CGO'16 is co-located with the ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP) and the International Symposium on High-Performance Computer Architecture (HPCA).

The International Symposium on Code Generation and Optimization continues to provide a premier venue bringing together researchers and practitioners working at the interface of hardware and software on a wide range of optimization and code generation techniques and related issues. The conferences spans the spectrum from purely static to fully dynamic approaches, including techniques ranging from pure software-based methods to architectural features and support.

CGO strives to expand and consolidate its position as the leading conference in this area. Therefore, we have created the CGO Compiler Cluster in addition to an attractive workshops and tutorials programme. As part of this new Compiler Cluster we have established new co-locations with two major European compiler events: The 25th International Conference on Compiler Construction (CC 2016) and 2016 European LLVM Developers' Meeting (EuroLLVM'16). CGO shares a venue with CC 2016 and EuroLLVM'16, which immediately follow after CGO, enabling attendees to experience a full week packed with exciting compiler research and community gatherings.

CGO is fully committed to further strengthen its Artifact Evaluation process. Authors of accepted papers have been invited to formally submit their supporting materials to the Artifact Evaluation process, run by a separate committee whose task is to assess how the artifacts support the work described in the papers. This submission has been voluntary and has not influenced the final decision regarding the papers. We continue to highlight papers, which have successfully passed the artifact evaluation, and have also encouraged authors to submit additional artifact documentation as supplementary material along with the camera-ready version of their papers.

CGO is a group effort and the result of the outstanding dedication and support of its entire organizing team. I want to express my gratitude towards the entire team of helpers, including the Steering Committee, the Organizing Committee, the Programme Committee and its Chairs, our partners from PPoPP, HPCA, CC and EuroLLVM, tutorial and workshop organizers, everyone at local event manager Grupo Pacifico, and the tireless local volunteers.

We also thank all our financial sponsors for their continued generous support: ACM, IEEE, the US National Science Foundation, Google, Microsoft Research, Oracle, Intel, IBM Research, SIGPLAN, and SIGMICRO.

Finally, thanks to all of you as attendees and once again, welcome to CGO!

**Björn Franke**, University of Edinburgh

# Message from the Program Chairs

On behalf of the Program Committee, we are glad to present this program for the 2016 International Symposium on Code Generation and Optimization (CGO'16). We hope you find it interesting, informative, and innovative.

This year, we accepted 25 high quality papers from the 108 submissions that the Program Committee (PC) reviewed (23% acceptance rate). The committee's expertise covered a broad range of topics including hardware, software, compiler, run-time, performance, energy, reliability, security, optimization, analysis, debugging, etc. The composition of the PC (40 members) has been balanced with regards to several criteria: affiliation (25 from academia, 3 from Google, 4 from IBM, 4 from Intel, 1 each from AMD, MSR, Oracle, and STMicroelectronics), continents (24 from North America, 8 from Europe, 5 from Asia, 1 each from North Africa, Australia, and South America), and gender (34 men and 6 women).

We followed a well-established routine for the paper review process. Conflicts of interest were handled in a way that preserved the double-blind review principle. The 140 abstracts submitted ahead of time were used to speed up the assignment process, whose goal was to maximize the match between the submissions and the PC expertise and interests. Each paper received a minimum of four reviews (many received five). Whenever review confidence about a paper was not high enough, external reviewers were carefully chosen to provide insightful feedback. A rebuttal period was provided to allow authors to respond to the reviews, which was quite helpful in many cases.

The PC meeting was held on November 7, 2015 at MIT in Boston. The objective of the meeting was to obtain a consensus by all the PC members on which papers were best to accept. Papers with wide disagreements were discussed via emails intensively among reviewers and PC members before the actual PC meeting. A few papers with clear consensus for acceptance or rejection were not discussed during the meeting. During the PC meeting, each paper was first summarized by a pre-assigned champion. Each reviewer then summarized the pros and cons, and discussion followed among all PC members to come up with a consensus. Voting was used only for a few papers to make the final decision after long discussions that could not reach a consensus. Papers submitted by PC members were required to be of higher quality than others to get accepted. Among the 12 papers submitted by PC members, 4 were selected.

For effectiveness and fairness, all PC members were required to attend the day-long meeting in person. We are extremely thankful to all PC members for their traveling effort, in particular to our international committee members who traveled from far away, e.g. Europe, Asia, or even Australia.

The entire PC is extremely happy with the quality of the papers. The CGO 2016 program covers a range of topics from programming models to code optimization through static analysis and profiling, performance, energy, correctness, and security. The authors of the submitted papers come from 25 countries dominated by the United State, followed by Germany, India, France, Canada, Brazil, UK, China, etc. We hope you enjoy the papers and we look forward to your contributions to CGO in the future!

Finally, we would like to thank the General Chair Bjoern Franke, the Artifact Evaluation Chairs Grigori Fursin and Bruce Childers, the Web Chair Tom Spink, the Publication Chair Chris Fensch and the Poster Chair Florian Brandner for their many hours of help.

**Fabrice Rastello**, Inria
**Youfeng Wu**, Intel Corp.

# CGO 2016 Organization

## Organizing Committee

| | |
|---|---|
| **General Chair** | |
| Björn Franke | University of Edinburgh, UK |
| **Program Co-Chairs** | |
| Fabrice Rastello | Inria, France |
| Youfeng Wu | Intel, USA |
| **Finance Chair** | |
| Christophe Dubach | University of Edinburgh, UK |
| **Workshop and Tutorials Chair** | |
| Jeronimo Castrillon | TU Dresden, Germany |
| **Publication Chair** | |
| Christian Fensch | Heriot-Watt University, UK |
| **Student Travel Chair** | |
| Ronald Mak | San Jose State University, USA |
| **Sponsors Chair** | |
| Tobias Edler von Koch | Qualcomm Innovation Center, USA |
| **Publicity Chair** | |
| Chris Margiolas | University of Edinburgh, UK |
| **Website Chair** | |
| Tom Spink | University of Edinburgh, UK |
| **Artifact Evaluation Chairs** | |
| Grigori Fursin | dividiti, UK |
| Bruce Childers | University of Pittsburgh, USA |
| **Poster Chair** | |
| Florian Brandner | Télécom ParisTech, France |
| **Local Arrangements Chair** | |
| Vladimir Subotic | Barcelona Supercomputing Center, Spain |

## Steering Committee

| | |
|---|---|
| Kim Hazelwood | Facebook, USA |
| Robert Hundt | Google, USA |
| Scott Mahlke | University of Michigan, USA |
| Jason Mars | University of Michigan, USA |
| Kunle Olukotun | Stanford University, USA |
| Vijay Janapa Reddi | University of Texas at Austin, USA |
| Olivier Temam (Chair) | Google, USA |

## Program Committee

| | |
|---|---|
| Erik Altman | IBM, USA |
| Saman Amarasinghe | MIT, USA |
| Edson Borin | University of Campinas, Brazil |
| Florian Brandner | Télécom ParisTech, France |
| Mauricio Breternitz | AMD, USA |
| Derek Bruening | Google, USA |
| Vugranam C. Sreedhar | IBM, USA |
| Wenguang Chen | Tsinghua University, China |
| Mila Dalla Preda | University of Verona, Italy |
| Evelyn Duesterwald | IBM, USA |
| Guang Gao | University of Delaware, USA |
| Antonio Gonzalez | UPC, Spain |
| Christophe Guillon | STmicroelectronics, France |
| Sebastian Hack | University of Saarland, Germany |
| Ben Hardekopf | UCSB, USA |
| Wei-Chung Hsu | National Taiwan University, Taiwan |
| Robert Hundt | Google, USA |
| Chris J. Newburn | Intel, USA |
| Vijay Janapa Reddi | University of Texas, USA |
| Alexandra Jimborean | Uppsala, Sweden |
| Alain Ketterlin | University Louis Pasteur, France |
| Jaejin Lee | Seoul National University, South Korea |
| Mary Lou Soffa | University of Virginia, USA |
| Scott Mahlke | University of Michigan, USA |
| Vineeth Mekkat | Intel, USA |
| John Mellor-Crummey | Rice University, USA |
| Soo-mook Moon | Seoul National University, South Korea |
| Tipp Moseley | Google, USA |
| Dorit Nuzman | Intel, Israel |
| Michael O'Boyle | University of Edinburgh, UK |
| Ramesh Peri | Intel, USA |
| Louis-Noël Pouchet | Ohio State University, USA |
| Aaron Smith | Microsoft, USA |

| | |
|---|---|
| Cheng Wang | Google, USA |
| Chenggang Wu | ICT, China |
| Jingling Xue | University of New South Wales, Australia |
| Qing Yi | University of Colorado, USA |
| Antonia Zhai | University of Minnesota, USA |

## Additional Reviewers

Aharon Abadi
Marti Anglada
Juan Luis Aragon
Jose-Maria Arnau
Rafael Auler
Riyadh Baghdadi
Ali Bakhoda
Raj Barik
Thierry
Omer Boehm
Jeffrey Bosboom
Diego Caballero
Gaurav Chadha
Benoit Claudel
Albert Cohen
Thanh Tuan Dao
Roshan Dathathri
Francois De Ferriere
Enrique de Lucas
Tyler Denniston
Peng Di
Johannes Doerfert
Gurbinder Gill
Clemens Hammacher
Muhammad Amber
Hassaan
Jiayuan He
Anup Holey
Michael Jacobs
Davoud Jamshidi
Yves Janin
Sanath Jayasena
Sudhanshu Jha
Gangwon Jo
Jaehoon Jung
Wookeun Jung
Rashid Kaleem
Hongjune Kim
Junghyun Kim
Vladimir Kiriansky
Fredrik Berg Kjolstad
David Klaftenegger
John Kloosterman
Rakesh Kumar
Michael Kuperstein
Julien Le Guen
Seong-Won
Roland Leißa
Xiaoming Li
Yue Li
Isabella Mastroeni
Richard Membarth
Charith Mendis
Christophe Monat
Antoine Moynault
Amine Naji
Ragavendra Natarajan
Shruti Padmanabha
Sreepathi Pai
Jason Jong Kyu Park
Jungho Park
Karthik Pattabiraman
Nitzan Peleg
Marius Pirvu
Flavia Pisani
Dimitrios Prountzos
Sandro Rigo
Sigurd Schneider
Jaeho Shin
Michel Steuwer
Kevin Streit
Xin Sui
Vijay Sundaresan
Olivier Temam
Marti Torrents
Gaurang Upasani
Alejandro Valero
Cédric Vincent
Wei Wang
Haitao Wei
Dong Hyuk Woo
Byung-Sun Yang
Ding Ye
Jieming Yin
Jiecao Yu
Ayal Zaks
Babak Zamirai
Mingwei Zhang
Yunming Zhang
Hao Zhou
Stéphane Zuckerman

## Artifact Evaluation Committee Members

| | |
|---|---|
| Ennan Zhai | Yale University, USA |
| Xiaowei Zhu | Tsinghua University, China |
| Yuhao Zhu | University of Texas at Austin, USA |

## CGO 2016 Sponsors and Supporters

### Sponsors

acm Association for Computing Machinery

IEEE computer society

### Supporters

Google

IBM Research

intel

Microsoft Research

NSF

ORACLE

acm sigmicro

acm SIGPLAN

# Beyond the Embarrassingly Parallel – New Languages, Compilers, and Runtimes for Big-Data Processing

Madan Musuvathi
Microsoft Research

Large-scale data processing requires large-scale parallelism. Data-processing systems from traditional databases to Hadoop and Spark rely on embarrassingly-parallel relational primitives (e.g. map, reduce, filter, and join) to extract parallelism from input programs. But many important applications, such as machine learning and log processing, iterate over large data sets with true loop-carried dependences across iterations. As such, these applications are not readily parallelizable in current data-processing systems.

In this talk, I will challenge the premise that parallelism requires independent computations. In particular, I will describe a general methodology for extracting parallelism from dependent computations. The basic idea is replace dependences with symbolic unknowns and execute the dependent computations symbolically in parallel. The challenge of parallelization now becomes a, hopefully mechanizable, task of performing the resulting symbolic execution efficiently. This methodology opens up the possibility of designing new languages for data-processing computations, compilers that automatically parallelize such computations, and runtimes that exploit the additional parallelism. I will describe our initial successes with this approach and the research challenges that lie ahead.

## Biography

Madan Musuvathi is a Principal Researcher at Microsoft Research working in the intersection of programming languages and systems, with specific focus on concurrency and parallelism. His interests span program analysis, systems, model checking, verification, and theorem proving. His research has led to several tools that improve the lives of software developers both at Microsoft and at other companies. He received his Ph.D. from Stanford University in 2004.

# 50 Years of Parallel Programming: Ieri, Oggi, Domani*

Keshav Pingali
University of Texas at Austin

Parallel programming started in the mid-60's with the pioneering work of Karp and Miller, David Kuck, Jack Dennis and others, and as a discipline, it is now 50 years old. What have we learned in the past 50 years about parallel programming? What problems have we solved and what problems remain to be solved? What can young researchers learn from the successes and failures of our discipline? This talk is a personal point of view about these and other questions regarding the state of parallel programming.

* The subtitle of the talk is borrowed from the title of a screenplay by Alberto Moravia, and it is Italian for "Yesterday, Today, Tomorrow."

**Biography**

Keshav Pingali is a Professor in the Department of Computer Science at the University of Texas at Austin, and he holds the W.A. "Tex" Moncrief Chair of Computing in the Institute for Computational Engineering and Sciences (ICES) at UT Austin. Pingali is a Fellow of the IEEE, ACM and AAAS. He was the co-Editor-in-chief of the ACM Transactions on Programming Languages and Systems, and currently serves on the editorial boards of the ACM Transactions on Parallel Computing, the International Journal of Parallel Programming and Distributed Computing. He has also served on the NSF CISE Advisory Committee (2009-2012).

# Knights Landing Intel Xeon Phi CPU: Path to Parallelism with General Purpose Programming

Avinash Sodani
Intel

The demand for high performance will continue to skyrocket in the future, fueled by the drive to solve the challenging problems in scientific world and to provide the horsepower needed to support the compute-hungry use cases that continue to emerge in commercial and consumer space, such as machine learning and deep data analytics. Exploiting parallelism will be crucial in achieving the huge performance gain required to solve these problems. This talk will present the new Xeon Phi Processor, called Knights Landing, which is architected to provide massive amounts of parallelism in a manner that is accessible with general purpose programming. The talk will provide insights into 1) the important architecture features of the processor and 2) the software technology to explore them. It will provide the inside story on the various architecture decisions made on Knights Landing – why we architected the processor the way we did, and on a few programming experience – how the general purpose programming model makes it easy to exploit parallelism on Xeon Phi. It will show measured performance numbers from the Knights Landing silicon on a range of workloads. The talk will conclude with showing the historical trends in architecture and what they mean for software as we extend the trends into the future.

**Biography**

Avinash Sodani is a Senior Principal Engineer at Intel Corporation and the chief architect of the Xeon-Phi Processor called Knights Landing. He specializes in the field of High Performance Computing (HPC). Previously, he was one of the architects of the 1st generation Core processor, called Nehalem, which has served as a foundation for today's line of Intel Core processors. Avinash is a recognized expert in computer architecture and has been invited to deliver several keynotes and public talks on topics related to HPC and future of computing. Avinash holds over 20 US Patents and is known for seminal work on the concept of "Dynamic Instruction Reuse". He has a PhD and MS in Computer Science from University of Wisconsin-Madison and a B.Tech (Hon's) in Computer Science from Indian Institute of Technology, Kharagpur in India.

# Poster Session

As in previous years, the 2016 International Symposium on Code Generation and Optimization (CGO) hosts a combined poster session and ACM Student Research Competition (SRC). We received 32 poster abstracts, of which 15 competed in the ACM SRC.

All poster abstracts were reviewed by three members of the selection committee. Each reviewer attributed up to 30 points to a poster and additionally provided a short comment. The posters then were ranked according to their total score. We were finally able to accept 22 posters, of which 10 competed in the ACM SRC.

I would like to thank the general chair Björn Franke and the local organizers, most notably Cristina Garcia, for hosting the poster session and helping with my inquiries. I would also like to thank the program co-chairs Fabrice Rastello and Youfeng Wu for their help and initiative during the preparations of the poster session. Special thanks go to the members of the selection committee for reviewing all the abstracts, on time, in four days only.

Last, but not least, I would like to thank all the authors who submitted a poster abstract. as well as all students who participate in the SRC. Good luck to all students for the competition!

**Florian Brandner**, Télécom ParisTech

## FastSim: Hybrid Analysis Based Fast Simulator with 900 MIPS Speed for CGRA Core-Profiling

Narasinga Rao Miniskar[1] Rahul R Patil[1] Raj Narayana Gadde[1]
Youngchul Rams-Cho[2] Sukjin Kim[2] Shi Hwa Lee[2]

[1] Samsung R&D Institute-Bengaluru, 560036, India, {nr.miniskar, patil.rahul, raj.gadde}@samsung.com
[2] DMC R&D, Samsung Electronics, Suwon-si, Korea, {rams.cho, sukj.kim, hwa.lee}@samsung.com

Coarse Grained Reconfigurable Architecture (CGRA), viz. Samsung Reconfigurable Processors, provides great flexibility of hardware re-configurability through software solution for high-performance computing. Application developers of CGRA require a fast and accurate core profiling tools to optimize hot-spots running on CGRA core. Further, the certification of various standards of multimedia (DivX, Dolby, DTS, etc.) and vision applications require fast profiling tools to evaluate functionality and performance. The state-of-the-art VLIW, RISC and Superscalar processor simulator design approaches cannot be directly applied to CGRA as the configuration lines of CGRA not only have the opcode and operands of instructions for each cycle but also have hardware reconfiguration information such as routing of operands, functional unit outputs, complex predicate logic, etc. The state-of-the-art CGRA core profiling simulators are very slow (2MIPS) as they simulate operations, register files, MUXes, Flip-Flops and predicate network to control prologue and epilogue. Hence, the validation of multimedia and vision standards often takes long (weeks) simulation times. We propose a novel approach for CGRA simulation by combining configuration analysis and architecture analysis to provide fast and cycle-accurate CGRA simulation model for CGRA profiling. Experimental results have shown that our proposed CGRA-cycle accurate profiling simulator gives $\sim$900 MIPS speed.

## Hot-Rodding the Browser Engine: Automatic Configuration of JavaScript Compilers

Chris Fawcett[1] Lars Kotthoff[1] Holger H. Hoos[1]

[1] University of British Columbia, Department of Computer Science, {fawcettc, larsko, hoos}@cs.ubc.ca

Software systems in many application areas offer to the user a multitude of parameters, switches and other customisation hooks. Humans tend to have difficulties determining the best configurations of these parameters for particular applications. Modern optimising compilers are an example of such software systems; they have hundreds of configurable parameters which need to be tuned for optimal performance, but which interact in unpredictable ways and are often left at their default values.

In this work, we automatically determine compiler parameter settings that result in optimised performance for particular applications. Specifically, we apply a state-of-the-art automated parameter configuration procedure based on sequential, model-based, gradient-free optimisation techniques to two prominent JavaScript compilers and demonstrate that runtime performance improvements, more than 35% in some cases, can be achieved over the default parameter settings on a diverse set of benchmarks. We investigate the performance of this approach applied to entire benchmark suites, as well as to individual benchmarks, showing that it is possible to obtain significant performance gains as the considered tasks get more specific. We also present an analysis of which compiler optimisation parameters (or parameter categories) are most responsible for the increased performance in each of our considered cases.

## A-MapCG: An Adaptive MapReduce Framework for GPUs

Lifeng Liu[1] Meilin Liu[1] Jun Wang[2]
Chongjun Wang[3]

[1] Wright State University, {liu.85, meilin.liu} @wright.edu
[2] University of Central Florida, Jun.Wang@UCF.edu
[3] Nanjing University, chjwang@nju.edu.cn

The MapReduce framework is proposed by Google to process large data sets, which is an efficient framework used in many areas, such as social network, scientific research, electronic business, etc. Hence, many MapReduce frameworks implemented on different platforms are proposed and developed, including Phoenix (based on multicore CPUs), MapCG (based on GPUs), and StreamMR (based on GPUs). However, these MapReduce frameworks have limitations, and they cannot handle the collision problem in the map phase, and the unbalanced workload problem in the reduce phase.To improve the performance of the MapReduce framework on GPGPUs, an Adaptive MapReduce Framework for GPUs (A-MapCG) is proposed and developed based on the MapCG framework, to reduce the number of collisions while inserting key-value pairs in the map phase with the proposed segmentation table, and to handle the unbalanced workload problems using dynamic parallelism in the reduce phase. The proposed A-MapCG framework is evaluated on the Tesla K40 GPU hosted by Intel Core i7-4790 with four benchmarks: Inverse Index (II), Word Count (WC), Page Rank (PR), and Monte Carlo (MC). The experimental results show that our proposed A-MapCG improves the performance by 15x on average compared with MapCG.

## From R to Heterogeneous Accelerators: the MACH project

Vincent Ducrot[1] Thierry Goubier[2] Kevin Juilly[1]
Benoit Da Mota[1,3] Gaetan Bayle Des Courchamps[1] Sébastien Monot[1]

[1] AS+, Eolen Group, {vincent.ducrot, kevin.juilly, g.bayledescourchamps, sebastien.monot}@eolen.com
[2] CEA List, Thierry.Goubier@cea.fr
[3] LERIA, University of Angers, France, benoit.damota@univ-angers.fr

The MACH project aims at making high level, domain oriented programming on heterogeneous accelerators simple through domain specific languages (DSLs) and domain specific embedded languages (DSeLs). One of the explored DSL is the R programming language, and our approach, as a subteam within the MACH project, is to combine the StarPU heterogeneous dataflow runtime, the LLVM compiler infrastructure, a dedicated R front-end and dedicated parallelizer and specializer passes implemented within the LLVM framework. In this poster, we will present the different points of focus of our project; type analysis and dataflow task extraction from R code in the front-end, task parallelization and hardware target independence in the specializer passes, and the articulation between those two main blocks.

## KGEN: a divide-and-conquer approach for Fortran application modernization

Youngsung Kim[1] John Dennis[1] Christopher Kerr[2]
Raghu Raj Prasanna Kumar[1] Amogh Simha[3] Allison Baker[1]
Sheri Mickelson[1]

[1] National Center for Atmospheric Research, Boulder, Colorado, U.S.A., {youngsun, dennis, raghuraj, abaker, mickelso}@ucar.edu
[2] chris.kerr.llc@gmail.com
[3] University of Colorado, Boulder, Colorado, U.S.A., amogh.simha@colorado.edu

Computational kernels, small piece of software that selectively captures the characteristics of larger applications, have been used successfully for decades. Kernels allow for the testing of a compiler's ability to optimize code, performance of future hardware and reproducing compiler bugs. Unfortunately they can be rather time consuming to create and do not always accurately represent the full complexity of large scientific applications. Furthermore, expert knowledge is often required to create such kernels. In this paper, we present a Python-based tool that greatly simplifies the generation of computational kernels from Fortran based applications. Our tool automatically extracts partial source code of a larger Fortran application into a stand-alone executable kernel. Additionally, our tool also generates state data necessary for proper execution and verification of the extracted kernel. We have utilized our tool to extract more than thirty computational kernels from a million-line climate simulation model. Our extracted kernels have been used for a variety of purposes including: code modernization, identification of limitations in compiler optimizations, numerical algorithm debugging, compiler bug reporting, and for procurement benchmarking.

## Atomicity Violation Checker for Task Parallel Programs

Adarsh Yoga[1] Santosh Nagarakatte[1]

[1] Rutgers University, USA, {adarsh.yoga, santosh.nagarakatte}@cs.rutgers.edu

Task based programming models (e.g., Cilk,TBB) simplify multicore programming in contrast to programming with threads. In a task based model, the programmer specifies parallel tasks and the runtime maps these tasks to hardware threads. The runtime automatically balances the load using work-stealing and provides performance portability. However, interference between parallel tasks can result in concurrency errors.

This paper proposes a dynamic analysis technique to detect atomicity violations in task parallel programs, which could occur in different schedules for a given input without performing interleaving exploration. Our technique leverages the series-parallel dynamic execution structure of a task parallel program to identify parallel accesses. It also maintains access history metadata with each shared memory location to identify parallel accesses that can cause atomicity violations in different schedules. To streamline metadata management, the access history metadata is split into global metadata that is shared by all tasks and local metadata that is specific to each task. The global metadata tracks a fixed number of access histories for each shared memory location. Our prototype tool for Intel Threading Building Blocks detects atomicity violations that can potentially occur in different interleavings for a given input with performance overheads similar to Velodrome atomicity checker.

## Hybrid-JIT : Hardware Accelerated JIT Compilation for Embedded VLIW Processors

Simon Rokicki[1] Erven Rohou[2] Steven Derrien[1]

[1] Université de Rennes 1/IRISA, {simon.rokicki, steven.derrien}@irisa.fr
[2] INRIA/IRISA, erven.rohou@inria.fr

Just-in-time (JIT) compilation is widely used in current embedded systems (mainly because of Java Virtual Machine). When targeting Very Long Instruction Word (VLIW) processors, JIT compilation back-ends grow more complex because of the instruction scheduling and register allocation phases. This tends to reduce the benefits of JIT compilation for such systems.

In the context of heterogeneous many-core systems based on VLIW processors, each VLIW core has a different configuration and are not binary compatible. In such a system, JIT compilation can be used to migrate tasks from one core to another. The cost of JIT compilation becomes a major constraint of the system.

In this document, we present the Hybrid-JIT platform, a hardware software co-design where JIT management is handled in software and the back-end is performed by specialized hardware. The hardware accelerator is called by the processor as any other instruction, and perform instruction scheduling following the list scheduling algorithm. This hardware accelerator also performs register allocation. Experimental studies show that this approach leads to a compilation up to 15 times faster and 18 times more energy efficient than a pure software compilation.

## PACXX: Programming Model and Code Generation for Accelerators using C++14

Michael Haidl[1] Sergei Gorlatch[1]

[1] University of Muenster, {m.haidl, gorlatch}@uni-muenster.de

We present PACXX – a unified programming model for programming many-core systems that comprise accelerators like Graphics Processing Units (GPUs). In PACXX, programs are written using exclusively the newest C++14 standard, with all modern features including type inference (auto), variadic templates, generic lambda expressions, as well as STL containers and algorithms. Code for accelerators is generated directly from C++14 code and optimized during program execution to achieve high performance. We aim at overcoming the current use of two distinct programming models: the host code for the CPU is written in C/C++ with the restricted, C-like API for memory management, while the device code for the GPU has to be written using a device-dependent, explicitly parallel programming model, e.g., OpenCL or CUDA, which leads to long, poorly structured and error-prone codes. We implement PACXX by a custom compiler (based on the Clang front-end and LLVM IR) and a runtime system that together perform major tasks of memory management and data synchronization automatically and transparently for the programmer. We show that PACXX codes are about 60% shorter than their OpenCL and CUDA equivalents and outperform them by up to 10%.

## Significance Based Computing for Reliability and Power Optimization

Nikolaos Bellas[1] Christos D.Antonopoulos[1] Spyros Lalis[1]
Konstantinos Parasyris[1] Vassilis Vassiliadis[1] Uwe Naumann[2]
Jan Riehme[2] Jens Deussen[2] Dimitrios S. Nikolopoulos[3]
Georgios Karakonstantis[3] Hans Vandierendonck[3] Charalampos Chalios[3]

[1] CERTH & University of Thessaly, Greece, {nbellas, cda, lalis,koparasy, vasiliad}@uth.gr
[2] RWTH Aachen University, Germany, {naumann, riehme, deussen}@stce.rwth-aachen.de
[3] Queen's University Belfast, United Kingdom, {d.nikolopoulos, g.karakonstantis,h.vandierendonck, cchalios01}@qub.ac.uk

Manufacturing process variability at low geometries and energy dissipation are the biggest challenges in the design of future computing systems. Currently, manufacturers go to great lengths to guarantee fault-free operation of their products by introducing redundancy in voltage margins, conservative layout rules, and extra protection circuitry.

However, such design redundancy leading to significant energy overheads may not be really required, since many modern workloads, such as multimedia, machine learning, visualization, etc. can tolerate a degree of imprecision in computations and data.

SCoRPiO seeks to exploit this observation and to relax reliability requirements for the hardware layer by allowing a controlled degree of unreliability to be introduced to computations and data. Focus is on researching methods that allow the system- and application-software layers to synergistically characterize the significance of various parts of the program for output quality, and their tolerance to faults. In turn, this will enable the system to aggressively reduce its power footprint by opportunistically powering hardware modules below nominal values.

Reliability issues in SCoRPiO are not seen as a problem but rather as an opportunity to rethink the concept of computation in a novel and vertical way that will allow semiconductor industry to progress to smaller technology geometries.

## Inference of Peak Density of Indirect Branches to Detect ROP Attacks

Mateus Tymburiba[1] Rubens E. A. Moreira[1] Fernando Magno Quintao Pereira[1]

[1] Department of Computer Science, UFMG, Brazil, {mateustymbu,rubens,fernando}@dcc.ufmg.br

A program subject to a Return-Oriented Programming (ROP) attack usually presents an execution trace with a high frequency of indirect branches. From this observation, several researchers have proposed to monitor the density of these instructions to detect ROP attacks. These techniques use universal thresholds: the density of indirect branches that characterizes an attack is the same for every application. This paper shows that universal thresholds are easy to circumvent. As an alternative, we introduce an inter-procedural semi-context-sensitive static code analysis that estimates the maximum density of indirect branches possible for a program. This analysis determines detection thresholds for each application; thus, making it more difficult for attackers to compromise programs via ROP. We have used an implementation of our technique in LLVM to find specific thresholds for the programs in SPEC CPU2006. By comparing these thresholds against actual execution traces of corresponding programs, we demonstrate the accuracy of our approach. Furthermore, our algorithm is practical: it finds an approximate solution to a theoretically undecidable problem, and handles programs with up to 700 thousand assembly instructions in 25 minutes.

## Symbolic Range Analysis of Pointers

Vitor Paisante[1] Maroua Maalej[2] Leonardo Barbosa[1]
Laure Gonnord[3] Fernando Magno Quintao Pereira[1]

[1] Department of Computer Science UFMG, Brazil, {paisante, leob, fernando}@dcc.ufmg.br
[2] University of Lyon, LIP, Lyon, France Maroua.Maalej@ens-lyon.fr
[3] Univ. Lyon 1, LIP, Lyon, France, Laure.Gonnord@ens-lyon.fr

Alias analysis is one of the most fundamental techniques that compilers use to optimize languages with pointers. However, in spite of all the attention that this topic has received, the current state-of-the-art approaches inside compilers still face challenges regarding precision and speed. In particular, pointer arithmetic, a key feature in C and C++, is yet to be handled satisfactorily. This paper presents a new alias analysis algorithm to solve this problem. The key insight of our approach is to combine alias analysis with symbolic range analysis. This combination lets us disambiguate fields within arrays and structs, effectively achieving more precision than traditional algorithms. To validate our technique, we have implemented it on top of the LLVM compiler. Tests on a vast suite of benchmarks show that we can disambiguate several kinds of C idioms that current state-of-the-art analyses cannot deal with. In particular, we can disambiguate 1.35x more queries than the alias analysis currently available in LLVM. Furthermore, our analysis is very fast: we can go over one million assembly instructions in 10 seconds.

## Android Performance Optimization from Intelligence of Dynamic Compiler Filters

Bhuwan Bhaskar Mishra[1] Anuradha Kanukotla[1] Atikant Singh[1]

[1] System Software and SoC, Samsung Research and Development, Institute Bangalore, India, {bhuwan.m, k.anuradha and Atikant.s}@samsung.com

The runtime environment of Android Lollipop is based on an Ahead of Time (AOT) compiler, which translates the bytecode of an application to native machine code during installation. Despite the performance improvements due to AOT, installation time increases and applications requires extra memory to store its native code. Google defines six compiler configurations such as everything, speed, balanced, space, interpret-only and verify-none which determine how to pre-optimize/compile an application, thereby prioritizing different aspects such as installation time, runtime performance and storage space. In this research paper, we propose to use a combination of different compiler filters to reduce installation time and memory usage (both RAM and ROM). We experimented with ten different combinations of filters provided by Android and found out that combination of speed and interpret-only filter performs better than other combinations. As per the result of our research, we found that RAM and ROM usage is reduced by more than 20% and installation time is reduced by 50% in comparison to the default filter.

## Translating Embedded VM Code in x86 Binary Executables

Haijiang Xie[1] Juanru Li[1] Yuanyuan Zhang[1]

[1] Shanghai Jiao Tong University, hjxie04@gmail.com, {jarod, yyjess}@sjtu.edu.cn

Code protection schemes nowadays adopt language embedding, a technique in which a customized language is built to obfuscate original code through transforming it into a customized form with which the analyst is not familiar. The transformed code is then interpreted by a so-called Embedded VM. This type of transformation does increase the cost of code comprehending and maintaining, and introduces extra runtime overhead.

In this paper, we conduct an in-depth study on embedded VM based code protection and propose a deobfuscation approach that aims to recover the original code form. Our approach first pinpoints the interpretation procedure and partitions handlers of the embedded VM, and then employs a VM-state based handler translating, which represents the behaviors of handlers in terms of VM-state operations. Finally, the translated operations of each handler is optimized and transformed into host code. After this process we can obtain a clear and runtime efficient code representation.

We build Nightingale, a binary translation tool, to fulfil this de-obfuscation automatically with x86 binary executables. We test our approach with three latest commercial code obfuscators and a set of home brewed obfuscation schemes. The results demonstrate that this kind of obfuscated code can be simplified with host language effectively.

## Resource Sharing for GPUs

Vishwesh Jatala[1] Jayvant Anantpur[2] Amey Karkare[1]

[1] Department of CSE, IIT Kanpur, {vjatala, karkare}@cse.iitk.ac.in
[2] SERC, IISc, Bangalore, jayvant@hpc.serc.iisc.ernet.in

Graphics Processing Units (GPUs) achieve high throughput by utilizing thread level parallelism (TLP). The amount of TLP present in a GPU depends on the number of resident threads in its streaming multiprocessors (SM). The number of threads and thread blocks that can be launched in an SM for a kernel depends on the kernel's requirement of resources, such as registers and scratchpad memory. Since the resources of an SM are allocated at thread block level granularity, some resources remain under utilized.

In this poster, we present a resource sharing approach that improves the resource utilization by launching additional thread blocks. These additional thread blocks share some of their resources with other existing thread blocks in the SM. Further, we propose three optimizations to effectively use the additional thread blocks. We show the effectiveness of our approach by implementing it for two resources: Registers and Scratchpad Memory. We validated our approach experimentally on several kernels from CUDA-SDK, GPGPU-Sim, Rodinia, and Parboil benchmarks. We observed that kernels that under utilize registers show an average improvement of 11% with register sharing, whereas, kernels that under utilize scratchpad memory show an average improvement of 12.5% with scratchpad sharing.

## Autotuning Multi-Tiered Applications for Performance

Vimuth Fernando[1] Sanath Jayasena[1]

[1] University of Moratuwa, {vimuth.10, sanath@cse.mrt.ac.lk}@cse.mrt.ac.lk

With the current popularity of cluster based web applications, getting the maximum possible performance from applications deployed in multi-tiered environments becomes an important task. But the large numbers of configurable parameters in today's server applications make manually tuning them for better performance a difficult task. In this work, we explore the autotuning approach, which is generally used to tune the performance of programs in traditional HPC settings, to tune multi-tiered web applications. Our approach is based on OpenTuner, a framework that can be used to build auto-tuners to search through a configuration space for an optimal configuration. However, the wide variations and the dynamic nature in the runtime environment, such as network congestion, variations in demand, possible node failures and changes in workloads pose a significant challenge for this approach.

In this work, we explore offline tuning techniques to overcome the challenges of autotuning multi-tiered applications. We present preliminary results of offline autotuning experiments that tuned benchmark applications for multiple performance goals. We show that 20% to 25% improvements in response time and throughput can be achieved through our offline autotuning approach. We also show that different performance goals can lead to differences in configurations and discuss the shortcomings of offline autotuning methods.

## Unison: Constraint-Based Register Allocation and Instruction Scheduling

Roberto Castañeda Lozano[1,2] Gabriel Hjort Blindell[2] Christian Schulte[1,2]
Mats Carlsson[1]

[1] SICS, Sweden, {rcas, matsc}@sics.se
[2] KTH, Sweden, {ghb, cschulte}@kth.se

Unison is a flexible, potentially optimal approach to integrated register allocation and instruction scheduling. Using modern combinatorial optimization allows Unison to handle, for the first time, all subproblems of global register allocation in integration with instruction scheduling while scaling to functions with hundreds of instructions. Unison can be used as an alternative or a complement to traditional approaches, which by solving each of the problems in isolation with heuristic algorithms trade off flexibility and code quality for compilation speed.

Our approach is based on a program representation (Unison IR) and an integrated combinatorial model that expresses the potential decisions taken by register allocation and instruction scheduling and the constraints imposed by the processor and the semantics of a low-level function representation. The combinatorial model is solved with constraint programming, an optimization technique that is particularly effective for allocation and scheduling problems.

Experiments with MediaBench for Hexagon (a multimedia DSP) show that Unison is both practical (it scales to medium- size functions) and effective (it yields 7% faster code than LLVM's heuristic algorithms on average). The approach is most suitable where higher compilation times can be tolerated or heuristic algorithms just cannot deal with the complexity of the targeted processors.

**Acknowledgments**
This research is partially funded by LM Ericsson AB and the Swedish Research Council (VR 621-2011-6229). The authors are grateful for helpful comments from the reviewers.

## A Predictive Modeling Framework For Compiler Phase-ordering Problem

Amir Hossein Ashouri[1] Andrea Bignoli[1] Gianluca Palermo[1]
Cristina Silvano[1]

[1] Politecnico Di Milano, ITALY, {name.family-name}@polimi.it

Today's compilers offer a huge number of transformation optimizations to choose among and this choice can significantly impact on the performance of the code being optimized. Not only the selection of compiler optimizations represent a hard problem to be solved [2, 3], but also finding the best ordering can add further complexity, making it a long standing problem in the compilation research [1, 5]. Classic predictive modelings simply can not cope with the enormous complexity of the optimizations within a sequence. This paper proposes a novel autotuning framework i) to dynamically explore and characterize the applications and ii) to predict the best compiler optimizations to be applied in order to maximize the performance. The framework also presents a mapping technique capable of transforming any representation of compiler optimizations vector in the phase-ordering space including those having repetitions into a new binary representation that is classified under the problem of selection of compiler optimization. This way the compiler researchers can benefit from exploiting fix-feature vectors for the predictive modelings. Experimental results using the latest LLVM compiler framework and cBench [4] suite have shown effectiveness of the mapping technique by utilizing a number of predictive modelings. We show statistical analysis over the distribution of the data and the quality comparison with respect to Random Iterative Compilation model and standard optimization levels -O2 and -O3.

**Acknowledgments**
This work is partially funded by the European Union's Horizon 2020 research and innovation programme under grant agreement ANTAREX-671623.

## Numerically Accurate Code Generation

Nasrine Damouche[1] Matthieu Martel[1] Alexandre Chapoutot[2]

[1] University of Perpignan, LAboratory of Mathematics and PhysicS, 52 Avenue Paul Alduy, 66860 PERPIGNAN Cedex 9, {nasrine.damouche, matthieu.martel}@univ-perp.fr
[2] U2IS, ENSTA ParisTech, Paris-Saclay University, 828 bd des Maréchaux, 91762 Palaiseau cedex, France, alexandre.chapoutot@ensta-paristech.fr

Floating-point computations are widely used in various areas including embedded and critical systems. However, the results of these computations are necessarily perturbed by small or large round-off errors. In critical scenarii, these errors may be accumulated and may generate serious damages to humans, industry, finance, et cetera. To deal with this matter, we correct these errors by transforming automatically programs in a source to source manner. Our transformation relies on a static analysis by abstract interpretation which provides variables ranges used to guide our tool. Currently, we transform not only arithmetic expressions, but also pieces of code with assignments, conditionals and loops and ongoing work is carried out to handle functions and arrays. We have shown that we significantly improve the numerical accuracy of computations by rewriting programs while trying to find the best candidate code in terms of accuracy and minimization of the error relatively to the exact model. We have also succeeded to reduce by about 20% the number of iterations needed for the convergence of numerical iterative methods for matrix inversion, orthogonalization or eigenvalues computations by improving their accuracy. In addition, we have proven that the generated programs do not necessarily have the same semantics as the original program but are mathematically equivalent.

## Generalized Tiling for Loops and Dataflow Programs

Duco van Amstel[1]

[1] Kalray SA / Inria, duco.van-amstel@{kalray.eu,inria.fr}

Loop tiling is a much studied code optimization. A limit to classical tiling is the fact that non-nested loops can not be tiled. Our approach generalizes tiling by considering the instructions of a single iterations as an extra dimension that can be tiled. A consequence is that data reuses can no longer be expressed in the same way for each point of the tiled domain. We transform the graph of data reuses into a canonical form on which we can compute the best possible tile sizes in order to optimize data reuse for a given target memory size. These computations involve a new model for both the memory footprint and the memory communications induced by a given code. The computed parameters can then used to transform the loops through loop fission, unrolling and instruction rescheduling. On a broader scope the canonical form that is used for the data reuse graph has very strong similarities with the graph of actors of a dataflow language as actors can be assimilated with the instructions of a loop. As such our generalized tiling approach can also be used for both scheduling and ressource allocation in a distributed dataflow runtime.

## MAFE: An Environment for MATLAB-to-C Compilation Supporting Static and Dynamic Memory Allocation and Multi-Level User Interactive Code Optimization

Christakis Lezos[1] Grigoris Dimitroulakos[1] Ioannis Latifis[1]
Konstantinos Masselos[1]

[1] University of Peloponnese, Department of Informatics and Telecommunications, Terma Karaiskaki, 22100 Tripoli, Greece, {lezos,dhmhgre,latifis,kmas}@uop.gr

MATLAB compilation to lower/implementation level languages is performed for application development (e.g. embedded C generation, high-level synthesis to VHDL) and for performance optimization. In this work MAFE, an environment for MATLAB-to-C compilation is proposed. The C code generated by MAFE allocates memory for arrays both statically and dynamically. MAFE's approach for dynamic memory allocation preallocates arrays and a maximum and an imaginary size are assigned to them. The imaginary size changes when a new value is assigned to the array. This way the array size can change dynamically without any reallocation cost. Furthermore, MAFE can optionally generate exception functions that implement runtime checks on the arrays' sizes which is an advantage over Mathworks' MATLAB Coder that infers all array sizes at compile time and does not generate code for execution time size checks. MAFE environment also includes a source code optimizer applying loop and data reuse exploitation transformations. The optimizer supports developers in efficiently applying transformations interactively both at low level (C code) and at high level (MATLAB code). Experimental results prove that the optimizer can: 1) improve execution time of a MATLAB algorithm up to 30% for the C code generated by MAFE and up to 22% for the C code generated by MATLAB Coder, and 2) reduce cache misses up to 31% for the C code generated by MAFE and up to 40% for the C code generated by MATLAB Coder.

# Iterative compilation on mobile devices

Paschalis Mpeis[1] Pavlos Petoumenos[1] Hugh Leather[1]

[1] University of Edinburgh, p.mpeis@ed.ac.uk, {ppetoume, hleather}@inf.ed.ac.uk

The abundance of poorly optimized mobile applications signifies tremendous underutilization of the already limited computing sources of mobile devices.

Iterative compilation can reclaim such waste, however, neither of the existing approaches fit the peculiar mobile context. Offline search is tailored for fixed applications, input, and architectures. Online self-adapting approaches overcome such limitations, however, they degrade a user's experience, as they inevitably evaluate poorly performing transformations during their early-learning stages. Moreover, evaluation is complicated by online execution noise.

We propose a fusion of iterative compilation with a capture and-replay mechanism to overcome these problems. Transparent online operations capture users' input to time-consuming functions of applications. The input is a minimal set of pages, accessed during the execution of such functions. It is identified by protecting relevant memory areas and subsequently handling deliberate segmentation violations. A fork that precedes memory protection, leverages the efficient copy-on-write mechanism to duplicate the input to an idle child process. On application's termination the child stores such pages on disk. Then, at idle-and-charging periods, different transformations are evaluated through replaying. Custom link strategies ensure that replay succeeds in spite of code alternations.

We have tested our approach on embedded devices' benchmarks, where we achieved at least 20% speedup against -O3 of Clang, after randomly visiting a tiny fraction of the space.

# Approximation at the Level of Functions

Aurangzeb[1] Rudolf Eigenmann[1]

[1] Purdue University, {orangzeb, eigenman}@purdue.edu

Approximate computing has emerged as an active area of research in recent years. A number of techniques have been proposed to increase performance of applications that can tolerate a certain degree of inaccuracy. Approximating functions can offer significant performance improvement but this opportunity has not yet been fully explored by the approximate computing community. We introduce techniques to perform approximation at the level of functions. We present our schemes with two flavors and discuss four realizations. We show that mathematical and scientific functions can be approximated and used in applications amenable to approximate computing by presenting results on 90 such functions from the GNU Scientific Library (GSL). Results show that our approximation scheme was able to speed up 92% of all functions. For 71% of the functions, the normalized RMS error in the approximated result was very small (0.06 on average) with 9.3x speedup, on average. Whereas for another 15% of functions it was 0.49 with an average speedup of 9.5x. We also demonstrate the feasibility and practicality of the approach by presenting results on three real applications. The average speedup for these applications is 1.74x with 0.5% error, on average.

# Contents

## Frontmatter

**Messages from the Chairs** . . . . . . . . . . iii

**Organization** . . . . . . . . . . v

**Sponsors and Supporters** . . . . . . . . . . x

**Keynote Abstracts** . . . . . . . . . . xi

**Poster Abstracts** . . . . . . . . . . xiv

## Profiling Feedback

**Cheetah: Detecting False Sharing Efficiently and Effectively**
Tongping Liu and Xu Liu — *University of Texas at San Antonio, USA; College of William and Mary, USA* . . . . . . 1

**AutoFDO: Automatic Feedback-Directed Optimization for Warehouse-Scale Applications**
Dehao Chen, David Xinliang Li, and Tipp Moseley — *Google, USA* . . . . . . 12

**Portable Performance on Asymmetric Multicore Processors**
Ivan Jibaja, Ting Cao, Stephen M. Blackburn, and Kathryn S. McKinley — *University of Texas at Austin, USA; Institute of Computing Technology at Chinese Academy of Sciences, Australia; Australian National University, Australia; Microsoft Research, USA* . . . . . . 24

## Data Layout and Vectorization

**StructSlim: A Lightweight Profiler to Guide Structure Splitting**
Probir Roy and Xu Liu — *College of William and Mary, USA* . . . . . . 36

**Exploiting Recent SIMD Architectural Advances for Irregular Applications**
Linchuan Chen, Peng Jiang, and Gagan Agrawal — *Ohio State University, USA* . . . . . . 47

**Exploiting Mixed SIMD Parallelism by Reducing Data Reorganization Overhead**
Hao Zhou and Jingling Xue — *UNSW, Australia* . . . . . . 59

## GPU

**A Black-Box Approach to Energy-Aware Scheduling on Integrated CPU-GPU Systems**
Rajkishore Barik, Naila Farooqui, Brian T. Lewis, Chunling Hu, and Tatiana Shpeisman — *Intel Labs, USA; Georgia Institute of Technology, USA* . . . . . . 70

**Portable and Transparent Software Managed Scheduling on Accelerators for Fair Resource Sharing**
Christos Margiolas and Michael F. P. O'Boyle — *University of Edinburgh, UK* . . . . . . 82

**Communication-Aware Mapping of Stream Graphs for Multi-GPU Platforms**
Dong Nguyen and Jongeun Lee — *Ulsan National Institute of Science and Technology, South Korea* . . . . . . 94

**gpucc: An Open-Source GPGPU Compiler**
Jingyue Wu, Artem Belevich, Eli Bendersky, Mark Heffernan, Chris Leary, Jacques Pienaar, Bjarke Roune, Rob Springer, Xuetian Weng, and Robert Hundt — *Google, USA* . . . . . . 105

## Affine Programs

**A Basic Linear Algebra Compiler for Structured Matrices**
Daniele G. Spampinato and Markus Püschel — *ETH Zurich, Switzerland* . . . . . . 117

**Opening Polyhedral Compiler's Black Box**
Lénaïc Bagnères, Oleksandr Zinenko, Stéphane Huot, and Cédric Bastoul — *INRIA, France; University of Strasbourg, France* . . . . . . 128

**Trace-Based Affine Reconstruction of Codes**
Gabriel Rodríguez, José M. Andión, Mahmut T. Kandemir, and Juan Touriño — *Universidade da Coruña, Spain; Pennsylvania State University, USA* . . . . . . 139

## Static Analysis

**Inference of Peak Density of Indirect Branches to Detect ROP Attacks**
Mateus Tymburibá, Rubens E. A. Moreira, and Fernando Magno Quintão Pereira — *Federal University of Minas Gerais, Brazil* . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 150

**Sparse Flow-Sensitive Pointer Analysis for Multithreaded Programs**
Yulei Sui, Peng Di, and Jingling Xue — *UNSW, Australia* . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 160

**Symbolic Range Analysis of Pointers**
Vitor Paisante, Maroua Maalej, Leonardo Barbosa, Laure Gonnord, and Fernando Magno Quintão Pereira — *Federal University of Minas Gerais, Brazil; University of Lyon, France* . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 171

## Programming Models

**Towards Automatic Significance Analysis for Approximate Computing**
Vassilis Vassiliadis, Jan Riehme, Jens Deussen, Konstantinos Parasyris, Christos D. Antonopoulos, Nikolaos Bellas, Spyros Lalis, and Uwe Naumann — *CERTH, Greece; RWTH Aachen University, Germany* . . . . . . . . . . . . . 182

**Have Abstraction and Eat Performance, Too: Optimized Heterogeneous Computing with Parallel Patterns**
Kevin J. Brown, HyoukJoong Lee, Tiark Rompf, Arvind K. Sujeeth, Christopher De Sa, Christopher Aberger, and Kunle Olukotun — *Stanford University, USA; Purdue University, USA* . . . . . . . . . . . . . . . . . . . . . . . . . . . 194

**NRG-Loops: Adjusting Power from within Applications**
Melanie Kambadur and Martha A. Kim — *Columbia University, USA* . . . . . . . . . . . . . . . . . . . . . . . . 206

## Correctness

**Validating Optimizations of Concurrent C/C++ Programs**
Soham Chakraborty and Viktor Vafeiadis — *MPI-SWS, Germany* . . . . . . . . . . . . . . . . . . . . . . . . . . . . 216

**IPAS: Intelligent Protection against Silent Output Corruption in Scientific Applications**
Ignacio Laguna, Martin Schulz, David F. Richards, Jon Calhoun, and Luke Olson — *Lawrence Livermore National Laboratory, USA; University of Illinois at Urbana-Champaign, USA* . . . . . . . . . . . . . . . . . . . . . . . . . . . 227

**Atomicity Violation Checker for Task Parallel Programs**
Adarsh Yoga and Santosh Nagarakatte — *Rutgers University, USA* . . . . . . . . . . . . . . . . . . . . . . . . . . . 239

## Binary/Virtualization

**Flexible On-Stack Replacement in LLVM**
Daniele Cono D'Elia and Camil Demetrescu — *Sapienza University of Rome, Italy* . . . . . . . . . . . . . . . . . . 250

**BlackBox: Lightweight Security Monitoring for COTS Binaries**
Byron Hawkins, Brian Demsky, and Michael B. Taylor — *University of California at Irvine, USA; University of California at San Diego, USA* . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 261

**Re-constructing High-Level Information for Language-Specific Binary Re-optimization**
Toshihiko Koju, Reid Copeland, Motohiro Kawahito, and Moriyoshi Ohara — *IBM Research, Japan; IBM, Canada* . 273

**Author Index** . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 284