Message from the General Chairs

Welcome to the 16th International Symposium on Code Generation and Optimization (CGO), held at the Austria Trend Eventhotel Pyramide, located just to the south of Vienna, Austria, February 24-28, 2018.

CGO provides a premier venue to bring 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 conference spans the spectrum from purely static to fully dynamic approaches, and from pure software-based methods to specific architectural features and support for code generation and optimization.

CGO’s strong reputation is the effort of many people, including the Program Committee and the numerous authors and speakers that contribute to the technical program. We thank this year’s Program Chairs (Teresa Johnson and Michael Boyle) for helping put together an excellent technical program involving 30 high-quality research articles. We also thank the Workshop and Tutorials Chair (Stephen Siegel) for putting together a highly selective set of workshops and tutorials, including domain specific languages, reversible computing, and the LLVM compiler infrastructure.

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 the Artifact Evaluation Committee, chaired by Michel Steuwer and Grigori Fursin. The task of this committee is to assess how the artifacts support the work described in the papers. We continue to highlight papers, which have successfully passed the artifact evaluation, and this year we also mark these papers in the proceedings with the respective stamps on the front page. This will also allow to search for papers based on an existing artifact evaluation in the ACM digital library in future.

CGO’s technical program is held in conjunction with the 24th International Symposium on High-Performance Computer Architecture (HPCA) and the 23rd International Symposium on Principles and Practice of Parallel Programming (PPoPP), and the 27th International Conference on Compiler Construction (CC). HPCA
provides a high-quality forum for scientists and engineers to present their latest research findings in this rapidly-changing field. PPoPP is the forum for leading work on all aspects of parallel programming, including theoretical foundations, techniques, languages, compilers, runtime systems, tools, and practical experience. CC is interested in work on processing programs in the most general sense: analyzing, transforming or executing input that describes how a system operates, including traditional compiler construction as a special case.

The three conferences CGO, HPCA, and PPoPP are held together, side-by-side, on the same main conference days, CC during the workshop days. The co-location is intended to foster collaboration between the communities by bringing together software and hardware architects, programming language designers, and compiler writers.

The three keynote speeches, and this year also a panel on women in academia and industry, are joint between the three conferences CGO, HPCA, and PPoPP. The side-by-side technical program talks, the co-located Conference on Compiler Construction and the workshops and tutorials, are all open to attendees from any of the conferences.

As in previous years, CGO also hosts a Student Research Competition (SRC) session, offering a unique forum for undergraduate and graduate students to present their original research before a panel of judges and attendees at CGO.

An integrated co-location would not be possible without the effort of CGO’s organizing committee, which has coordinated several logistical details carefully with HPCA and PPoPP. We are extremely grateful for the effort of all members involved. In addition to the Program Chairs, Program Committee, and Organizing Committee, we are grateful to the Austrian Computer Society (OCG) who have helped with the registration and hotel planning, the IEEE/ACM staff that have helped us with the budget, planning, and finances, and Conference Publishing Consulting (CPC) who have helped us with preparing the proceedings.

One of CGO’s long-standing goals is to support student development by providing them with easy access to the conference, heavily subsidizing their travel and registration costs. This is made possible by the generous donations from our industry supporters and our sponsors, i.e., NSF, SIGPLAN and SIGMICRO communities.

Jens Knoop, TU Vienna
Markus Schordan, Lawrence Livermore National Laboratory
CGO 2018 General Chairs
Message from the Program Chairs

We would like to welcome you to CGO 2018 in the beautiful city of Vienna. On behalf of the Program Committee, we are delighted to present the program for the 2018 International Symposium on Code Generation and Optimization. We hope that you find it informative and stimulating.

We selected 30 high quality papers from 105 submissions giving an acceptance rate of 29\%. We had a total of 37 PC members, with 23 from academia and 14 from the industry. The committee consisted of 12 women and 25 men, 3 of whom were from Asia, 8 from Europe, 1 from the Middle East, with the remainder from North America.

Each paper received at least 4 reviews during the double-blind process. In those cases where the PC was less confident in its expertise, the papers received additional reviews from our external reviewer committee.

The program committee meeting was held in New York on October 27th, and was kindly hosted by Google. We would like to thank all PC members for their considerable commitment and effort in selecting the CGO’18 program.

The final program includes papers that focus on traditional and fundamental areas of compiler research, as well as emerging topics such as compiling for heterogeneous architectures, mobile systems, security, and program synthesis.

Finally, we would also like to thank the submission chair Chris Cummins for managing the submission site and facilitating a smooth and seamless PC meeting.

Teresa Johnson, Google
Michael O’Boyle, University of Edinburgh
CGO 2018 Program Chairs
Organization Committee

**General Chairs**  
Jens Knoop, Vienna University of Technology, Austria  
Markus Schordan, Lawrence Livermore National Laboratory

**Program Chairs**  
Teresa Johnson, Google  
Michael O’Boyle, University of Edinburgh

**Workshop and Tutorials Chair**  
Stephen Siegel, University of Delaware

**Artifact Evaluation Chairs**  
Michel Steuwer, University of Glasgow  
Grigori Fursin, Dividiti /c Tuning Foundation

**Student Research Competition Chair**  
Simone Campanoni, Northwestern University

**Students/Travel Chair**  
Ronald Mak, San Jose State University

**Publicity Chair**  
Franz Franchetti, Carnegie Mellon University

**Sponsorship Chair**  
Aaron Smith, Microsoft

**Proceedings Chair**  
Alexandra Jimborean, Uppsala University

**Treasurer/Finance Chair**  
Jack W. Davidson, University of Virginia

**Web Chair**  
Chris Cummins University of Edinburgh
Steering Committee

Carol Eidt, Microsoft
Fabrice Rastello, Inria
Jack W. Davidson, University of Virginia
Jason Mars, University of Michigan
Kunle Olukotun, Stanford University
Scott Mahlke, University of Michigan
Vijay Janapa Reddi, University of Texas at Austin

Program Committee

Albert Cohen, Inria
Antonia Zhai, University of Minnesota
Apan Qasem, Texas State / AMD Research
Ayal Zaks, Intel, Haifa
Carol Eidt, Microsoft
Chenggang Wu, Institute of Computing Technology, Chinese Academy of Sciences
Cristina Silvano, Politecnico di Milano
David August, Princeton University
Eva Darulova, MPI-SWS
Evelyn Duesterwald, IBM
Fabrice Rastello, Inria
Guilherme Ottoni, Facebook
Hugh Leather, University of Edinburgh
Hyesoon Kim, Georgia Tech
Irene Finocchi Sapienza, University of Rome
Jaejin Lee, Seoul National University
Jeff Huang, Texas A&M
Jingyue Wu, Waymo
Jörg Henkel, Karlsruhe Institute of Technology
Keshav Pingali, University of Texas, Austin
Lingjia Tang, University of Michigan
Mahmut T. Kandemir, Pennsylvania State University
Maria Garzaran, Intel, University of Illinois at Urbana-Champaign
Michael Carbin, MIT
Milind Chabbi, Baidu Research, USA
Norm Rubin, Nvidia
Robert Hundt, Google
Saman Amarasinghe, MIT
Sasa Misailovic, University of Illinois at Urbana-Champaign
Taewook Oh, Facebook
Tatiana Shpeisman, Google
Timothy M. Jones, University of Cambridge
Tipp Moseley, Google
Uma Srinivasan, Twitter
Wei-Chung Hsu, NTU (National Taiwan University)
Xu Liu, College of William and Mary
Zehra Sura, IBM Research

External Review Committee

Ali Javadi-Abhari, IBM
Kristof Beyls, AMD
Michel Steuwer, University of Glasgow
Pramod Bhatotia, University of Edinburgh
Ravi Teja Mullanpudi, CMU
Santosh Nagarakatte, Rutgers University
Vassil Vassilev, CERN
Venkatesh Srinivasan, Google
William Thies, Microsoft Research
Zheng Wang, Lancaster University

Additional Reviewers

Abdulrahman Mahmoud
Adam Singer
Ahmet Erdem
Blake Hechtman
Charith Mendis
David Majnemer
Doe Hyun Yoon
Fredrik Kjolstad
Giovanni Agosta
Gurtej Kanwar
Hansen Zhang
HyoukJoong Lee
Jason Jong Kyu Park
Jeremiah Willcock
Jessica Ray
Jordan Fix
Jose Baiocchi Paredes
Karthik Murthy
Nayana Prasad Nagendra
Peng (Tom) Sun
Peter Gavin
Radha Venkatagiri
Raghavan Raman
Riyadh Bağdadi
Ruoyu (Kevin) Zhou
Sam Ainsworth
Shane Delmore
Simeon Franklin
Sotirios Xydis
Sotiris Apostolakis
Stephen Beard
Stephen Chou
Tayo Oguntebi
Vladimir Kiriansky
Yili Zheng
Yunmin Zhang

Artifact Evaluation Committee

Adilla Susungi, MINES ParisTech, PSL Research University
Amir H. Ashouri, University of Toronto
Bastian Hagedorn, Muenster University, Germany
Bozhen Liu, Texas A&M University
Cindy Catherine, Orozco Bohorquez Stanford University
Gangwon Jo, Seoul National University
Grigori Fursin (chair), dividi'ti / c Tuning foundation
Gurbinder Gill, University of Texas
Gyeongmin Lee, POSTECH
Jie Zhao, INRIA and ENS, DI
Jungho Park, Seoul National University
Khemraj Shukla, Oklahoma State University
Michel Steuwer (chair), University of Glasgow
Pengfei Su, College of William and Mary
Qingsen Wang, College of William and Mary
Roshan Dathathri, University of Texas at Austin
Sam Ainsworth, University of Cambridge
Snehasish Kumar, Google
Soham Chakraborty, MPI-SWS
Tuan Tran, University of California, Santa Cruz
Valentin Radu, University of Edinburgh
Wei Zhang, The George Washington University
Authors of accepted papers were given the opportunity to participate in the artifact evaluation process\(^1\) by submitting a research artifact. ACM defines an artifact as "a digital object that was either created by the authors to be used as part of the study or generated by the experiment itself". The artifact evaluation process checks if the submitted artifact supports the claims made in the paper. Ultimately, artifact evaluation is intended to encourage researchers to take extra care in conducting experiments in a reproducible way and to package experimental workflows and all related materials for making them accessible for others.

This year, 15 artifacts were submitted out of which 14 were successfully evaluated. This means that for half of the accepted papers at CGO 2018 an artifact was submitted. Each artifact received at least two reviews while most artifacts received three reviews form the Artifact Evaluation Committee of 20 researchers and engineers.

Our philosophy of artifact evaluation is that it should act as a mechanism to help authors to prepare their materials and replicate their experimental results. Therefore, the artifact evaluation process was very interactive with authors and reviewers communicating frequently to overcome technical issues with the goal to eventually allow the successful evaluation of the artifact by the reviewer. We would like to thank all authors and reviewers for their efforts in the artifact evaluation!

For the first time, the artifact evaluation of CGO has followed the new ACM Review and Badging guidelines\(^2\) which award papers up to three badges to indicate: 1. if an artifact is publicly available; 2. if an artifact has been functionally evaluated or if base functionality has been exceeded and the artifact is considered easily reusable; 3. if the experimental results have been independently replicated. These badges are printed onto the papers itself but also incorporated into the metadata in ACM’s digital library. This increases their visibility but also allows, for example, to

\(^{1}\)http://cTuning.org/ae/cgo2018.html

\(^{2}\)https://www.acm.org/publications/policies/artifact-review-badging

---
specifically search for papers with artifacts.

We were delighted to see that we were able to award the available and reusable to all 14 accepted artifacts. Eight artifacts were awarded the functionally badge, while six artifacts were awarded the reusable badge.

Michel Steuwer, University of Glasgow
Grigori Fursin, Dividiti/cTuning Foundation
Sponsors

Supporters

xii
ACM Student Research Competition

As in previous years, the 2018 International Symposium on Code Generation and Optimization (CGO) hosts the ACM Student Research Competition (SRC). We received eight poster abstracts, of which six were selected to compete in the ACM SRC. Out of these six submissions, four were selected in the graduate category and two were selected in the undergraduate category. All submissions were reviewed by five members of the selection committee. Each reviewer attributed up to 30 points to a submission and provided a short comment. The submissions were then ranked according to their total score. We would like to thank the General Co-Chairs and Program Co-Chairs for helping with our inquiries and for their initiative during the preparations of the SRC. Special thanks go to the members of the selection committee for reviewing all the abstracts on time and to all the authors who submitted a poster abstract, as well as all students who participated in the SRC.

Selection Committee
Simone Campanoni, Northwestern University (Co-Chair)
Aaron Smith, Microsoft Research and University of Edinburgh (Co-Chair)
Jeronimo Castrillon, Technische Universitat Dresden
Chris Fensch, Heriot-Watt University
Daniel Goodman, Oracle Labs UK
Timothy Jones, University of Cambridge
Sasa Misailovic, University of Illinois
Michel Steuwer, University of Glasgow
Register allocation is a mandatory task for most compilers. Often, global register allocation approaches such as linear scan or graph coloring are used. The flexibility of these approaches is limited since they process a whole method at once. We developed a novel trace register allocation framework which competes with global approaches in both compile time and code quality. Instead of processing the whole method, our allocator processes linear code segments (called traces) independently and can therefore select different allocation strategies based on the characteristics of a trace. This provides us with fine-grained control over the trade-off between compile time and peak performance.

When having a close look at compilation units, we see that not all parts of the method are equally important. Most optimizing compilers use global register allocation, i.e., they process a whole method at once. Compiler optimizations, such as inlining or code duplication, cause methods to become large. This poses two problems: (1) Register allocation time increases with method complexity, often in a non-linear fashion. (2) Different regions contribute differently to the overall performance of the compiled code. We assume that most time is spent in a small portion of the method. Global allocators do not differentiate between important and unimportant parts, or only in a limited way.

We solved the problems with a non-global approach based on traces, i.e., a sequence of sequentially executed code. Traces are constructed using profiling feedback. They are allocated independently, potentially using different strategies. We use strategies that yield good code quality for important traces and fast strategies for the others. Our framework consists of four components: (i) Trace Building partitions the blocks of the control-flow graph into traces. (ii) Global Liveness Analysis captures the liveness of variables at trace boundaries. (iii) Allocate Traces: For each trace, we select the most suitable register allocation strategy: Linear Scan for high-quality code, Bottom-Up for fast allocation. Due to the linear structure of traces, strategies are significantly simpler compared to a global algorithm. (iv) A global Data-flow Resolution pass is required since the locations of variables might be different across an inter-trace edge.

To validate our approach, we needed to answer the following questions: (1) Can a trace-base approach achieve peak performance similar to that of a global approach? (2) Can we improve compile time and/or peak performance by switching allocation strategies within a method? We implemented our approach in GraalVM and evaluated it using standard benchmarks, including DaCapo, SPECjvm2008 and SPECjbb2015. The results shows that our approach can compete with a global allocator. In addition, the flexibility allows us to save up to 40% allocation time.

Automated Partitioning of Data-Parallel Kernels using Polyhedral Compilation
Alexander Matz, Institute of Computer Engineering Heidelberg University

GPUs are well-established in domains outside of computer graphics, including scientific computing, artificial intelligence, data warehousing, and other computationally intensive areas. They differ in many regards from general-purpose CPUs, for instance, the highly parallel architecture, the bulk-synchronous execution model, and the data-parallel programming model. Due to their thread hierarchy, GPU workloads can be safely partitioned along boundaries formed by the blocks of threads, with the most efficient partitioning strategy depending on the application’s memory access patterns.

We leverage these observations for a concept that compiles single-GPU code for an execution on multiple GPUs. In this work, we present this concept (called Mekong) and provide a prototype performance analysis. The two key components are polyhedral compilation and a run-time system. Static analysis extracts memory access patterns within a kernel as a function of the kernel’s execution grid and
polyhedral codegeneration enables efficiently enumerating the buffer chunks making up the read or write set of a kernel instance. The runtime system consists of static, pre-compiled code that replaces host-side CUDA API functions such as buffer allocations, data movements, and kernel launches. These functions, originally performing their actions on a single GPU, are replaced by functions that perform the equivalent action on multiple GPUs instead. The replacements for memcopies and kernel launches require knowledge about the memory access patterns of the kernels of an application. For this, the run-time system relies on the code previously generated.

Having multiple instances of a buffer on multiple GPUs requires keeping track of the instances owning the most recent elements of the buffer. A tracker based on a red-black tree solves this with low time overhead and agnostic to the memory access pattern of any specific application. Full knowledge about which data is required on which GPU is only available at kernel launch time, which include the execution grid configuration and kernel arguments shaping memory access patterns. Due to this, data movements to and among GPUs are deferred at the API Call site and executed just before kernel launches instead, using the ownership information in the tracker.

The scope of the Mekong project is applications exhibiting memory access patterns that can be accurately modelled using Z-polyeder. The concept is implemented using the LLVM compiler infrastructure and libraries for polyhedral analysis and code generation. The performance and feasibility is verified based on three regular benchmarks selected from the Berkeley Dwarves. The hardware platform is a dual socket Intel node, equipped with 8 NVIDIA K80 cards, resulting in a total of 16 GPUs. We observe speedups of up to approximately 12x, and that the total fraction of the execution time spent for communication ranges from up to 16% for matrix multiplication down to less than 1% for an n-body simulation. This demonstrates the feasibility of the concept, as well as the low overhead at execution time. Future work will focus on extending the coverage, exploring the use of unified memory and overlap in between computation and communication.

**A C++/CUDA DSL for Object-oriented Programming with Structure-of-Arrays Data Layout**

Matthias Springer, Tokyo Institute of Technology

Previous work has demonstrated that Structure of Arrays (SOA) is an effective technique for achieving high performance on SIMD architectures such as GPUs or CPUs with vector instructions. Given a C struct or class, SOA groups all values of a field together. This is contrary to traditional Array of Structures (AOS), where the fields of every object are grouped together, and allows for more efficient memory access (parallel vector register loads, memory coalescing on GPUs) and cache usage.

The goal of our research is to provide a mechanism that lets programmers write object-oriented code in readable and concise AOS style, while automatically layouting data as SOA in the background. Existing C++ libraries like SoAx or the Intel SPMD Program Compiler (ispc) provide that functionality for C structs without methods. Our research extends that work to object-oriented programming. The contributions of our research prototype Ikra-Cpp are support for classes, methods and constructors with minimal changes to standard C++ notation in both C++ and CUDA. Our approach is closer to SoAx than to ispc: We are developing a lightweight C++ library/DSL instead of a new C++ compiler or invasive compiler extension, due to the high engineering effort of compiler construction.

Ikra-Cpp stores all data of a SOA class in a large statically allocated storage buffer. Whenever a field is accessed, Ikra-Cpp calculates the address at which the storage buffer is accessed. Our main insight is that this can be achieved efficiently and seamlessly (without changing AOS-style notation) in C++ with template metaprogramming, operator overloading and preprocessor macros. SOA objects are always referenced with pointers and cannot be stack allocated. Pointers to SOA objects do not point to actual data but are used to encode object IDs (fake pointers). For example, the address 0x1 is used to
reference the first object. Fields must be declared with special types (e.g., float). Their implicit conversion and assignment operators compute the object ID from their “this” pointer and access the storage buffer at the correct location.

Developing Ikra-Cpp is challenging from a technical point of view: Due to the complex combination of operator overloading and address computation, it is harder for C++ compilers to apply optimizations. While Ikra-Cpp is now on par with hand-written SOA code in gcc, clang still has trouble performing automatic loop vectorization.

A General Purpose Automatic Overlapped Tiling Technique in Polyhedral Frameworks
Jie Zhao, INRIA & École Normale Supérieure

Loop tiling is an essential transformation to exploit data locality and parallelism, being implemented as a general technique in numerous existing polyhedral frameworks including Pluto, PPCG, etc. and domain specific automatic code generators like PolyMage. Typically, automatic polyhedral frameworks either choose to implement the parallelogram tiling, avoiding the difficulty in performing code generation but leading to pipelined startup, or resort to sophisticated tiling shapes, e.g., diamond tiling, enabling concurrent startup but complicating both scheduling and code generation. Overlapped tiling is a technique designed to eliminate pipelined startup by modifying tile shapes obtained from existing frameworks, but no implementations for general purpose multicores have been reported in a polyhedral framework except those domain specific code generators, preventing its application in general purpose languages and missing a comparison with other state-of-the-art techniques.

We design and implement an automatic overlapped tiling technique for both general purpose and heterogeneous multicores in a polyhedral framework, neither being restricted to domain specific languages nor introducing sophisticated rescheduling in polyhedral frameworks. We first let a general polyhedral framework perform a parallelogram tiling and then expand the bounding faces of a tile by taking into consideration the constraints caused by inter-tile dependences, followed by expressing with well-defined nodes in a schedule-tree-based intermediate representation in general polyhedral frameworks, and finally achieve automatic code generation for overlapped tiling.

We conduct some preliminary experiments on both stencil computations and image processing pipelines extracted from the PolyMage benchmarks but written in general purpose languages, validating our technique is comparable with the state-of-the-art diamond tiling on stencil computations by finely selecting tiling sizes without modifying the scheduler of a polyhedral framework and achieves the performance the same as or even better than PolyMage for image processing pipelines without resorting to domain specific languages.

LLVM Superoptimizer
Siddharth Bhatia and Yash Sinha, BITS Pilani

Although superoptimizers have the capability to discover the best machine specific optimizations without writing manually the code transformations or searching for optimal instruction sequence, they are yet to be widely used. We extend the GreenThumb framework and present an LLVM superoptimizer. We take up the LLVM Intermediate representation (IR) instructions, exploit the unique strengths of enumerative, stochastic, and symbolic (SAT-solver-based) search algorithms and generate equivalent but faster instructions.

We extend the framework for parallel architectures that handles complex instruction types such as vectors. A parallel to sequential optimisation is generally thought of as unlikely. But due to
parallelization, certain optimisations cannot be introduced, which could have been present in the sequential version of a code. We recognize the overhead of fork-join model in parallel programs, serialize the code to a sequential version, and superoptimize the resulting code. Consequently, the generated code runs faster as it avoids bottlenecks such as WAR dependency, wait for resources, network communication latency, pipeline stalls etc. This benefits single-core machines.

Secondly, we superoptimize to balance the workloads between two parallel running blocks to optimize processing time and resource accesses. For example, in a CPU that implements simultaneous multithreading, the superoptimization aids conflict resolution on shared resources so that concurrent running threads do not hamper each-other’s performance.

Fast and Flexible Instruction Selection with Constraints
Patrick Thier, Institut fur Computersprachen Technische Universitat Wien

Instruction selection is an important component of code generation and a variety of techniques have been proposed in the past, ranging from simple macro expansion that selects target instructions based on a single node of the intermediate representation (IR) to global instruction selection methods on the whole IR graph. Simpler methods tend to be faster, but produce slower code. Instruction selection in JIT compilers is especially difficult as fast techniques are needed while still maintaining good code quality. Bottom-up tree-parsing automata, as generated by burg, offer a good trade-off between speed and code quality. The Java HotSpot Server compiler is a prominent example for using this technique. Code quality in tree-parsing instruction selectors can be further improved by using dynamic costs to gain flexibility and to handle situations such as read-modify-write instructions efficiently. The disadvantage of dynamic costs is that the tree grammar can no longer be turned into a tree-parsing automaton, limiting their use to slower tree parsing methods like dynamic programming at compiler run-time.

In this work we introduce constraints as a replacement for dynamic costs. The typical use of dynamic costs in a tree grammar is to determine whether a rule is applicable, by giving that rule a fixed, small cost if it is (based on contextual information not expressible in a tree grammar), and an infinite cost if it is not. Constraints are a formalization of these applicability tests. In a tree grammar with constraints, rules are assigned a fixed cost that is optionally augmented with a condition (the constraint). As only static costs are used, all possible states can be pre-computed and the grammar can be turned into a fast tree-parsing automaton. At instruction-selection-time the state of a node is looked up using the node’s operator and the states of the node’s children as in traditional tree-parsing automata. If any of the optimal rules of a state has a constraint assigned, the constraint is evaluated and if evaluation fails, a fall-back state not utilizing the constrained rule is used instead.

At instruction selector generation these fall-back states need to be computed accordingly. The concept was implemented in a tree-parser generator called cdburg. To evaluate the impact of constraints on compilation time, lcc was modified to use cdburg instead of lburg. In addition cdburg was utilized as instruction selector in a Java VM JIT compiler that is under development. We recorded instruction selection speedups by a factor of 1.33 – 1.89 while maintaining the same code quality as if dynamic costs were used.
Contents

Frontmatter
Message from the General Chairs .................................................. iii
Message from the Program Chairs .................................................. v
CGO 2018 Organization .................................................................. vi
Report from the Artifact Evaluation Committee ........................... x
Sponsors and Supporters ............................................................... xii
Student Research Competition ...................................................... xiii

Keynote
Biological Computation (Keynote)
  Sara-Jane Dunn — Microsoft Research, UK .............................. 1

Managed Runtimes
SIMD Intrinsics on Managed Language Runtimes
  Alen Stojanov, Ivaylo Toskov, Tiark Rompf, and Markus Püschel — ETH Zurich, Switzerland, Purdue University, USA ............................... 2
CollectionSwitch: A Framework for Efficient and Dynamic Collection Selection
  Diego Costa and Artur Andrzejak — University of Heidelberg, Germany ......................................................... 16
Analyzing and Optimizing Task Granularity on the JVM
  Andrea Rosà, Eduardo Rosales, and Walter Binder — University of Lugano, Switzerland ...................................................... 27

Resilience and Security
Automating Efficient Variable-Grained Resiliency for Low-Power IoT Systems
  Sara S. Baghsorkhi and Christos Margiolas — Intel, USA .................. 38
Resilient Decentralized Application Repackaging Detection Using Logic Bombs
  Qiang Zeng, Lannan Luo, Zhiyun Qian, Xiaojiang Du, and Zhoujun Li — Temple University, USA; University of South Carolina, USA; University of California at Riverside, USA; Beihang University, China .................. 50
nAdroid: Statically Detecting Ordering Violations in Android Applications
  Xinwei Fu, Dongyoon Lee, and Changhee Jung — Virginia Tech, USA ......................................................... 62
SGXElide: Enabling Enclave Code Secrecy via Self-Modification
  Erick Bauman, Huibo Wang, Mingwei Zhang, and Zhiqiang Lin — University of Texas at Dallas, USA; Intel Labs, USA ............................................ 75

Best Paper Finalists
Poker: Permutation-Based SIMD Execution of Intensive Tree Search by Path Encoding
  Feng Zhang and Jingling Xue — UNSW, Australia ............................ 87
High Performance Stencil Code Generation with Lift
  Bastian Hagedorn, Larisa Stoltzfus, Michel Steuwer, Sergei Gorlatch, and Christophe Dubach — University of Münster, Germany; University of Edinburgh, UK; University of Glasgow, UK ......................................................... 100
Qubit Allocation
  Marcos Yukio Siraichi, Vinicius Fernandes dos Santos, Sylvain Collange, and Fernando Magno Quintao Pereira — Federal University of Minas Gerais, Brazil; Inria, France; University of Rennes, France; CNRS, France; IRISA, France ................................................................. 113
Dominance-Based Duplication Simulation (DBDS): Code Duplication to Enable Compiler Optimizations
  David Leopoldseder, Lukas Stadler, Thomas Würthinger, Josef Eisl, Doug Simon, and Hanspeter Mössenböck — JKU Linz, Austria; Oracle Labs, Austria; Oracle Labs, Switzerland ......................................................... 126

Linear Algebra and Vectorization
The Generalized Matrix Chain Algorithm
  Henrik Barthels, Marcin Copik, and Paolo Bientinesi — RWTH Aachen University, Germany ............................... 138
CVR: Efficient Vectorization of SpMV on X86 Processors
  Biwei Xie, Jianfeng Zhan, Xu Liu, Wanling Gao, Zhen Jia, Xiwen He, and Lixin Zhang — Institute of Computing Technology at Chinese Academy of Sciences, China; University at Chinese Academy of Sciences, China; College of William and Mary, USA; Princeton University, USA ......................................................... 149
Look-Ahead SLP: Auto-vectorization in the Presence of Commutative Operations
Vasileios Porpodas, Rodrigo C. O. Rocha, and Luis F. W. Goes — Intel, USA; University of Edinburgh, UK; PUC-MG, Brazil 163

Conflict-Free Vectorization of Associative Irregular Applications with Recent SIMD Architectural Advances
Peng Jiang and Gagan Agrawal — Ohio State University, USA; The Ohio State University, USA 175

Static and Dynamic Analysis
Scalable Concurrency Debugging with Distributed Graph Processing
Long Zheng, Xiaofei Liao, Hai Jin, Jieshan Zhao, and Qinggang Wang — Huazhong University of Science and Technology, China 188

Lightweight Detection of Cache Conflicts
Probir Roy, Shuawen Leon Song, Sriram Krishnamoorthy, and Xu Liu — College of William and Mary, USA; Pacific Northwest National Laboratory, USA 200

CUDAdvisor: LLVM-Based Runtime Profiling for Modern GPUs
Du Shen, Shuawen Leon Song, Ang Li, and Xu Liu — College of William and Mary, USA; Pacific Northwest National Laboratory, USA 214

May-Happen-in-Parallel Analysis with Static Vector Clocks
Qing Zhou, Lian Li, Lei Wang, Jingling Xue, and Xiaobing Feng — Institute of Computing Technology at Chinese Academy of Sciences, China; University at Chinese Academy of Sciences, China; UNSW, Australia 228

Memory Usage Optimisation
DeLICM: Scalar Dependence Removal at Zero Memory Cost
Michael Kruse and Tobias Grosser — Inria, France; ETH Zurich, Switzerland 241

Loop Transformations Leveraging Hardware Prefetching
Savvas Sioutas, Sander Stuijk, Henk Corporaal, Twan Basten, and Lou Somers — Eindhoven University of Technology, Netherlands 254

Transforming Loop Chains via Macro Dataflow Graphs
Eddie C. Davis, Michelle Mills Strout, and Catherine Olschanowsky — Boise State University, USA; University of Arizona, USA 265

Local Memory-Aware Kernel Perforation
Daniel Maier, Biagio Cosenza, and Ben Juurlink — TU Berlin, Germany 278

Program Generation and Synthesis
AutoPA: Automatically Generating Active Driver from Original Passive Driver Code
Jia-Ju Bai, Yu-Ping Wang, and Shi-Min Hu — Tsinghua University, China 288

Synthesizing an Instruction Selection Rule Library from Semantic Specifications
Sebastian Buchwald, Andreas Fried, and Sebastian Hack — KIT, Germany; Saarland University, Germany 300

Synthesizing Programs That Expose Performance Bottlenecks
Luca Della Toffola, Michael Pradel, and Thomas R. Gross — ETH Zurich, Switzerland; TU Darmstadt, Germany 314

Program Generation for Small-Scale Linear Algebra Applications
Daniele G. Spampinato, Diego Fabregat-Traver, Paolo Bientinesi, and Markus Püschel — ETH Zurich, Switzerland; RWTH Aachen University, Germany 327

Compilation for Specialised Domains
Optimal DNN Primitive Selection with Partitioned Boolean Quadratic Programming
Andrew Anderson and David Gregg — Trinity College Dublin, Ireland 340

Register Allocation for Intel Processor Graphics
Wei-Yu Chen, Guei-Yuan Lueh, Pratik Ashar, Kaiyu Chen, and Buqi Cheng — Intel, USA; Intel, India 352

A Compiler for Cyber-Physical Digital Microfluidic Biochips
Christopher Curtis, Daniel Grissom, and Philip Brisk — University of California at Riverside, USA; Azusa Pacific University, USA 365

Author Index 378