Activity Report 2016

Team COMPSYS

Compilation and Embedded Computing Systems

RESEARCH CENTER
Grenoble - Rhône-Alpes

THEME
Architecture, Languages and Compilation

Table of contents

Team COMPSYS

Creation of the Project-Team: 2004 January 01, updated into Team: 2016 January 01, end of the Team: 2016 December 31

Compsys is located at Ecole Normale Supérieure de Lyon.

Keywords:

Computer Science and Digital Science:

2.1.1. - Semantics of programming languages
2.1.6. - Concurrent programming
2.1.10. - Domain-specific languages
2.2.1. - Static analysis
2.2.5. - GPGPU, FPGA, etc.
2.4.1. - Analysis
6.2.6. - Optimization
6.2.7. - High performance computing
7.2. - Discrete mathematics, combinatorics

Other Research Topics and Application Domains:

6.6. - Embedded systems
9.4.1. - Computer science

1. Members

Research Scientists

Alain Darte [Team leader, CNRS, Senior Researcher, HDR]
Tomofumi Yuki [Inria, Researcher]

Faculty Member

Paul Feautrier [ENS Lyon, Emeritus Professor, HDR]

PhD Students

Alexandre Isoard [ENS Lyon, until May 2016]
Guillaume Iooss [ENS Lyon/Colorado State University, until July 2016]

Visiting Scientists

Hammami Emna [PhD student, Tunis University, April-June 2016]
Julien Versaci [Master 2 internship, Claude Bernard University, April-June 2016]
Waruna Ranasinghe [PhD student, Colorado State University, June-August 2016]

Administrative Assistant

Chiraz Benamor [ENS]

2. Overall Objectives

2.1. Introduction

Keywords: Compilation, code analysis, code optimization, memory optimization, combinatorial optimization, algorithmics, polyhedral optimization, hardware accelerators, high-level synthesis, high-performance computing, multicore, GPU, FPGA, DSP.

Compsys has been developing compilation techniques, more precisely code analysis and code optimization techniques, to help programming or designing "embedded computing systems" and platforms for "small" HPC (High-Performance Computing). The team focused first on both low-level (back-end) optimizations and high-level (front-end, mainly source-to-source) transformations, for specialized embedded processors (DSP) and high-level synthesis of hardware accelerators (FPGA). More recent activities included a shift towards abstract interpretation and program termination, the compilation for GPUs and multicores, and the analysis of parallel languages. The main characteristic of Compsys is its use of algorithmic and formal methods (with graph algorithms, linear programming, polyhedral optimizations) to address code analysis and optimization problems (e.g., termination, register allocation, memory optimizations, scheduling, automatic generation of interfaces) and the validation of these techniques through the development of compilation tools.

Compsys started as an Inria project in 2004, after 2 years of maturation. This first period of Compsys, Compsys I, was positively evaluated in Spring 2007 after its first 4 years period (2004-2007). It was again evaluated by AERES in 2009, as part of the general evaluation of LIP, and got the best possible mark, A+. The second period (2007-2012), Compsys II, was again evaluated positively by Inria in Spring 2012 and formally prolonged into Compsys III at the very end of 2012. In 2013, Fabrice Rastello moved to Grenoble first to expand the activities of Compsys in the context of Giant, a R&D technology center with several industrial and academic actors. He left officially the team in 2014 to work on his own. The research directions of Compsys III then followed the lines presented in the synthesis report provided for the 2012 evaluation, including a shift towards the compilation of streaming programming, the analysis and optimizations of parallel languages, and an even stronger focus on polyhedral optimizations and their extensions. While Christophe Alias was mostly involved in his developments of the Zettice/XTREMLOGIC start-up, the hiring of Laure Gonnord (in 2013) and Tomofumi Yuki (in 2014) added new forces on the code analysis research aspects and on HPC polyhedral-related topics. However, Christophe Alias and Laure Gonnord left the team in Sep. 2015. Reaching the limit of 12 years, the project-team ended officially in Dec. 2015, but with no possible future as a new project, because it was below critical mass. Compsys was nevertheless extended as an Inria team until Dec. 2016, in particular to allow the (positive) final Inria evaluation in Spring 2016 and to let the last participants think about their future. At the end of 2016, Tomofumi Yuki moved back to Rennes in the Cairn Inria team, Paul Feautrier is still a member of LIP as an emeritus professor at ENS-Lyon but is now a long term visitor to the Parkas Inria team in Paris, and Alain Darte remains CNRS researcher at LIP, ENS-Lyon, but is not affiliated to Inria anymore.

Section 2.2 defines the general context of the team's activities. Section 2.3 presents the research objectives and main achievements in Compsys I, i.e., until 2007. Section 2.4 shows how the research directions of the team were modified for Compsys II and outlines the main results we obtained in this period (until 2012). Finally, Section 2.5 summarizes the goals and achievements of Compsys III. More details can be found in the annual Inria reports. As for the highlights of the past year, i.e., 2016, they are given in Section 5.1.

2.2. General Presentation

Classically, an embedded computer is a digital system that is part of a larger system and that is not directly accessible to the user. Examples are appliances like phones, TV sets, washing machines, game platforms, or even larger systems like radars and sonars. In particular, this computer is not programmable in the usual way. Its program, if it exists, is supplied as part of the manufacturing process and is seldom (or ever) modified thereafter. As the embedded systems market grows and evolves, this view of embedded systems is becoming obsolete and tends to be too restrictive. Many aspects of general-purpose computers apply to modern embedded platforms. Nevertheless, embedded systems remain characterized by a set of specialized application domains, rigid constraints (cost, power, efficiency, heterogeneity), and its market structure. The term *embedded system* has been used for naming a wide variety of objects. More precisely, there are two categories of so-called *embedded systems*: a) control-oriented and hard real-time embedded systems (automotive, plant control, airplanes, etc.); b) compute-intensive embedded systems (signal processing, multi-media, stream processing) processing large data sets with parallel and/or pipelined execution. Compsys is primarily concerned with this second type of embedded systems, referred to as *embedded computing systems*.

Today, the industry sells many more embedded processors than general-purpose processors; the field of embedded systems is one of the few segments of the computer market where the European industry still has a substantial share, hence the importance of embedded system research in the European research initiatives. Our priority towards embedded software was motivated by the following observations: a) the embedded system market was expanding, among many factors, one can quote pervasive digitalization, low-cost products, appliances, etc.; b) research on software for embedded systems was poorly developed in France, especially if one considers the importance of actors like Alcatel, STMicroelectronics, Matra, Thales, etc.; c) since embedded systems increase in complexity, new problems are emerging: computer-aided design, shorter time-to-market, better reliability, modular design, component reuse, etc.

A specific aspect of embedded computing systems is the use of various kinds of processors, with many particularities (instruction sets, registers, data and instruction caches, now multiple cores) and constraints (code size, performance, storage, power). The development of *compilers* is crucial for this industry, as selling a platform without its programming environment and compiler would not be acceptable. To cope with such a range of different processors, the development of robust, generic (retargetable), though efficient compilers is mandatory. Unlike standard compilers for general-purpose processors, compilers for embedded processors and hardware accelerators can be more aggressive (i.e., take more time to optimize) for optimizing some important parts of applications. This opens a new range of optimizations. Another interesting aspect is the introduction of platform-independent intermediate languages, such as Java bytecode, that is compiled dynamically at runtime (aka just-in-time). Extreme lightweight compilation mechanisms that run faster and consume less memory have to be developed. The introduction of intermediate languages such as OpenCL was also a sign of the need for portability (as well as productivity) across diverse (if not heterogeneous) platforms. One of the initial objectives of Compsys was thus to revisit existing compilation techniques in the context of such embedded computing systems, to deconstruct some of these techniques, to improve them, and to develop new techniques taking constraints of embedded processors and platforms into account.

As for *high-level synthesis* (HLS), several compilers/systems have appeared, after some first unsuccessful industrial attempts in the past. These tools are mostly based on C or C++ as for example SystemC, VCC, CatapultC, Altera C2H, Pico-Express, Vivado HLS. Academic projects also exist (or existed) such as Flex and Raw at MIT, Piperench at Carnegie-Mellon University, Compaan at the University of Leiden, Ugh/Disydent at LIP6 (Paris), Gaut at Lester (Bretagne), MMAlpha (Insa-Lyon), and others. In general, the support for parallelism in HLS tools is minimal, especially in industrial tools. Also, the basic problem that these projects have to face is that the definition of performance is more complex than in classical systems. In fact, it is a multi-criteria optimization problem and one has to take into account the execution time, the size of the program, the size of the data structures, the power consumption, the manufacturing cost, etc. The impact of the compiler on these costs is difficult to assess and control. Success will be the consequence of a detailed knowledge of all steps of the design process, from a high-level specification to the chip layout. A strong cooperation of the compilation and chip design communities was needed. The main expertise in Compsys for this aspect was in the *parallelization* and optimization of *regular computations*. Hence, we targeted applications with a large potential parallelism, but we attempted to integrate our solutions into the big picture of CAD environments.

More generally, the aims of Compsys were to develop new compilation and optimization techniques for the field of embedded computing system design. This field is large, and Compsys did not intend to cover it in its entirety. As previously mentioned, we were mostly interested in the automatic design of accelerators, for example designing a VLSI or FPGA circuit for a digital filter, or later GPUs and multicores, and in the development of new back-end compilation strategies for embedded processors. We studied code transformations that optimize features such as execution time, power consumption, code and die size, memory constraints, and compiler reliability. These features are related to embedded systems but some are not specific to them. The code transformations we developed were both at source level and at assembly level. A specificity of Compsys has always been to mix a solid theoretical basis for all code optimizations we introduced with algorithmic/software developments. Within Inria, our project was related to the “architecture and compilation” theme, more precisely code optimization, as some of the research conducted in Parkas (previously known as Alchemy), Alf (previously known as Caps), Camus, and to high-level architectural synthesis, as some of the research in Cairn.

At the end of the 90s, most french researchers working on high-performance computing (automatic parallelization, languages, operating systems, networks) moved to grid computing. We thought that applications, industrial needs, and research problems were more interesting in the design of embedded platforms. Furthermore, we were convinced that our expertise on high-level code transformations could be more useful in this field. This is the reason why Tanguy Risset came to Lyon in 2002 to create the Compsys team with Anne Mignotte and Alain Darte, before Paul Feautrier, Antoine Fraboulet, and Fabrice Rastello joined the group. Before integrating the team, all Compsys members had a background in automatic parallelization, and high-level program analyses and transformations. Paul Feautrier was the initiator of the polyhedral model for program transformations around 1990 and, before coming to Lyon, started to be more interested in programming models and optimizations for embedded applications, in particular through collaborations with Philips. Alain Darte worked on mathematical tools and algorithmic issues for parallelism extraction in programs. He became interested in the automatic generation of hardware accelerators, thanks to his stay at HP Labs in the Pico project in 2001. Antoine Fraboulet did a PhD with Anne Mignotte – who was working on high-level synthesis (HLS) – on code and memory optimizations for embedded applications. Fabrice Rastello did a PhD on tiling transformations for parallel machines, then was hired by STMicroelectronics where he worked on assembly code optimizations for embedded processors. Tanguy Risset worked for a long time on the synthesis of systolic arrays, being the main architect of the HLS tool MMAlpha. Christophe Alias did a PhD on algorithm recognition for program optimizations and parallelization, and two post-docs, one in Compsys on array contraction, one in Ohio State University with Prof. P. Sadayappan on memory optimizations. Laure Gonnord did a PhD on invariant generation and program analysis and became interested on compilation and code generation since her postdoc in the team. Finally, Tomofumi Yuki did a PhD on polyhedral programming environments and optimizations (in Colorado State University, with Prof. S. Rajopadhye) before a post-doc on polyhedral HLS in the Cairn team (Rennes).

To understand why we think automation in our field is highly important, it may be worth to quote Bob Rau and his colleagues (IEEE Computer, Sep. 2002):

"Engineering disciplines tend to go through fairly predictable phases: ad hoc, formal and rigorous, and automation. When the discipline is in its infancy and designers do not yet fully understand its potential problems and solutions, a rich diversity of poorly understood design techniques tends to flourish. As understanding grows, designers sacrifice the flexibility of wild and woolly design for more stylized and restrictive methodologies that have underpinnings in formalism and rigorous theory. Once the formalism and theory mature, the designers can automate the design process. This life cycle has played itself out in disciplines as diverse as PC board and chip layout and routing, machine language parsing, and logic synthesis.

We believe that the computer architecture discipline is ready to enter the automation phase. Although the gratification of inventing brave new architectures will always tempt us, for the most part the focus will shift to the automatic and speedy design of highly customized computer systems using well-understood architecture and compiler technologies.”

We share this view of the future of architecture and compilation. Without targeting too ambitious objectives, we were convinced of two complementary facts: a) the mathematical tools developed in the past for manipulating programs in automatic parallelization were lacking in high-level synthesis and embedded computing optimizations and, even more, they started to be rediscovered frequently in less mature forms, b) before being able to really use these techniques in HLS and embedded program optimizations, we needed to learn a lot from the application side, from the electrical engineering side, and from the embedded architecture side. Our primary goal was thus twofold: to increase our knowledge of embedded computing systems and to adapt/extend code optimization techniques, primarily designed for high performance computing, to the special case of embedded computing systems. In the initial Compsys proposal, we proposed four research directions, centered on compilation methods for embedded applications, both for software and accelerators design:

- Code optimization for specific processors (mainly DSP and VLIW processors);
- Platform-independent loop transformations (including memory optimization);
- Silicon compilation and hardware/software codesign;
- Development of polyhedral (but not only) optimization tools.

These research activities were primarily supported by a marked investment in polyhedra manipulation tools and, more generally, solid mathematical and algorithmic studies, with the aim of constructing operational software tools, not just theoretical results. Hence the fourth research theme was centered on the development of these tools.

2.3. Summary of Compsys I Achievements

The main achievements of Compsys I were the following:

- The development of a strong collaboration with the compilation group at STMicroelectronics, with important results in aggressive optimizations for instruction cache and register allocation.
- New results on the foundation of high-level program transformations, including scheduling techniques for process networks and a general technique for array contraction (memory reuse) based on the theory of lattices.
- Many original contributions with partners closer to hardware constraints, including CEA, related to SoC simulation, hardware/software interfaces, power models, and simulators.

The Compsys team has been evaluated by Inria for the first time in April 2007. The evaluation, conducted by Erik Hagersted (Uppsala University), Vinod Kathail (Synfora, inc), J. (Ram) Ramanujam (Baton Rouge University) was positive. Compsys I thus continued into Compsys II for 4-5 years but in a new configuration as Tanguy Risset (who was hired professor at Insa-Lyon) and Antoine Fraboulet (assistant professor at Insa-Lyon) left the project to follow research directions closer to their host laboratory at Insa-Lyon.

2.4. Summary of Compsys II Achievements

Due to Compsys size reduction (from 5 permanent researchers to 3 in 2008, then 4 again in 2009), the team then focused, in Compsys II, on two research directions only:

- Code generation for embedded processors, on the two opposite, though connected, aspects: aggressive compilation and just-in-time compilation.
- High-level program analysis and transformations for high-level synthesis tools.

The main achievements of Compsys II were:

- the great success of the collaboration with STMicroelectronics with many deep results on SSA (Static Single Assignment), register allocation, liveness scalar analysis, and intermediate program representations;
- the design of high-level program analysis, optimizations, and tools, mainly related to high-level synthesis (some leading to the development of the Zettice start-up), including liveness array analysis, memory folding, as well as program (while loops) termination.

For more details on the past years of Compsys II, see the previous annual reports from 2008 to 2012. Compsys II was positively evaluated in Spring 2012 by Inria. The evaluation committee members were Walid Najjar (University of California Riverside), Paolo Faraboschi (HP Labs), Scott Mahlke (University of Michigan), Pedro Diniz (University of Southern California), Peter Marwedel (TU Dortmund), and Pierre Paulin (STMicroelectronics, Canada), the last three assigned specifically to Compsys.

2.5. Summary of Compsys III Achievements

For Compsys III, the changes in the permanent members (departure of Fabrice Rastello and arrival of Laure Gonnord, while she was only external collaborator of Compsys until Sep. 2013) reduced the forces on back-end code optimizations, and in particular dynamic compilation, but increased (for a short period only) the forces on program analysis. In this context, Compsys III has continued to develop fundamental concepts or techniques whose applicability should go beyond a particular architectural or language trend, as well as stand-alone tools (either as proofs of concepts or to be used as basic blocks in larger tools/compilers developed by others) and our own experimental prototypes. One of the main objectives of Compsys III has been to try to push the polyhedral model beyond its present limits both in terms of analysis techniques (possibly integrating approximation and runtime support) and of applicability (e.g., analysis of parallel or streaming languages, program verification, compilation towards accelerators such as GPU or multicores). The hiring of Tomofumi Yuki supported this new direction. The achievements of Compsys III include work on:

- Back-end code analysis including fast scalar liveness analysis, register spilling analysis, pointer and array analysis.
- Polyhedral code analysis and optimizations, including communication analysis for kernel offloading to FPGA and GPU, analysis of while loops, analysis of parallel and streaming languages (liveness, memory folding, race detection), parametric tiling, polynomial extensions.

Compsys III was positively evaluated in Spring 2016 (with regrets with respect to its undesired stop) in Spring 2016. This evaluation also served as the final evaluation of Compsys after 12 years. The evaluation committee members were Krzystof Czarnecki (University of Waterloo), Benoît Dupont de Dinechin (Kalray), Nikil Dutt (UC Irvine), Walid Najjar (UC Riverside), Kristoffer Rose (Two Sigma Investments, NYW), Christian Schulte (KTH Royal Institute of Technology), Tulika Mitra (NUS), J. (Ram) Ramanujam (Lousiana State Univ.), Kathryn S. McKinley (chair, Microsoft), the last three being directly responsible for Compsys evaluation.

More details on the 2013, 2014, 2015 activities are given in the corresponding annual reports (see also the synthesis report provided for the 2016 evaluation). The new results for this year (2016) are given in Section 5.1 (highlights) and from Section 7.1 to 7.7 (new results).

3. Research Program

3.1. Architecture and Compilation Trends

The embedded system design community is facing two challenges:

- The complexity of embedded applications is increasing at a rapid rate.
- The needed increase in processing power is no longer obtained by increases in the clock frequency, but by increased parallelism.

While, in the past, each type of embedded application was implemented in a separate appliance, the present tendency is toward a universal hand-held object, which must serve as a cell-phone, as a personal digital assistant, as a game console, as a camera, as a Web access point, and much more. One may say that embedded applications are of the same level of complexity as those running on a PC, but they must use a more constrained platform in terms of processing power, memory size, and energy consumption. Furthermore, most of them depend on international standards (e.g., in the field of radio digital communication), which are evolving rapidly. Lastly, since ease of use is at a premium for portable devices, these applications must be integrated seamlessly to a degree that is unheard of in standard computers.

All of this dictates that modern embedded systems retain some form of programmability. For increased designer productivity and reduced time-to-market, programming must be done in some high-level language, with appropriate tools for compilation, run-time support, and debugging. This does not mean however that all embedded systems (or all of an embedded system) must be processor based. Another solution is the use of field programmable gate arrays (FPGA), which may be programmed at a much finer grain than a processor, although the process of FPGA "programming" is less well understood than software generation. Processors are better than application-specific circuits at handling complicated control and unexpected events. On the other hand, FPGAs may be tailored to just meet the needs of their application, resulting in better energy and silicon area usage. It is expected that most embedded systems will use a combination of general-purpose processors, specific processors like DSPs, and FPGA accelerators (or even low-power GPUs). Such a combination DSP+FPGA is already present in recent versions of the Atom Intel processor.

As a consequence, parallel programming, which has long been confined to the high-performance community, must become the common place rather than the exception. In the same way that sequential programming moved from assembly code to high-level languages at the price of a slight loss in performance, parallel programming must move from low-level tools, like OpenMP or even MPI, to higher-level programming environments. While fully-automatic parallelization is a Holy Grail that will probably never be reached in our lifetimes, it will remain as a component in a comprehensive environment, including general-purpose parallel programming languages, domain-specific parallelizers, parallel libraries and run-time systems, back-end compilation, dynamic parallelization. The landscape of embedded systems is indeed very diverse and many design flows and code optimization techniques must be considered. For example, embedded processors (micro-controllers, DSP, VLIW) require powerful back-end optimizations that can take into account hardware specificities, such as special instructions and particular organizations of registers and memories. FPGA and hardware accelerators, to be used as small components in a larger embedded platform, require "hardware compilation", i.e., design flows and code generation mechanisms to generate non-programmable circuits. For the design of a complete system-on-chip platform, architecture models, simulators, debuggers are required. The same is true for multicores of any kind, GPGPU ("general-purpose" graphical processing units), CGRA (coarse-grain reconfigurable architectures), which require specific methodologies and optimizations, although all these techniques converge or have connections. In other words, embedded systems need all usual aspects of the process that transforms some specification down to an executable, software or hardware. In this wide range of topics, Compsys concentrated on the code optimizations aspects (and the associated analysis) in this transformation chain, restricting to compilation (transforming a program to a program) for embedded processors and programmable accelerators, and to high-level synthesis (transforming a program into a circuit description) for FPGAs.

Actually, it is not a surprise to see compilation and high-level synthesis getting closer (in the last 10 years now). Now that high-level synthesis has grown up sufficiently to be able to rely on place-and-route tools, or even to synthesize C-like languages, standard techniques for back-end code generation (register allocation, instruction selection, instruction scheduling, software pipelining) are used in HLS tools. At the higher level, programming languages for programmable parallel platforms share many aspects with high-level specification languages for HLS, for example the description and manipulations of nested loops, or the model of computation/communication (e.g., Kahn process networks and its many "streaming" variants). In all aspects, the frontier between software and hardware is vanishing. For example, in terms of architecture, customized processors (with processor extensions as first proposed by Tensilica) share features with both general-purpose processors and hardware accelerators. FPGAs are both hardware and software as they are fed with "programs" representing their hardware configurations.

In other words, this convergence in code optimizations explains why Compsys studied both program compilation and high-level synthesis, and at both front-end and back-end levels, the first one acting more at the granularity of memories, transfers, and multiple cores, the second one more at the granularity of registers, system calls, and single core. Both levels must be considered as they interact with each other. Front-end optimizations must be aware of what back-end optimizations will do, as single core performance remain the basis for good parallel performances. Some front-end optimizations even act directly on back-end features, for example register tiling considered as a source-level transformation. Also, from a conceptual point of view, the polyhedral techniques developed by Compsys are actually the symbolic front-end counterpart, for structured loops, of back-end analysis and optimizations of unstructured programs (through control-flow graphs), such as dependence analysis, scheduling, lifetime analysis, register allocation, etc. A strength of Compsys was to juggle with both aspects, the first one based on graph theory with SSA-type optimizations, the other on polyhedra representing loops, and to exploit the correspondence between both. This has still to be exploited, for applying polyhedral techniques to more irregular programs. Besides, Compsys had a tradition of building free software tools for linear programming and optimization in general, as needed for our research.

3.1.1. Compilation and Languages Issues in the Context of Embedded Processors, "Embedded Systems", and Programmable Accelerators

Compilation is an old activity, in particular back-end code optimizations. The development of embedded systems was one of the reasons for the revival of compilation activities as a research topic. Applications for embedded computing systems generate complex programs and need more and more processing power. This evolution is driven, among others, by the increasing impact of digital television, the first instances of UMTS networks, and the increasing size of digital supports, like recordable DVD, and even Internet applications. Furthermore, standards are evolving very rapidly (see for instance the successive versions of MPEG). As a consequence, the industry has focused on programmable structures, whose flexibility more than compensates for their larger size and power consumption. The appliance provider has a choice between hard-wired structures (Asic), special-purpose processors (Asip), (quasi) general-purpose processors (DSP for multimedia applications), and now hardware accelerators (dedicated platforms – such as those developed by Thales or the CEA –, or more general-purpose accelerators such as GPUs or even multicores, even if these are closer to small HPC platforms than truly embedded systems). Our cooperation with STMicroelectronics, until 2012, focused on investigating the compilation for specialized processors, such as the ST100 (DSP processor) and the ST200 (VLIW DSP processor) family. Even for this restricted class of processors, the diversity is large, and the potential for instruction level parallelism (SIMD, MMX), the limited number of registers and the small size of the memory, the use of direct-mapped instruction caches, of predication, generated many open problems. Our goal was to contribute to their understanding and their solutions.

An important concept to cope with the diversity of platforms is the concept of *virtualization*, which is a key for more portability, more simplicity, more reliability, and of course more security. This concept – implemented at low level through binary translation and just-in-time (JIT) compilation [1] – consists in hiding the architecture-dependent features as long as possible during the compilation process. It has been used for a while for servers such as HotSpot, a bit more recently for workstations, and now for embedded computing. The same needs drive the development of intermediate languages such as OpenCL to, not necessarily hide, but at least make more uniform, the different facets of the underlying architectures. The challenge is then to design and compile high-productivity and high-performance languages [2] (coping with parallelism and heterogeneity) that can be ported to such intermediate languages, or to architecture-dependent runtime systems. The offloading of computation kernels, through source-to-source compilation, targeting back-end C dialects, has the same goals: to automate application porting to the variety of accelerators.

For JIT compilation, the compactness of the information representation, and thus its pertinence, is an important criterion for such late compilation phases. Indeed, the intermediate representation (IR) is evolving not only from a target-independent description to a target-dependent one, but also from a situation where the compilation time is almost unlimited (cross-compilation) to one where any type of resource is limited. This is one of the reasons why static single assignment (SSA), a sparse compact representation of liveness information, became popular in embedded compilation. If time constraints are common to all JIT compilers (not only for

[1] *Aggressive compilation* consists in allowing more time to implement more complete and costly solutions: compilation time is less relevant than execution time, size, and energy consumption of the produced code, which can have a critical impact on the cost and quality of the final product. The application is usually cross-compiled, i.e., compiled on a powerful platform distinct from the target processor. *Just-in-time compilation*, on the other hand, corresponds to compiling applets on demand on the target processor. The code can be uploaded or sold separately on a flash memory. Compilation is performed at load time and even dynamically during execution. The optimization heuristics, constrained by time and limited resources, are far from being aggressive. They must be fast but smart enough.

[2] For examples of such languages, see the keynotes event we organized in 2013: http://labexcompilation.ens-lyon.fr/hpc-languages.

embedded computing), the benefit of using SSA is also in terms of its good ratio pertinence/storage of information. It also enables to simplify algorithms, which is also important for increasing the reliability of the compiler. In this context, our aim has been, in particular, to develop exact or heuristic solutions to *combinatorial* problems that arise in compilation for VLIW and DSP processors, and to integrate these methods into industrial compilers for DSP processors (mainly ST100, ST200, Strong ARM). Such combinatorial problems can be found in register allocation, opcode selection, code placement, when removing the SSA multiplexer functions (known as ϕ functions). These optimizations are usually done in the last phases of the compiler, using an assembly-level intermediate representation. As mentioned in Sections 2.3 and 2.4, we made a lot of progress in this area in our past collaborations with STMicroelectronics (see also previous activity reports). Through the Sceptre and Mediacom projects, we first revisited, in the light of SSA, some code optimizations in an aggressive context, to develop better strategies, without eliminating too quickly solutions that may have been considered as too expensive in the past. Then we exploited the new concepts introduced in the aggressive context to design better algorithms in a JIT context, focusing on the speed of algorithms and their memory footprint, without compromising too much on the quality of the generated code.

Our recent research directions were more focused on programmable accelerators, such as GPU and multicores, but still considering *static* compilation and without forgetting the link between high-level (in general at source-code level) and low-level (i.e., at assembly-code level) optimizations. They concerned program analysis (of both sequential and parallel specifications), program optimizations (for memory hierarchies, parallelism, streaming, etc.), and also the link with applications, and between compilers and users (programmers). Polyhedral techniques play an important role in these directions, even if control-flow-based techniques remain in the background and may come back at any time in the foreground. This is also the case for high-level synthesis, as exposed in the next section.

3.1.2. Context of High-Level Synthesis and FPGA Platforms

High-level synthesis has become a necessity, mainly because the exponential increase in the number of gates per chip far outstrips the productivity of human designers. Besides, applications that need hardware accelerators usually belong to domains, like telecommunications and game platforms, where fast turn-around and time-to-market minimization are paramount. When Compsys started, we were convinced that our expertise in compilation and automatic parallelization could contribute to the development of the needed tools.

Today, synthesis tools for FPGAs or ASICs come in many shapes. At the lowest level, there are proprietary Boolean, layout, and place-and-route tools, whose input is a VHDL or Verilog specification at the structural or register-transfer level (RTL). Direct use of these tools is difficult, for several reasons:

- A structural description is completely different from an usual algorithmic language description, as it is written in term of interconnected basic operators. One may say that it has a spatial orientation, in place of the familiar temporal orientation of algorithmic languages.
- The basic operators are extracted from a library, which poses problems of selection, similar to the instruction selection problem in ordinary compilation.
- Since there is no accepted standard for VHDL synthesis, each tool has its own idiosyncrasies and reports its results in a different format. This makes it difficult to build portable HLS tools.
- HLS tools have trouble handling loops. This is particularly true for logic synthesis systems, where loops are systematically unrolled (or considered as sequential) before synthesis. An efficient treatment of loops needs the polyhedral model. This is where past results from the automatic parallelization community are useful.
- More generally, a VHDL specification is too low level to allow the designer to perform, easily, higher-level code optimizations, especially on multi-dimensional loops and arrays, which are of paramount importance to exploit parallelism, pipelining, and perform communication and memory optimizations.

Some intermediate tools were proposed that generate VHDL from a specification in restricted C, both in academia (such as SPARK, Gaut, UGH, CloogVHDL), and in industry (such as C2H, CatapultC, Pico-Express, Vivado HLS). All these tools use only the most elementary form of parallelization, equivalent to instruction-level parallelism in ordinary compilers, with some limited form of block pipelining, and communication through FIFOs. Targeting one of these tools for low-level code generation, while we concentrate on exploiting loop parallelism, might be a more fruitful approach than directly generating VHDL. However, it may be that the restrictions they impose preclude efficient use of the underlying hardware. Our first experiments with these HLS tools reveal two important issues. First, they are, of course, limited to certain types of input programs so as to make their design flows successful, even if, over the years, they become more and more mature. But it remains a painful and tricky task for the user to transform the program so that it fits these constraints and to tune it to get good results. Automatic or semi-automatic program transformations can help the user achieve this task. Second, users, even expert users, have only a very limited understanding of what back-end compilers do and why they do not lead to the expected results. An effort must be done to analyze the different design flows of HLS tools, to explain what to expect from them, and how to use them to get a good quality of results. Our first goal was thus to develop high-level techniques that, used in front of existing HLS tools, improve their utilization. This should also give us directions on how to modify them or to design new tools from scratch.

More generally, HLS has to be considered as a more global parallelization process. So far, no HLS tools is capable of generating designs with communicating *parallel* accelerators, even if, in theory, at least for the scheduling part, a tool such as Pico-Express could have such capabilities. The reason is that it is, for example, very hard to automatically design parallel memories and to decide the distribution of array elements in memory banks to get the desired performances with parallel accesses. Also, how to express communicating processes at the language level? How to express constraints, pipeline behavior, communication media, etc.? To better exploit parallelism, a first solution is to extend the source language with parallel constructs, as in all derivations of the Kahn process networks model, including communicating regular processes (CRP). The other solution is a form of automatic parallelization. However, classical methods, which are mostly based on scheduling, need to be revisited, to pay more attention to locality, process streaming, and low-level pipelining, which are of paramount importance in hardware. Besides, classical methods mostly rely on the runtime system to tailor the parallelism degree to the available resources. Obviously, there is no runtime system in hardware. The real challenge is thus to invent new scheduling algorithms that take resource, locality, and pipelining into account, and then to infer the necessary hardware from the schedule. This is probably possible only for programs that fit into the polyhedral model, or in an incrementally-extended model.

Our research activities on polyhedral code analysis and optimizations directly targeted these HLS challenges. But they are not limited to the automatic generation of hardware as can be seen from our different contributions on X10, OpenStream, parametric tiling, etc. The same underlying concepts also arise when optimizing codes for GPUs and multicores. In this context of polyhedral analysis and optimizations, we focused on three aspects:

- developing high-level transformations, especially for loops and memory/communication optimizations, that can be used in front of HLS tools so as to improve their use, as well as for hardware accelerators;
- developing concepts and techniques in a more global view of high-level synthesis and high-level parallel programming, starting from specification languages down to hardware implementation;
- developing more general code analysis so as to extract more information from codes as well as to extend the programs that can be handled.

3.2. Code Analysis, Code Transformations, Code Optimizations

Embedded systems, as we recalled earlier, generated new problems in code analysis and optimization both for optimizing embedded software (compilation) and hardware (HLS). We now give a bit more details on some general challenges for program analysis, optimizations, and transformations, induced by this context, and on our methodology, in particular our development and use of polyhedral optimizations and its extensions.

3.2.1. Processes, Scheduling, Mapping, Communications, etc.

Before mapping an application to an architecture, one has to decide which execution model is targeted and where to intervene in the design flow. Then one has to solve scheduling, placement, and memory management problems. These three aspects should be handled as a whole, but present state of the art dictates that they be treated separately. One of our aims was to develop more comprehensive solutions. The last task is code generation, both for the processing elements and the interfaces processors/accelerators.

There are basically two execution models for embedded systems: one is the classical accelerator model, in which data is deposited in the memory of the accelerator, which then does its job, and returns the results. In the streaming model, computations are done on the fly, as data items flow from an input channel to the output. Here, the data are never stored in (addressable) memory. Other models are special cases, or sometimes compositions of the basic models. For instance, a systolic array follows the streaming model, and sometimes extends it to higher dimensions. Software radio modems follow the streaming model in the large, and the accelerator model in detail. The use of first-in first-out queues (FIFO) in hardware design is an application of the streaming model. Experience shows that designs based on the streaming model are more efficient that those based on memory, for such applications. One of the point to be investigated is whether it is general enough to handle arbitrary (regular) programs. The answer is probably negative. One possible implementation of the streaming model is as a network of communicating processes either as Kahn process networks (FIFO based) or as our more recent model of communicating regular processes (memory based, such as CRP mentioned hereafter). It is an interesting fact that several researchers have investigated the translation from process networks [12] and to process networks [20], [21]. Streaming languages such as StreamIt and OpenStream are also interesting solutions to explore.

Kahn process networks (KPN) were introduced 30 years ago as a notation for representing parallel programs. Such a network is built from processes that communicate via perfect FIFO channels. Because the channel histories are deterministic, one can define a semantics and talk meaningfully about the equivalence of two implementations. As a bonus, the dataflow diagrams used by signal processing specialists can be translated on-the-fly into process networks. The problem with KPNs is that they rely on an asynchronous execution model, while VLIW processors and FPGAs are synchronous or partially synchronous. Thus, there is a need for a tool for synchronizing KPNs. This can be done by computing a schedule that has to satisfy data dependences within each process, a causality condition for each channel (a message cannot be received before it is sent), and real-time constraints. However, there is a difficulty in writing the channel constraints because one has to count messages in order to establish the send/receive correspondence and, in multi-dimensional loop nests, the counting functions may not be affine. The same situation arises for the OpenStream language (see Section 7.2. Recent developments on the theory of polynomials (see Section 7.1) may offer a solution to this problem. One can also define another model, *communicating regular processes* (CRP), in which channels are represented as write-once/read-many arrays. One can then dispense with counting functions and prove that the determinacy property still holds. As an added benefit, a communication system in which the receive operation is not destructive is closer to the expectations of system designers.

The main difficulty with this approach is that ordinary programs are usually not constructed as process networks. One needs automatic or semi-automatic tools for converting sequential programs into process networks. One possibility is to start from array dataflow analysis [15] or variants. Another approach attempts to construct threads, i.e., pieces of sequential code with the smallest possible interactions. In favorable cases, one may even find outermost parallelism, i.e., threads with no interactions whatsoever. Tiling mechanisms can also be used to define atomic processes that can be pipelined as we proposed initially for FPGA [9].

Whatever the chosen solution (FIFO or addressable memory) for communicating between two accelerators or between the host processor and an accelerator, the problems of optimizing communication between processes and of optimizing buffers have to be addressed. Many local memory optimization problems have already been solved theoretically. Some examples are loop fusion and loop alignment for array contraction, techniques for data allocation in scratch-pad memory, or techniques for folding multi-dimensional arrays [11]. Nevertheless, the problem is still largely open. Some questions are: how to schedule a loop sequence (or even a process network) for minimal scratch-pad memory size? How is the problem modified when one introduces unlimited and/or bounded parallelism (same questions for analyzing explicitly-parallel programs)? How does one take into account latency or throughput constraints, bandwidth constraints for input and output channels, memory hierarchies? All loop transformations are useful in this context, in particular loop tiling, and may be applied either as source-to-source transformations (when used in front of HLS or C-level compilers) or to generate directly VHDL or lower-level C-dialects such as OpenCL. One should keep in mind that theory will not be sufficient to solve these problems. Experiments are required to check the relevance of the various models (computation model, memory model, power consumption model) and to select the most important factors according to the architecture. Besides, optimizations do interact: for instance, reducing memory size and increasing parallelism are often antagonistic. Experiments will be needed to find a global compromise between local optimizations. In particular, the design of cost models remain a fundamental challenge.

Finally, there remains the problem of code generation for accelerators. It is a well-known fact that methods for program optimization and parallelization do not generate a new program, but just deliver blueprints for program generation, in the form, e.g., of schedules, placement functions, or new array subscripting functions. A separate code generation phase must be crafted with care, as a too naive implementation may destroy the benefits of high-level optimization. There are two possibilities here as suggested before; one may target another high-level synthesis or compilation tool, or one may target directly VHDL or low-level code. Each approach has its advantages and drawbacks. However, both situations require that the input program respects some strong constraints on the code shape, array accesses, memory accesses, communication protocols, etc. Furthermore, to get the compilers do what the user wants requires a lot of program tuning, i.e., of program rewriting or of program annotations. What can be automated in this rewriting process? Semi-automated?

In other words, we still need to address scheduling, memory, communication, and code generation issues, in the light of the developments of new languages and architectures, pushing the limits of such an automation of program analysis, program optimizations, and code generation.

3.2.2. Beyond Static Control Programs

With the advent of parallelism in supercomputers, the bulk of research in code transformation resulted in (semi-)automatic parallelization, with many techniques (analysis, scheduling, code generation, etc.) based on the description and manipulation of nested loops with polyhedra. Compsys has always taken an active part in the development of these so-called “polyhedral techniques”. Historically, these analysis were (wrongly) understood to be limited to static control programs.

Actually, the polyhedral model is neither a programming language nor an execution model, rather an intermediate representation. As such, it can be generated from imperative sequential languages like C or Fortran, streaming languages like CRP, or equational languages like Alpha. While the structure of the model is the same in all three cases, it may enjoy different properties, e.g., a schedule always exists in the first case, not in the two others. The import of the polyhedral model is that many questions relative to the analysis of a program and the applicability of transformations can be answered precisely and efficiently by applying well-known mathematical results to the model.

For irregular programs, the basic idea is to construct a polyhedral over-approximation, i.e., a program which has more operations, a larger memory footprint, and more dependences than the original. One can then parallelize the approximated program using polyhedral tools, and then return to the original, either by introducing guards, or by insuring that approximations are harmless. This technique is the standard way of dealing with approximated dependences. We already started to study the impact of approximations in our kernel offloading technique, for optimizing remote communications [10]. It is clear however that this extension method based on over-approximation will apply only to mildly non-polyhedral programs. The restriction to arrays as the only data structure is still present. Its advantage is that it will be able to subsume in a coherent framework many disparate tricks: the extraction of SCoPs, induction variable detection, the omission of non-affine subscripts, or the conversion of control dependences into data dependences. The link with the techniques developed in the PIPS compiler (based on array region analysis) is strong and will have to be explored.

Such over-approximations can be found by mean of abstract interpretation, a general framework to develop static analysis on real-life programs. However, they were designed mainly for verification purposes, thus precision was the main issue before scalability. Although many efforts were made in designing specialized analyses (pointers, data structures, arrays), these approaches still suffer from a lack of experimental evidence concerning their applicability for code optimization. Following our experience and work on termination analysis (that connects the work on back-end CFG-like and front-end polyhedral-like optimizations), and our work on range analysis of numerical variables and on the memory footprint on real-world C programs [18], one of our objectives for the future was to bridge the gap between abstract interpretation and compilation, by designing cheaper analyses that scale well, mainly based on compact representations derived from variants of static single assignment (SSA), with a special focus on complex control, and complex data structures (pointers, lists) that still suffer from complexity issues in the area of optimization.

Another possibility is to rely on application specific knowledge to guide compiler decisions, as it is impossible for a compiler alone to fully exploit such pieces of information. A possible approach to better utilize such knowledge is to put the programmers “in the loop”. Expert parallel programmers often have a good idea about coarse-grain parallelism and locality that they want to use for an application. On the other hand, fine-grain parallelism (e.g., ILP, SIMD) is tedious and specific to each underlying architecture, and is best left to the compiler. Furthermore, approximations will have opportunities to be refined using programmer knowledge. The key challenge is to create a programming environment where compiler techniques and programmer knowledge can be combined effectively. One of the difficulties is to design a common language between the compiler and the programmer. The first step towards this objective is to establish inter-disciplinary collaborations with users, and take the time to analyze and optimize their applications together.M

3.3. Mathematical Tools

All compilers have to deal with *sets* and relations. In classical compilers, these sets are finite: the set of statements of a program, the set of its variables, its abstract syntax tree (AST), its control-flow graph (CFG), and many others. It is only in the first phase of compilation, parsing, that one has to deal with infinite objects, regular and context-free languages, and those are represented by finite grammars, and are processed by a symbolic algorithm, `yacc` or one of its clones.

When tackling parallel programs and parallel compilation, it was soon realized that this position was no longer tenable. Since it makes no sense to ask whether a statement can be executed in parallel with itself, one has to consider sets of operations, which may be so large as to forbid an extensive representation, or even be infinite. The same is true for dependence sets, for memory cells, for communication sets, and for many other objects a parallel compiler has to consider. The representation is to be *symbolic*, and all necessary algorithms have to be promoted to symbolic versions.

Such symbolic representations have to be efficient – the formula representing a set has to be much smaller than the set itself – and effective – the operations one needs, union, intersection, emptiness tests and many others – have to be feasible and fast. As an aside, note that progress in algorithm design has blurred the distinction between polynomially-solvable and NP-complete problems, and between decidable and undecidable questions. For instance SAT, SMT, and ILP software tools solve efficiently many NP-complete problems, and the Z3 tool is able to “solve” many instances of the undecidable Hilbert’s 10th problem.

Since the times of Pip and of the Polylib, Compsys has been active in the implementation of basic mathematical tools for program analysis and synthesis. Pip is still developed by Paul Feautrier and Cédric Bastoul, while the Polylib is now taken care of by the Inria Camus project, which introduced Ehrhart polynomials. These tools are still in use world-wide and they also have been reimplemented many times with (sometimes slight) improvements, e.g., as part of the Parma Polylib, of Sven Verdoolaege’s Isl and Barvinok libraries, or of the Jollylib of Reservoir Labs. Other groups also made a lot of efforts towards the democratization of the use of polyhedral techniques, in particular the Alchemy Inria project, with Cloog and the development of Graphite in GCC, and Sadayappan’s group in the USA, with the development of U. Bondhugula’s Pluto prototype compiler. The same effort is made through the PPCG prototype compiler (for GPU) and Pencil (directives-based language on top of PPCG).

After 2009, Compsys continued to focus on the introduction of concepts and techniques to extend the polytope model, with a shift toward tools that may prepare the future. For instance, PoCo and C2fsm are able to parse general programs, not just SCoPs (static control programs), while the efficient handling of Boolean affine formulas [13] is a prerequisite for the construction of non-convex approximations. Euclidean lattices provide an efficient abstraction for the representation of spatial phenomena, and the construction of *critical lattices* as embedded in the tool Cl@k is a first step towards memory optimization in stream languages and may be useful in other situations. Our work on Chuba introduced a new element-wise array reuse analysis and the possibility of handling approximations. Our work on the analysis of while loops is both an extension of the polytope model itself (i.e., beyond SCoPs) and of its applications, here links with program termination and worst-case execution time (WCET) tools.

A recent example of this extension idea is the proposal by Paul Feautrier to use polynomials for program analysis and optimization [14]. The associated tools are based on Handelman and Schweighofer theorems, the polynomial analogue of Farkas lemma. While this is definitely work in progress, with many unsolved questions, it has the potential of greatly enlarging the set of tractable programs.

As a last remark, observe that a common motif of these developments is the transformation of finite algorithms into symbolic algorithms, able to solve very large or even infinite instances. For instance, PIP is a symbolic extension of the Simplex; our work on memory allocation is a symbolic extension of the familiar register allocation problem; loop scheduling extends DAG scheduling. Many other algorithms await their symbolic transformation: a case in point is resource-constrained scheduling.

4. Application Domains

4.1. Compilers for Embedded Computing Systems

The previous sections described our main activities in terms of research directions, but also placed Compsys within the embedded computing systems domain, especially in Europe. We will therefore not come back here to the importance, for industry, of compilation and embedded computing systems design.

In terms of application domain, the embedded computing systems we considered are mostly used for multimedia: phones, TV sets, game platforms, etc. But, more than the final applications developed as programs, our main application has always been the computer itself: how the system is organized (architecture) and designed, how it is programmed (software), how programs are mapped to it (compilation and high-level synthesis).

The industry that can be impacted by our research is thus all the companies that develop embedded processors, hardware accelerators (programmable or not), embedded systems, and those (the same plus other) that need software tools to map applications to these platforms, i.e., that need to use or even develop programming languages, program optimization techniques, compilers, operating systems. Compsys did not focus on all these critical parts, but our activities were connected to them.

4.2. Users of HPC Platforms and Scientific Computing

The convergence between embedded computing systems and high-performance computing (HPC) technologies offers new computing platforms and tools for the users of scientific computing (e.g., people working in numerical analysis, in simulation, modeling, etc.). The proliferation of “cheap” hardware accelerators and multicores makes the “small HPC” (as opposed to computing centers with more powerful computers, grid computing, and exascale computing) accessible to a larger number of users, even though it is still difficult to exploit, due to the complexity of parallel programming, code tuning, interaction with compilers, which result from the multiple levels of parallelism and of memories in the recent architectures. The link between compiler and code optimization research (as in Compsys) and such users are still to be reinforced, both to guarantee the relevance of compiler research efforts with respect to application needs, and to help users better interact with compiler choices and understand performance issues.

The support of Labex MILYON (through its thematic quarters, such as the thematic quarter on compilation we organized in 2013 [3], or the 2016 thematic quarter on high-performance computing, with a dedicated inter-disciplinary spring school between numerical simulation and polyhedral compilation, see hereafter) and the activities of the LyonCalcul initiative [4] are means to get closer to users of scientific computing, even if it is too early to know if Compsys will indeed be directly helpful to them.

5. Highlights of the Year

5.1. Highlights of the Year

Scientific Results and Dissemination

Despite the approaching end of Compsys, we continued the objectives we fixed for Compsys III, i.e., pushing static compilation beyond its present limits, both in terms of techniques and applications. Our most important efforts in 2016 were to extend static analysis from sequential codes to parallel specifications and languages, to develop polynomial techniques, and to increase inter-disciplinary collaborations and dissemination towards HPC users and their applications. The most important results in 2016 are the following:

- **Publications** Well recognized in the polyhedral community, we got three papers at IMPACT'16, the central event of this community, one paper at the main compiler conference (CC'16), and a last one in the field of FPGA, which remains an important target for polyhedral optimizations. See Sections 7.1 to 7.7 for more details.
- **Interdisciplinary spring school** With colleagues from HPC numerical simulation, we organized a very successful inter-disciplinary event in May 2016, to bridge the gap between polyhedral compilation and HPC users. See details in Section 10.1.
- **Move towards HPC users** In addition to the spring school we organized, we increased our activity towards HPC users and their applications through the supervision of the internship of J. Versaci (quantum physics), the reviewing of T. Gasc's PhD thesis (fluid dynamics), and the regular contacts with the LMGC lab (mechanics).
- **PhD theses** The end of Compsys coincided also with the end of two PhD theses, the PhD thesis of Guillaume Iooss [16] and the PhD thesis of Alexandre Isoard [17], see Section 10.2.2.
- **Final evaluation** The team was evaluated in March 2016, this was also its final evaluation.

Final Evaluation and End of Compsys

Compsys has been created in 2002 as an Inria team, then in 2004 as an Inria project-team, and evaluated by Inria first in 2007, then in 2012. It was evaluated again in March 2016, which was its final evaluation because an Inria project-team is limited to 12 years. The construction of a new project was planned in early 2015, following the shift in the research directions that started in the second half of Compsys III. A few tentative research directions were:

- Shift the application domain from embedded systems to high performance computing (HPC) but at small scale (desktop HPC: FPGA, GPU, multicores). In fact, the two ecosystems are nowadays slowly converging.
- A stronger attention to real HPC users and real HPC applications may lead to better programming models ("putting the programmer in the loop").
- Design new models of programs. The polynomial model is but an example.
- Explore the synergy between parallel programming and program verification and certification; in particular, import approximation methods from one field to the other. Abstract interpretation is a case in point.

[3]Thematic quarter on compilation: http://labexcompilation.ens-lyon.fr/
[4]Lyon Calcul federation: http://lyoncalcul.univ-lyon1.fr

However, while its field of expertise, compilation for parallel and heterogeneous systems, is still of crucial importance, the unexpected departure in Sep. 2015 of two of its staff members made this future impossible. We nevertheless continued in 2016, in particular to present our activities in this last evaluation, until the three last members had to split in three different cities (Lyon, Paris, Rennes). We report here some of the comments made by the external reviewers that, we think, summarize well some aspects of our efforts, successes, and difficulties during 15 years:

- *Compsys established and matured the polyhedral optimization approach, which is the state of the art for locality and parallelism optimization in optimizing compilers. The project has had world-wide impact.*
- *We strongly recommend that the members of the team are accommodated in Camus, Cairn, Parkas, or another complementary Inria team, irrespective of the geographical location. Otherwise, Inria will lose one of its peaks of research excellence in Computer Science.*
- *This team is a prime example where Inria requirements on teams are damaging science and collaboration.*
- *This team has produced many impactful results and is considered as the Polyhedral center of excellence. It is globally recognized for its research in both front-end (polyhedral optimizations) and back-end (graph optimizations) compiler optimization techniques integrating elegant foundational theory with real implementation on various architectures (multi-core, FPGAs, DSP, GPU etc.).*
- *In back-end optimizations, the team had developed the state-of-the-art SSA and decoupled register allocation techniques that are important to achieving peak performance.*
- *They have internationally visible and impactful research in compilers, technology transfer to companies through collaborations and through start-ups. They raised the global awareness of polyhedral analysis through creation of workshops, summer schools etc., essentially reviving interest in the topic about a decade ago, and finally educating next-generation of researchers in this area, who are now contributing to both academic and industrial research landscape in France and beyond.*
- *The start-up company (XtremLogic on HLS) is an excellent concrete evidence of technology transfer from the team. [...] In the future, a more careful analysis of the trade-off between technology transfer and academic research is necessary for small project teams so that a promising research direction does not get jeopardized in Inria.*
- *The Compsys team has truly achieved research excellence in compilation techniques. Unfortunately, the future of the team remains uncertain due to administrative policies. Inria should enable the team to continue with their research strengths in polyhedral analysis and graph-theory based SSA-type optimizations.*

6. New Software and Platforms

6.1. Lattifold

Lattice-based Memory Folding
KEYWORDS: Polyhedral compilation - Euclidean Lattices
FUNCTIONAL DESCRIPTION

Implements advanced lattice-based memory folding techniques. The idea is to reduce memory footprint of multidimensional arrays by reducing the size of each dimension. Given a relation denoting conflicting array cells, it produces a new mapping based on affine functions bounded by moduli. The moduli induces memory reuse and bound memory accesses to a tighter area, allowing to reduce the array size without loss of correctness. Status: proof of concept, see related paper [2].

- Partner: ENS Lyon
- Contact: Alexandre Isoard

6.2. PolyOrdo

Polynomial Scheduler
FUNCTIONAL DESCRIPTION

Computes a polynomial schedule for a sequential polyhedral program having no affine schedule, in lieu of multidimensional schedules. Uses algorithms for finding positive polynomials in semi-algebraic sets. Status: proof of concept software, see related paper [14].

- Contact: Paul Feautrier

6.3. OpenOrdo

OpenStream scheduler
FUNCTIONAL DESCRIPTION

Finds polynomial schedules for the streaming language OpenStream. Main use: detecting deadlocks. The scheduler has been extended to bound the size of stream buffers, either directly or as a side-effect of constructing bounded delay schedules. An effort for bounding the number of in-flight tasks is under way.

Status: proof of concept, see related paper [1].

- Contact: Paul Feautrier

6.4. ppcg-paramtiling

Parametric Tiling Extension for PPCG
KEYWORDS: Source-to-source compiler - Polyhedral compilation
FUNCTIONAL DESCRIPTION

PPCG is a source-to-source compiler, based on polyhedral techniques, targeting GPU architectures. It involves automatic parallelization and tiling using polyhedral techniques. This version replaces the static tiling of PPCG by a fully parametric tiling and code generator. It allows to choose tile sizes at run time when the memory size is known. It also provides a symbolic expression of memory usage depending on the problem size and the tile sizes.

Status: proof of concept, unfinished, see Alexandre Isoard's thesis [17].

- Partner: ENS Lyon
- Contact: Alexandre Isoard

7. New Results

7.1. Handling Polynomials for Program Analysis and Transformation

Participant: Paul Feautrier.

As is well known in natural language processing, the first step in translating a text from one language to another is to understand it. The situation is the same for formal languages. A language processor has to "understand" a program before translating or optimizing or verifying it. Such understanding takes the form of a *model*, usually a mathematical representation whose natural operations mimic the behavior of its program. The polyhedral model is such a representation. However, the set of programs it can represent is too restricted, and the hunt for more powerful models has been under way since the millennium.

An obvious ideas is to replace affine formulas by polynomials, and hence polyhedra by semi-algebraic sets. Polynomials are ubiquitous in HPC and embedded system programming. For instance, the so-called "linearizations" (replacing a multi-dimensional object by a one-dimensional one) generate polynomial access functions. These polynomials then reappear in dependence testing, scheduling, and invariant construction. It may also happen that polynomials are absent from the source program, but are created either by an enabling analysis, as for OpenStream (see Section 7.2), or are imposed by complexity consideration. Lastly, polynomials may be native to the underlying algorithm, as when distances are computed by the usual Euclidean formula. What is needed here is a replacement for the familiar emptiness tests and for Farkas lemma (deciding whether an affine form is positive inside a polyhedron). Recent mathematical results by Handelman and Schweighofer on the *Positivstellensatz* allow one to devise algorithms that are able to solve these problems. The difference is that one gets only sufficient conditions, and that complexity is much higher than in the affine cases.

A paper presenting applications of these ideas to three use cases – dependence testing, scheduling, and transitive closure approximation – was presented at (IMPACT'15) [14]. A tool to manipulate polynomials, polynomial constraints and objective functions, needed for the derivation of polynomial schedules, complements this work (see Section 6.2). It implements Farkas lemma and its generalization with Handelman & Schweighofer formulations, and is in constant development, including improvements on the objective functions, in particular to make schedule selection more stable, independently on the degree of the polynomial schedule.

7.2. Static Analysis of OpenStream Programs

Participants: Albert Cohen [Inria Parkas team], Alain Darte, Paul Feautrier.

In the context of the ManycoreLabs project, we started to study the applicability of polyhedral techniques to the parallel language OpenStream [19]. When applicable, polyhedral techniques are indeed invaluable for compile-time debugging and for generating efficient code well suited to a target architecture. OpenStream is a two-level language in which a control program directs the initialization of parallel task instances that communicate through *streams*, with possibly multiple writers and readers. It has a fairly complex semantics in its most general setting, but we restricted ourselves to the case where the control program is sequential, which is representative of the majority of the OpenStream applications.

In contrast to the language X10, which we studied in previous years, this restriction offers deterministic concurrency by construction, but deadlocks are still possible. We showed that, if the control program is polyhedral, one may statically compute, for each task instance, the read and write indices to each of its streams, and thus reason statically about the dependences among task instances (the only scheduling constraints in this polyhedral subset). If the control program has nested loops, communications use one-dimensional channels in a form of linearization, and these indices may be polynomials of arbitrary degree, thus requiring to extend to polynomials the standard polyhedral techniques for dependence analysis, scheduling, and deadlock detection. Modern SMT allow to solve polynomial problems, albeit with no guarantee of success; the approach previously developed by P. Feautrier [14], and recalled in Section 7.1, offers an alternative solution.

The usual way of disproving deadlocks is by exhibiting a schedule for the program operations, a well-known problem for polyhedral programs where dependences can be described by affine constraints. In the case of OpenStream, we established two important results related to deadlocks: 1) a characterization of deadlocks in terms of dependence paths, which implies that streams can be safely bounded as soon as a schedule exists with such sizes, 2) the proof that deadlock detection is undecidable, even for polyhedral OpenStream. Details of this work have been published at the international workshop IMPACT'16 [1].

Some further developments are in progress for scheduling OpenStream programs using polynomial techniques (with a corresponding prototype scheduling tool, specific to OpenStream, see Section 6.3). In particular, we made some progress for parsing a simplified version of OpenStream, exhibiting the relevant structure, and on the properties and construction of schedules with bounded streams and bounded delays, and on the analysis of the "foot bath", i.e., the pool of tasks that are created (already requiring some resources) but not activated yet (because they need to wait for the termination of other tasks due to dataflow semantics). This work should have interesting connections with the way runtime systems of tasks are managed.

7.3. Liveness Analysis in Explicitly-Parallel Programs

Participants: Alain Darte, Alexandre Isoard, Tomofumi Yuki.

In the light of the parallel specifications encountered in our other work – kernel offloading with pipelined communications [10], automatic parallelization, analysis of X10 [22], [23] and of OpenStream (see Section 7.2), intra-array reuse (see Section 7.4) – we revisited scalar and array element-wise liveness analysis for programs with parallel specifications. In earlier work on memory allocation/contraction (register allocation or intra- and inter-array reuse in the polyhedral model), a notion of "time" or a total order among the iteration points was used to compute the liveness of values. In general, the execution of parallel programs is not a total order, and hence the notion of time is not applicable.

We first revised how conflicts are computed by using ideas from liveness analysis for register allocation, studying the structure of the corresponding conflict/interference graphs. Instead of considering the conflict between two pairs of live ranges, we only consider the conflict between a live range and a write. This simplifies the formulation from having four instances involved in the test down to three, and also improves the precision of the analysis in the general case. Then we extended the liveness analysis to work with partial orders so that it can be applied to many different parallel languages/specifications with different forms of parallelism. An important result is that the complement of the conflict graph with partial orders is directly connected to memory reuse, even in presence of races. However, programs with conditionals do not even have a partial order, and our next step will be to handle such cases with more accuracy. Details of this work have been published at the international workshop IMPACT'16 [3].

Some new developments are in progress to explore even further the properties of such liveness analysis and the construction of conflict sets, in the general case (with connections with the concept of trace monoid) or for some common situations such as series-parallel graphs, appearing in languages such as X10 or OpenMP.

7.4. Extended Lattice-Based Memory Allocation

Participants: Alain Darte, Alexandre Isoard, Tomofumi Yuki.

We extended lattice-based memory allocation [11], an earlier work on memory (array) reuse analysis. The main motivation is to handle in a better way the more general forms of specifications we see today, e.g., with loop tiling, pipelining, and other forms of parallelism available in explicitly parallel languages. Our extension has two complementary aspects. We showed how to handle more general specifications where conflicting constraints (those that describe the array indices that cannot share the same location) are specified as a (non-convex) union of polyhedra. Unlike convex specifications, this also requires to be able to choose suitable directions (or basis) of array reuse. For that, we extended two dual approaches, previously proposed for a fixed basis, into optimization schemes to select suitable basis. Our final approach relies on a combination of the two, also revealing their links with, on one hand, the construction of multi-dimensional schedules for parallelism and tiling (but with a fundamental difference that we identify) and, on the other hand, the construction of universal reuse vectors (UOV), which was only used so far in a specific context, for schedule-independent mapping.

This algorithmic work, connected to our previous work on parametric tiling [10] and the liveness analysis results of Section 7.3, is complemented by a set of prototype scripting tools, see Section 6.1. Details of this work have been published at the 2016 International Conference on Compiler Construction [2].

7.5. Stencil Accelerators

Participants: Steven Derrien [University of Rennes 1, Inria/CAIRN], Sanjay Rajopadhye [Colorado State University], Tomofumi Yuki.

Stencil computations have been known to be an important class of programs for scientific calculations. Recently, various architectures (mostly targeting FPGAs) for stencils are being proposed as hardware accelerators with high throughput and/or high energy efficiency. There are many different challenges for such design: How to maximize compute-I/O ratio? How to partition the problem so that the data fits on the on-chip memory? How to efficiently pipeline? How to control the area usage? We seek to address these challenges by combining techniques from compilers and high-level synthesis tools.

One project in collaboration with the CAIRN team and Colorado State University targets stencils with regular dependence patterns. Although many architectures have been proposed for this type of stencils, most of them use a large number of small processing elements (PE) to achieve high throughput. We are exploring an alternative design that aims for a single, large, deeply-pipelined PE. The hypothesis is that the pipelined parallelism is more area-efficient compared to replicating small PEs. We have published a work-in-progress paper on this topic at IMPACT'16 [4].

7.6. Efficient Mapping of Irregular Memory Accesses on FPGA

Participants: Xinyu Niu [Imperial College London], Tomofumi Yuki.

In a collaboration with Imperial College, we looked at efficiently implementing dynamic dependences on FPGAs. The collaboration is in the context of the EURECA project [5] where the dynamic reconfigurability of modern FPGAs is used to efficiently handle dynamic access patterns. We worked on analyzing data dependent array accesses to identify regularities within irregular memory accesses to reduce the cost of a dynamic memory reconfiguration module.

One part of this work has been published at the 2016 International Conference on Field Programmable Logic and Applications [5].

7.7. PolyApps

Participant: Tomofumi Yuki.

Loop transformation frameworks using the polyhedral model have gained increased attention since the rise of the multi-core era. We now have several research tools that have demonstrated their power on important kernels found in scientific computations. However, there remains a large gap between the typical kernels used to evaluate these tools and the actual applications used by the scientists.

PolyApps is an effort to collect applications from other domains of science to better establish the link between the compiler tools and “real” applications. The applications are modified to bypass some of the front-end issues of research tools, while keeping the ability to produce the original output. The goal is to assess how the state-of-the-art automatic parallelizers perform on full applications, and to identify new opportunities that only arise in larger pieces of code.

We showed that, with a few enhancements, the current tools will be able to reach and/or exceed the performance of existing parallelizations of the applications. One of the most critical element missing in current tools is the ability to modify the memory mappings.

8. Bilateral Contracts and Grants with Industry

8.1. Bilateral Contracts with Industry

Since the team was going to be stopped, Compsys did not try to establish any long-term contract with industry.

8.2. Bilateral Grants with Industry

Same situation.

[5] http://www.doc.ic.ac.uk/~nx210/2015/09/01/eureca.html

9. Partnerships and Cooperations

9.1. Regional Initiatives

Compsys followed or participated to the activities of LyonCalcul (http://lyoncalcul.univ-lyon1.fr/), a network to federate activities on high-performance computing in Lyon. In this context, and with the support of the Labex MILYON (http://milyon.universite-lyon.fr/), Compsys had organized in 2013 a thematic quarter on compilation (http://labexcompilation.ens-lyon.fr). A second thematic quarter on high performance computing (HPC) was organized in 2016, initiated by Violaine Louvet (Institute Camille Jordan), with the participation of the LIP teams Aric, Avalon, Compsys, and Roma. Among other events, it included a CNRS inter-disciplinary spring school (https://mathsinfohpc.sciencesconf.org) co-organized by Compsys, connecting mathematics (HPC numerical analysis) and computer science (polyhedral optimizations for HPC) that can be seen as a follow-up of the first polyhedral school organized by Compsys in 2013. See details in Section 10.1.

Alain Darte, Alexandre Isoard, and Tomofumi Yuki had also some exchanges with Violaine Louvet and Thierry Dumont on tiling code optimizations, advising (in an informal way) some of their students during their internships, for implementations on multicore machines and GPUs.

9.2. National Initiatives

9.2.1. French Compiler Community

In 2010, Laure Gonnord and Fabrice Rastello created the french community of compilation, which had no organized venue in the past. All groups with activities related to compilation were contacted and the first “compilation day” was organized in Lyon. This effort has been quickly a success: the community (http://compilfr.ens-lyon.fr/) is now well identified and 3-days workshops now occur at least once a year (the 11th event has been organized in Sep. 2016). The community is animated by Laure Gonnord and Fabrice Rastello since 2010, and now also by Florian Brandner (ex-Compsys too). Alain Darte and Tomofumi Yuki participated to the 11th edition.

Recognized as a sub-group of the CNRS GDR GPL (Software Engineering and Programming), the community is also in charge, since 2014, of organizing one day of the research school “Ecole des jeunes chercheurs en Algorithmique et Programmation” (EJCP). Tomofumi Yuki, in this context, gave a half-day lecture at the 2016 edition (http://ejcp2016.univ-lille1.fr/), following his 2015 course.

9.2.2. Collaboration with Parkas group, in Paris

Alain Darte and Paul Feautrier have regular meetings with Albert Cohen, from the Parkas team at ENS Paris. The current discussions are mostly related to the analysis and compilation of the OpenStream language developed by Parkas, a research topic that started though the ManycoreLabs project (see previous reports). The results of Sections 7.2 and 7.1 are related to this collaboration. Now that Compsys has been stopped, Paul Feautrier is affiliated to Parkas, in addition to his emeritus position at ENS-Lyon.

9.2.3. Collaboration with Cairn group, in Rennes

Tomofumi Yuki continues to work with the Cairn group through regular meetings and occasional visits. The topic of the collaboration is in applying compiler techniques for hardware design using high-level synthesis. Section 7.5 presents the results through this collaboration.

9.2.4. Collaboration with Camus group, in Strasbourg

Paul Feautrier and Tomofumi Yuki have an ongoing cooperation with Alain Ketterlin and Eric Violard (Camus group, Strasbourg). The main result has been the determination of the *happens before* relation of clocked X10, a prerequisite for the detection of races in clocked programs. The resulting formula has been proved correct using the Coq proof assistant. Publishing formal proofs is known to be difficult, but we will give it a try soon.

9.3. European Initiatives

9.3.1. FP7 & H2020 Projects

After the participation to a (rejected) H2020 proposal in 2015, Compsys did not try any effort in this direction as the team was going to be stopped.

9.3.2. Collaborations in European Programs, Except FP7 & H2020

Same situation.

9.3.3. Collaborations with Major European Organizations

Compsys members participate to the European Network of Excellence on High Performance and Embedded Architecture and Compilation (HiPEAC, http://www.hipeac.net/), either as members or affiliate members. The International Workshop on Polyhedral Compilation Techniques (IMPACT, see Section 9.4.2), co-created by Christophe Alias in 2011, is now an annual event of the HIPEAC conference, as an official workshop. The 5th edition, IMPACT'15, was co-chaired by Alain Darte (see http://impact.gforge.inria.fr/impact2015/), while the 6h edition, IMPACT'16, was co-chaired by Tomofumi Yuki (see http://impact.gforge.inria.fr/impact2016/).

9.4. International Initiatives

9.4.1. Collaboration with Colorado State University

Compsys had always kept strong connections with Colorado State University (CSU):

- In July 2016, Guillaume Iooss defended his joint ENS-Lyon/CSU PhD thesis [16]. He was co-advised by both Sanjay Rajopadhye (CSU) and Christophe Alias (with supplementary support by Alain Darte for administrative reason, as he has no HDR yet).
- Tomofumi Yuki, who did his PhD with Sanjay Rajopadhye, then a post-doc in the Cairn team in Rennes, continued his collaboration with these two groups, as the results described in Section 7.5 illustrate. He also participates regularly, over the net, to the reading group "Melange" of S. Rajodapdhye's group, with CSU students. Due to the stop of Compsys, Tomofumi Yuki has now returned to the Cairn team.
- Waruna Ranasinghe, a PhD student from S. Rajopadhye's team, visited Compsys, to work with Tomofumi Yuki, for 2 months (see Section 9.5).

9.4.2. Polyhedral Community

In 2011, as part of the organization of the workshops at CGO'11, Christophe Alias (with Cédric Bastoul) organized IMPACT'11 (international workshop on polyhedral compilation techniques, http://impact2011.inrialpes.fr/). This workshop in Chamonix was the very first international event on this topic, although it was introduced by Paul Feautrier in the late 80s. Alain Darte gave the introductory keynote talk. After this successful edition (more than 60 people), IMPACT continued as a satellite workshop of the HIPEAC conference, in Paris (2012), Berlin (2013), Vienna (2014). Alain Darte was program co-chair and co-organizer of the 2015 edition in Amsterdam, and Tomofumi Yuki of the 2016 edition in Prague.

The creation of IMPACT, now the annual event of the polyhedral community, helped to identify this community and to make it more visible. This effort was complemented by the organization by Alain Darte of the first school on polyhedral code analysis and optimizations (http://labexcompilation.ens-lyon.fr/polyhedral-school/). A second polyhedral school (https://mathsinfohpc.sciencesconf.org), more open, because involving themes and researchers from numerical analysis (users of HPC), was organized in 2016 by Alain Darte (for the compiler side) and Violaine Louvet (for the HPC side). See details in Section 10.1.

Alain Darte also manages two new mailing lists for news (polyhedral-news@listes.ens-lyon.fr) and discussions (polyhedral-discuss@listes.ens-lyon.fr) on polyhedral code analysis and optimizations. Tomofumi Yuki is involved in the development of PolyBench (http://sourceforge.net/projects/polybench), a suite of kernels used for illustrating polyhedral optimizations. He is also developing PolyApps, a set of larger applications to evaluate the gap between kernels and "real" applications, see more details in Section 7.7.

9.5. International Research Visitors

9.5.1. Visits of International Scientists

9.5.1.1. Visiting PhD students

- Emna Hammami (Tunis University, with Yosr Slama) visited Compsys from April to June 2016 to refine her PhD topic with Compsys members. She also participated to the spring school on numerical simulation and polyhedral compilation.
- Waruna Ranasinghe (Colorado State University, with Sanjay Rajopadhye) visited Compsys from end of June to mid August 2016 to work with Tomofumi Yuki on extending cache oblivious techniques to polyhedral programs.

9.5.1.2. Internships

- Julien Versaci, M2 student from Lyon 1 University, from both physics and computer science departments, worked from April to June 2016 in Compsys, to work on the parallelization of a model of quantum physics. Julien was co-supervised by Jean-Philippe Guillet (physicist) and Tomofumi Yuki, the second part of his internship (until mid August) being done affiliated to Annecy physics laboratory (LAPTH). Julien also participated to the spring school on numerical simulation and polyhedral compilation.

9.5.2. Visits to International Teams

No long (more than one month) stay abroad in 2016.

10. Dissemination

10.1. Promoting Scientific Activities

10.1.1. Scientific Events Organisation

10.1.1.1. General Chair, Scientific Chair

Alain Darte is general chair of the steering committee of CPC (International Workshop on Compilers for Parallel Computing), which regroups in Europe, every 18 months, a large community of researchers interested in compilers for HPC. He participated to CPC'16 in Valladolid in July 2016.

10.1.1.2. Member of the Organizing Committees

Tomofumi Yuki was co-organizer of IMPACT'16 (International Workshop on Polyhedral Compilation Techniques, http://impact.gforge.inria.fr/impact2016/) with Michelle Strout (University of Arizona).

10.1.1.3. Spring School on Numerical Simulation and Polyhedral Compilation

Alain Darte (with the help of Tomofumi Yuki for the program) co-organized with Violaine Louvet (Institute Camille Jordan in Lyon, now lead of UMS Gricad in Grenoble) a second polyhedral spring school, May 9-13 2016, targeting both the polyhedral community and HPC users from numerical analysis. This spring school has been labelled (and funded) as a CNRS interdisciplinary spring school (https://mathsinfohpc.sciencesconf.org/), with a total budget of roughly 39 Keuros, including funding from Labex MILYON, CNRS, GDR Calcul, ENS, LIP, and registrations fees, which were kept low to keep the spirit of the first spring school on polyhedral code analysis and optimizations.

This second spring school was motivated by the need for a more global approach for HPC applications, that combines the design of numerical methods with extensive hardware considerations, in interaction with languages and compilers, so as to take into account both the complexity of architectures and the needs of their non-expert users. Research communities in computer science (architecture, compilation) and applied mathematcs (numerical simulation) are not always aware of this need; at least their work do not always spread enough across the other discipline to lead to mutual influence. Automatic code optimizations and tools also require a better evaluation of their applicability. The goal of this research school – or meeting place of two communities – was to make the link between some of the most recent advances on automatic program optimizations (in particular polyhedral techniques and tools) and applied mathematics (schemes for numerical simulation), in relation with application needs. This school was therefore interdisciplinary, with a strong will to bring communities together on the common theme of supercomputing.

We finally opted for a single track instead of parallel sessions, which helped federate the two communities. The school included courses on architectures (M. Haefele, Maison de la simulation), on numerical schemes in connection with stencils (T. Dumont, ICJ), on simulation methods (discontinuous Galerkin) in particular for GPU (P. Helluy, Strasbourg), on polyhedral techniques and tiling (A. Darte, Compsys), on some polyhedral compilers such as Pluto (U. Bondhugula, Bangalore) and PPCG (S. Verdoolaege, ENS), on the roofline model for performance analysis (M. Püschel, ETH Zürich), on stencils and tensors optimizations (Ramanujam, Baton Rouge), on numerical precision (C. Rubio-Gonzalez, UC Davis), plus some additional talks on reproducibility, applications, the ECM model, etc. The school was a success, with 71 participants, roughly half from each community, with 29 coming from abroad (Italy, Algeria, USA, India, Canada, Germany, Croatia, Switzerland, Austria, Belgium), and a majority (37) being PhD students.

The future will tell if our objectives have been reached, i.e., if the two communities will interact more on the long term and rethink their work with an interdisciplinary look, to invent new computing schemes and compilers more suitable for the constraints of today's architectures, in particular their memory hierarchy and locality needs. In Compsys at least, one can already see some moves in this direction, with the interdisciplinary internship of Julien Versaci co-advised by Tomofumi Yuki, the participation of Alain Darte as a referee to the PhD jury of T. Gasc (CEA, Maison de la Simulation, ENS Cachan), a planned seminar by Alain Darte at Maison de la Simulation in early 2017, starting exchanges with the LMGC lab (Montpellier) on their applications, and a planned mini-symposium, following the line of this spring school, at SMAI 2017.

10.1.2. Scientific Events Selection

10.1.2.1. Chair of Conference Program Committees

In addition to the organization, Tomofumi Yuki was program co-chair of IMPACT'16, with Michelle Strout (University of Arizona).

10.1.2.2. Member of the Conference Program Committees

Alain Darte was a member of the program committee of HPCS'16 (International Conference on High Performance Computing & Simulation) and will be member of the program committee of PACT'17 (International Conference on Parallel Architectures and Compilation Techniques).

Paul Feautrier was a member of the program committees of IMPACT'16 and IMPACT'17.

Tomofumi Yuki was a member of the program committees of SC'16, X10 Workshop'16, IMPACT'16, and IMPACT'17.

10.1.2.3. Reviewer

Alain Darte, Paul Feautrier, and Tomofumi Yuki were reviewers for the different program committees to which they participated.

10.1.3. Journal

10.1.3.1. Member of the Editorial Boards

No participation to journal editorial boards in 2016.

10.1.3.2. Reviewer - Reviewing Activities

Alain Darte was a reviewer for the “Software, Practice, and Experience” journal.

Paul Feautrier was a reviewer for the “International Journal of Parallel Programming”.

Tomofumi Yuki was a reviewer for the TACO, TOPLAS, JPDC, and TPDS journals.

10.1.4. Invited Talks

Alain Darte was invited to give a talk on “Liveness Analysis in Explicitly-Parallel Programs” at ScalPerf’16 in Bertinoro (Italy), Sep. 2016.

Paul Feautrier was invited to give a talk (in two parts) “Toward A Polynomial Model with Application to the OpenStream Language” at the second and third LCS (Language, Compilation, Semantics) LIP seminars, in June and November 2016.

10.2. Teaching - Supervision - Juries

10.2.1. Teaching

Master:

- Paul Feautrier was invited to give a talk on “New Architectures, New Compilations Problems”, at the student seminar for the IMAG M2 course, Grenoble, December 5, 2016.

Spring/Summer Schools:

- Alain Darte, as part of the spring school on numerical simulation and polyhedral compilation, gave a half-day course on “Introduction to Automated Polyhedral Code Optimizations and Tiling”, see https://mathsinfohpc.sciencesconf.org.
- Tomofumi Yuki, a part of the École Jeunes Chercheurs en Programmation 2016, gave a half-day course on “Research in Compilers and Introduction to Loop Transformations”, see http://ejcp2016.univ-lille1.fr/.

10.2.2. Supervision

PhD: Guillaume Iooss, “Detection of linear algebra operations in polyhedral programs” [16], joint PhD ENS-Lyon/Colorado State University, started Sep. 2011, defended July 1st, 2016, advisors: Christophe Alias and Alain Darte (ENS-Lyon) / Sanjay Rajopadhye (Colorado State University).

PhD: Alexandre Isoard, “Extending Polyhedral Techniques towards Parallel Specifications and Approximations” [17], ENS-Lyon, started in Sep. 2012, defended July 5th, 2016, advisor: Alain Darte.

Guillaume Iooss is now post-doc in the Parkas team, while Alexandre Isoard is R&D engineer at Xilinx (Dublin, Ireland, then San Jose, Ca).

10.2.3. Juries

Alain Darte was one of the two reviewers of the PhD of Thibault Gasc (CEA DAM DIF, Maison de la Simulation, November 2016), entitled “Modèles de performance pour l’adaptation des méthodes numériques aux architectures multi-cœurs vectorielles. Application aux schémas Lagrange-Projection en hydrodynamique compressible”. He was also member of the juries of the PhD of Alexandre Isoard, as adviser, and of Guillaume Iooss as administrative co-adviser.

10.3. Popularization

The interdisciplinary spring school organized in May 2016 (see Section 10.1) is a form of popularization of compiler technology (in particular polyhedral optimizations) towards HPC users from the numerical simulation community.

11. Bibliography

Publications of the year

International Conferences with Proceedings

[1] A. Cohen, A. Darte, P. Feautrier. *Static Analysis of OpenStream Programs*, in "6th International Workshop on Polyhedral Compilation Techniques (IMPACT'16), held with HIPEAC'16", Prague, Czech Republic, Proceedings of the IMPACT series, Michelle Strout and Tomofumi Yuki, January 2016, https://hal.inria.fr/hal-01251845

[2] A. Darte, A. Isoard, T. Yuki. *Extended Lattice-Based Memory Allocation*, in "25th International Conference on Compiler Construction (CC'16)", Barcelona, Spain, 25th International Conference on Compiler Construction (CC'16), March 2016, https://hal.archives-ouvertes.fr/hal-01272969

[3] A. Darte, A. Isoard, T. Yuki. *Liveness Analysis in Explicitly-Parallel Programs*, in "6th International Workshop on Polyhedral Compilation Techniques (IMPACT'16), held with HIPEAC'16", Prague, Czech Republic, Proceedings of the IMPACT series, Michelle Strout and Tomofumi Yuki, January 2016, https://hal.inria.fr/hal-01251843

[4] G. Deest, N. Estibals, T. Yuki, S. Derrien, S. Rajopadhye. *Towards Scalable and Efficient FPGA Stencil Accelerators*, in "6th International Workshop on Polyhedral Compilation Techniques (IMPACT'16), held with HIPEAC'16", Prague, Czech Republic, Proceedings of the IMPACT series, http://impact.gforge.inria.fr/, January 2016, https://hal.inria.fr/hal-01254778

[5] X. Niu, N. Ng, S. Wang, T. Yuki, N. Yoshida, W. Luk. *EURECA Compilation: Automatic Optimisation of Cycle-Reconfigurable Circuits*, in "26th International Conference on Field Programmable Logic and Applications", Lausanne, Switzerland, Proceedings of the 26th International Conference on Field Programmable Logic and Applications, August 2016 [*DOI :* 10.1109/FPL.2016.7577359], https://hal.archives-ouvertes.fr/hal-01413307

Conferences without Proceedings

[6] G. Deest, N. Estibals, T. Yuki, S. Derrien, S. Rajopadhye. *Towards Scalable and Efficient FPGA Stencil Accelerators*, in "IMPACT'16", Prague, Czech Republic, January 2016, https://hal.inria.fr/hal-01425018

Research Reports

[7] A. Cohen, A. Darte, P. Feautrier. *Static Analysis of OpenStream Programs*, CNRS ; Inria ; ENS Lyon, January 2016, n^{o} RR-8764, 26 p. , Corresponding publication at IMPACT'16 (http://impact.gforge.inria.fr/impact2016), https://hal.inria.fr/hal-01184408

[8] A. Darte, A. Isoard, T. Yuki. *Liveness Analysis in Explicitly-Parallel Programs*, CNRS ; Inria ; ENS Lyon, January 2016, n^{o} RR-8839, 25 p. , Corresponding publication at IMPACT'16 (http://impact.gforge.inria.fr/impact2016), https://hal.inria.fr/hal-01251579

References in notes

[9] C. ALIAS, A. DARTE, A. PLESCO. *Optimizing Remote Accesses for Offloaded Kernels: Application to High-Level Synthesis for FPGA*, in "International Conference on Design, Automation and Test in Europe (DATE'13)", Grenoble, France, March 2013, pp. 575-580

[10] A. DARTE, A. ISOARD. *Exact and Approximated Data-Reuse Optimizations for Tiling with Parametric Sizes*, in "24th International Conference on Compiler Construction (CC'15), part of ETAPS'15", London, United Kingdom, April 2015, https://hal.inria.fr/hal-01099017

[11] A. DARTE, R. SCHREIBER, G. VILLARD. *Lattice-Based Memory Allocation*, in "IEEE Transactions on Computers", October 2005, vol. 54, n^{o} 10, pp. 1242-1257, Special Issue: Tribute to B. Ramakrishna (Bob) Rau

[12] P. FEAUTRIER. *Scalable and Structured Scheduling*, in "International Journal of Parallel Programming", October 2006, vol. 34, n^{o} 5, pp. 459–487

[13] P. FEAUTRIER. *Simplification of Boolean Affine Formulas*, Inria, July 2011, n^{o} RR-7689, http://hal.inria.fr/inria-00609519/PDF/RR-7689.pdf

[14] P. FEAUTRIER. *The Power of Polynomials*, in "5th International Workshop on Polyhedral Compilation Techniques (IMPACT'15)", Amsterdam, Netherlands, A. JIMBOREAN, A. DARTE (editors), January 2015, https://hal.inria.fr/hal-01094787

[15] P. FEAUTRIER. *Dataflow Analysis of Scalar and Array References*, in "International Journal of Parallel Programming", February 1991, vol. 20, n^{o} 1, pp. 23–53

[16] G. IOOSS. *Detection of linear algebra operations in polyhedral programs*, École normale supérieure de Lyon and Colorado State University, 2016

[17] A. ISOARD. *Extending Polyhedral Techniques towards Parallel Specifications and Approximations*, École normale supérieure de Lyon, 2016

[18] H. NAZARÉ, I. MAFFRA, W. SANTOS, L. OLIVEIRA, F. M. Q. PEREIRA, L. GONNORD. *Validation of Memory Accesses Through Symbolic Analyses*, in "ACM International Conference on Object Oriented Programming Systems Languages & Applications (OOPSLA'14)", Portland, Oregon, United States, October 2014, pp. 791-809, https://hal.inria.fr/hal-01006209

[19] A. POP, A. COHEN. *OpenStream: Expressiveness and data-flow compilation of OpenMP streaming programs*, in "ACM Transactions on Architecture and Code Optimization (TACO)", 2013, vol. 9, n^{o} 4, pp. 1-25

[20] A. TURJAN, B. KIENHUIS, E. DEPRETTERE. *Translating Affine Nested-Loop Programs to Process Networks*, in "International Conference on Compilers, Architecture, and Synthesis for Embedded Systems (CASES'04)", New York, NY, USA, ACM, 2004, pp. 220–229

[21] S. VERDOOLAEGE, H. NIKOLOV, N. TODOR, P. STEFANOV. *Improved Derivation of Process Networks*, in "International Workshop on Optimization for DSP and Embedded Systems (ODES'06)", 2006

[22] T. YUKI, P. FEAUTRIER, S. RAJOPADHYE, V. SARASWAT. *Array Dataflow Analysis for Polyhedral X10 Programs*, in "18th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP'13)", Shenzhen, China, ACM, 2013, http://hal.inria.fr/hal-00761537

[23] T. YUKI. *Revisiting Loop Transformations with X10 Clocks*, in "Proceedings of the ACM SIGPLAN Workshop on X10", Portland, OR, United States, June 2015 [*DOI :* 10.1145/2771774.2771778], https://hal.inria.fr/hal-01253630