IN PARTNERSHIP WITH:
CNRS

Université Claude Bernard (Lyon 1)

Ecole normale supérieure de Lyon

Activity Report 2014

Project-Team COMPSYS

Compilation and embedded computing systems

IN COLLABORATION WITH: Laboratoire de l'Informatique du Parallélisme (LIP)

RESEARCH CENTER
Grenoble - Rhône-Alpes

THEME
Architecture, Languages and Compilation

Table of contents

Project-Team COMPSYS

Keywords: Compilation, Combinatorial Optimization, Hardware Accelerators, High-level Synthesis, High Performance Computing

Creation of the Project-Team: 2004 January 01.

1. Members

Research Scientists

Alain Darte [Team leader, CNRS, Senior Researcher, HdR]
Christophe Alias [Inria, Researcher]
Tomofumi Yuki [Inria, Researcher]

Faculty Members

Paul Feautrier [ENS Lyon, Emeritus Professor]
Laure Gonnord [Univ. Lyon I, Associate Professor]

PhD Students

Guillaume Iooss [PhD student ENS Lyon]
Alexandre Isoard [PhD student ENS Lyon]
Maroua Maalej [PhD student, Univ. Lyon since October 2014]

Visiting Scientists

Romain Labolle [ENS Lyon, internship from June 2014 to July 2014]
Shikhar Makkar [National Institute of Technology Kurukshetra, internship from June 2014 to August 2014]
Amir Teshome Wonjiga [ENS Lyon, internship from May 2014 to August 2014]

Administrative Assistants

Evelyne Blesle [Inria, from July 2014]
Laetitia Lecot-Gauthé [Inria, until July 2014]

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.

Compsys develops compilation techniques, more precisely code analysis and code optimization techniques, to help programming or designing "embedded computing systems" or platforms for "small" HPC (High-Performance Computing). The team focuses on both low-level (back-end) optimizations and high-level (front-end, mainly source-to-source) transformations, for specialized processors, programmable hardware accelerators (GPU, multicores), and FPGA platforms (high-level synthesis). Recent activities include a shift towards the analysis of parallel languages, and links with abstract interpretation and program termination. 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 follow the lines presented in the synthesis report provided for the 2012 evaluation [1], 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. Christophe Alias is also very involved in the development of the Zettice start-up. The hiring of Laure Gonnord and Tomofumi Yuki add new forces on the code analysis research aspects and on HPC polyhedral-related topics.

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, and how its research directions were modified for Compsys II. Section 2.4 briefly presents the main achievements of Compsys II and of the first years of Compsys III, referring to the annual reports from 2008 to 2013 for details. Finally, Section 6.1 highlights the main novelties of the past year, i.e., 2014.

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 never) 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, now 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 is motivated by the following observations: a) the embedded system market is expanding, among many factors, one can quote pervasive digitalization, low-cost products, appliances, etc.; b) research on software for embedded systems is 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, and component reuse.

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). 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

[1] See http://www.ens-lyon.fr/LIP/COMPSYS/wordpress/wp-content/uploads/2013/09/ficheSynthese.pdf

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 is 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 is needed. The main expertise in Compsys for this aspect is in the *parallelization* and optimization of *regular computations*. Hence, we target applications with a large potential parallelism, but we attempt to integrate our solutions into the big picture of CAD environments.

More generally, the aims of Compsys are to develop new compilation and optimization techniques for the field of embedded computing system design. This field is large, and Compsys does not intend to cover it in its entirety. As previously mentioned, we are mostly interested in the automatic design of accelerators, for example designing a VLSI or FPGA circuit for a digital filter, and in the development of new back-end compilation strategies for embedded processors. We study 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 develop are both at source level and at assembly level. A specificity of Compsys is to mix a solid theoretical basis for all code optimizations we introduce with algorithmic/software developments. Within Inria, our project is 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.

Most french researchers working on high-performance computing (automatic parallelization, languages, operating systems, networks) moved to grid computing at the end of the 90s. 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, Fabrice Rastello, and finally Christophe Alias joined the group. Then, Tanguy Risset left Compsys to become a professor at INSA Lyon, and Antoine Fraboulet and Anne Mignotte moved to other fields of research. As for Laure Gonnord, after a post-doc in Compsys, she obtained an assistant professor position in Lille but remained external collaborator of the team for the period 2009-2013 and finally obtained an assistant professor position in Lyon. Then Fabrice Rastello left while Tomofumi Yuki was hired as Inria researcher.

All present and past members of Compsys have 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. He first spent a year in Compsys working on array contraction, where he started to develop his tool Bee, then a year at Ohio State University with Prof. P. Sadayappan on memory optimizations. He finally joined Compsys as an Inria researcher. Laure Gonnord did a PhD on invariant generation and program analysis and became interested on application 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, sept. 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 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 and Antoine Fraboulet left the project to follow research directions closer to their host laboratory at Insa-Lyon. The main achievements of Compsys I, for this period, 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.

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.

2.4. Quick view of Compsys II Achievements and directions for Compsys III

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, 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.

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.

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) reduces the forces on back-end code optimizations, and in particular dynamic compilation, but increases the forces on program analysis. In this context, Compsys III will continue 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 is 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 supports this new direction. A summary of 2013 activities is given in the 2013 annual report, while new results for 2014 are provided in this document, from Section 6.1 (highlights) to Section 6.10.

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 concentrates 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 extension 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 studies 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 so far was to juggle with both aspects, one more on graph theory with SSA-type optimizations, the other with 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 has a tradition of building free software tools for linear programming and optimization in general, and will continue it, as needed for our current 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, generate 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 [2] – 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

[2] *Aggressive compilation* consists in allowing more time to implement more complete and costly solutions: the compiled program is loaded in permanent memory (ROM, flash, etc.) and its compilation time is less relevant than the 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. Hence, the application is 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. For compatibility and compactness, the source languages are CIL or Java bytecode. 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.

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 [3] (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 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 research directions are currently more focused on programmable accelerators, such as GPU and multi-cores, 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 concern 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.

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

- 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 is 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, we want to consider HLS 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, see later). 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 target 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 will focus 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 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 will be to find 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, see for example CRP below). It is an interesting fact that several researchers have investigated translation from process networks [22] and to process networks [27], [28]. Streaming languages such as StreamIt and OpenStream have also been developed.

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. Recent developments on the theory of polynomials (see Section 6.7) 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 [24] 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 [17].

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 [21]. 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 modern 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.

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 [4]. It is clear however that this method 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 [9], our objective is 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). We will focus on complex control, and complex data structures (pointers, lists) that still suffer from complexity issues in the area of optimisation.

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 usable interface between the compiler and the programmer.

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 a parenthesis, 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 [23] 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 the same approach is the proposal by Paul Feautrier to use polynomials for program analysis and optimization [5]. 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 development 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 places 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 consider are mostly used for multimedia: phones, TV sets, game platforms, etc. But, more than the final applications developed as programs, our main application is 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 do not focus on all these critical parts, but our activities are 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 [4], or the upcoming 2016 thematic quarter on high-performance computing) and the activities of the LyonCalcul initiative [5] 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. New Software and Platforms

5.1. Introduction

This section lists and briefly describes the software developments conducted within Compsys. Most are tools that we extend and maintain over the years. They mainly concern three activities: a) the development of research tools, in general available on demand, linked to polyhedra and loop/array transformations, b) the development of tools linked to the start-up XTREMLOGIC (mostly done outside Compsys but partly inspired by work from Compsys), and c) the development of algorithms in the context of our collaborations with STMicroelectronics. These last developments have been stopped right now, since the end of the Mediacom project in 2012. They are described in previous Compsys activity reports: they concerned register allocation, SSA deconstruction, liveness analysis, intermediate representations, etc. They were done within the compilers of STMicroelectronics, not as stand-alone tools, so they are not available as the other tools.

Concerning tools based on a polyhedral representation of nested loops, many of them are now available. They have been developed and maintained over the years by different teams, after the introduction of Paul Feautrier's Pip, a tool for parametric integer linear programming. This "polyhedral model" view of codes is now widely accepted: it was or is used by Cairn, Parkas, and Camus Inria project-teams, PIPS at École des Mines de Paris, Suif from Stanford University, Compaan at Berkeley and Leiden, PiCo from the HP-Labs (continued as PicoExpress by Synfora and now Synopsis), the DTSE methodology at Imec, Sadayappan's group at Ohio State University, Rajopadhye's group at Colorado State's University, etc. In the last 10 years, several compiler groups have shown their interest in polyhedral methods, e.g., the Gcc group, IBM, and Reservoir Labs, a company that develops a compiler fully based on the polyhedral model and on the techniques that we (the french community) introduced for loop and array transformations. Polyhedra are also used in test and certification projects (Verimag, Lande, Vertecs). Now that these techniques are well-established and disseminated in and by other groups, we prefer to focus on the development of new techniques and tools, which are described here. Some of these tools can be used through a web interface on the Compsys tool demonstrator web page http://compsys-tools.ens-lyon.fr/.

Recent activity concerns the development, by Christophe Alias, of HLS compiler parts to be transferred to the XTREMLOGIC start-up (Zettice project) (see Section 7.2). An important effort of applied research and software development [12] has been achieved, resulting in the Dcc (DPN C Compiler) tool, outlined in Section 5.5. Also, optimization developments (scalability, memory leaks, parallelization, etc) were performed on the PoCo compiler framework (see Section 5.6) and the Bee tool (see Section 5.7).

Also, several successive developments have been made related to termination tools. Our first implementation, RanK (see Section 5.9), was extended by other tools such as SToP (see Section 5.12) and, more recently Termite, (see Section 5.13).

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

5.2. Pip

Participants: Cédric Bastoul [professor, Strasbourg University and Inria/CAMUS], Paul Feautrier.

Paul Feautrier is the main developer of Pip (Parametric Integer Programming) since its inception in 1988. Basically, Pip is an "all integer" implementation of the Simplex, augmented for solving integer programming problems (the Gomory cuts method), which also accepts parameters in the non-homogeneous term. Pip is freely available under the GPL at http://www.piplib.org. It is widely used in the automatic parallelization community for testing dependences, scheduling, several kind of optimizations, code generation, and others. Beside being used in several parallelizing compilers, Pip has found applications in some unconnected domains, as for instance in the search for optimal polynomial approximations of elementary functions (see the Inria project Aric, previously known as Arénaire).

5.3. Cl@k

Participants: Fabrice Baray [Mentor Graphics, Former post-doc in Compsys], Alain Darte.

Cl@k (Critical LAttice Kernel) is a stand-alone optimization tool which computes or approximates the critical lattice for a given 0-symmetric polytope. (An admissible lattice is a lattice whose intersection with the polytope is reduced to 0; a critical lattice is an admissible lattice with minimal determinant). This tool is useful for the automatic derivation of array mappings that enable memory reuse, based on the notions of admissible lattice and of modular allocation (linear mapping plus modulo operations). It has been developed in 2005-2006 by Fabrice Baray, former post-doc Inria under Alain Darte's supervision.

Its application to array contraction has been implemented by Christophe Alias in a tool called Bee (see Section 5.7). More information is available at http://compsys-tools.ens-lyon.fr/clak/. The Cl@k tool is unfortunately outdated today (it is hard, if not impossible, to recompile it) and would need to be re-implemented. An extension of its underlying theory is also in progress.

5.4. Syntol

Participant: Paul Feautrier.

Syntol is a modular process network scheduler. The source language is C augmented with specific constructs for representing communicating regular process (CRP) systems. The present version features a syntax analyzer, a semantic analyzer to identify DO loops in C code, a dependence computer, a modular scheduler, and interfaces for CLooG (loop generator developed by C. Bastoul) and Cl@k (see Sections 5.3 and 5.7). The dependence computer now handles casts, records (`structures`), and the modulo operator in subscripts and conditional expressions. The latest developments are, firstly, a new code generator, and secondly, several experimental tools for the construction of bounded parallelism programs.

- The new code generator, based on the ideas of Boulet and Feautrier [20], generates a counter automaton that can be presented as a C program, as a rudimentary VHDL program at the RTL level, as an automaton in the Aspic input format, or as a drawing specification for the DOT tool.
- Hardware synthesis can only be applied to bounded parallelism programs. Our present aim is to construct threads with the objective of minimizing communications and simplifying synchronization. The distribution of operations among threads is specified using a placement function, which is found using techniques of linear algebra and combinatorial optimization.

5.5. Dcc

Participants: Christophe Alias, Alexandru Plesco [XtremLogic].

Dcc (DPN C Compiler) compiles a C program annotated with pragmas to a data-aware process network (DPN), a regular process network close to a circuit description that makes explicit the I/O transfers and the synchronizations. Dcc features throughput optimization, communication vectorization, and automatic parallelization.

Dcc is registered at the APP ("agence de protection des programmes") and has been transferred to the XTREMLOGIC start-up under an Inria licence. It uses a patented technology [12] and serves as a *front-end* for the HLS suite of the XTREMLOGIC start-up. Dcc has been implemented by Christophe Alias, using the PoCo compiler infrastructure (Section 5.6) and the Bee tool (Section 5.7). It represents more than 3000 lines of C++ code.

5.6. PoCo

Participant: Christophe Alias.

PoCo is a polyhedral compilation framework providing many features to prototype program analysis and optimizations in the polyhedral model:

- C parsing and extraction of the polyhedral representation.
- Symbolic layer on the state-of-art polyhedral libraries Polylib (set operations on polyhedra) and Piplib (parameterized ILP, see Section 5.2).
- Dependence analysis (PRDG, array dataflow analysis), array region analysis, array liveness analysis.
- C and VHDL code generation.

PoCo is registered at the APP ("agence de protection des programmes") and has been transferred to the XTREMLOGIC start-up under an Inria licence. The Bee, Chuba, and RanK tools presented thereafter make an extensive use of PoCo abstractions. PoCo has been developed by Christophe Alias. It represents more than 19000 lines of C++ code.

5.7. Bee

Participants: Christophe Alias, Alain Darte.

Bee is a source-to-source optimizer that resizes and reallocates optimally the arrays used by a program under scheduling constraints. Bee applies a fine-grain lifetime analysis for arrays. Then, the mathematical optimization of the Cl@k tool (Section 5.3) finds the array allocation (expressed as an affine lattice). Finally, Bee derives the actual array allocation and generates the C code accordingly. Bee was – to our knowledge – the first complete (i.e., with an element-wise lifetime analyzer integrated with an allocator) array contraction tool. Bee allows to allocate and to size the channels in process networks, providing a global affine schedule. This feature is fundamental in HLS (see Section 3.1.2) and more generally in automatic parallelization where communication buffers must be allocated and sized. An online demonstrator is available at http://compsys-tools.ens-lyon.fr/bee/index.html.

Bee is registered at the APP ("agence de protection des programmes") and has been transferred to the XTREMLOGIC start-up under an Inria licence. It is also used as an external tool by the compiler Gecos developed in the Cairn team at Irisa. Bee has been implemented by Christophe Alias, using the PoCo compiler infrastructure (see Section 5.6). It represents more than 2400 lines of C++ code.

5.8. Chuba

Participants: Christophe Alias, Alain Darte, Alexandru Plesco [Compsys/Zettice].

Chuba is a source-level optimizer that improves a C program in the context of the high-level synthesis (HLS) of hardware. Chuba is an implementation of the work described in the PhD thesis of Alexandru Plesco. The optimized program specifies a system of multiple communicating accelerators, which optimizes the data transfers with the external DDR memory. The program is divided into blocks of computations obtained thanks to tiling techniques, and, in each block, data are fetched by block to reduce the penalty due to line changes in the DDR accesses. Four accelerators achieve data transfers in a macro-pipeline fashion so that data transfers and computations (performed by a fifth accelerator) are overlapped.

So far, the back-end of Chuba is specific to the HLS tool C2H but the analysis is quite general and adapting Chuba to other HLS tools should be possible. Besides, it is interesting to mention that the program analysis and optimizations implemented in Chuba address a problem that is also very relevant in the context of GPGPUs. The underlying theory and corresponding experiments are described in [17].

Chuba has been implemented by Christophe Alias, using the PoCo compiler infrastructure (see Section 5.6). It represents more than 900 lines of C++ code.

5.9. RanK

Participants: Christophe Alias, Alain Darte, Paul Feautrier, Laure Gonnord [Compsys].

RanK is a software tool that can prove the termination of a program (in some cases) by computing a *ranking function*, i.e., a mapping from the operations of the program to a well-founded set that *decreases* as the computation advances. In case of success, RanK can also provide an upper bound of the worst-case time complexity of the program as a symbolic affine expression involving the input variables of the program (parameters), when it exists. In case of failure, RanK tries to prove the non-termination of the program and then to exhibit a counter-example input. This last feature is of great help for program understanding and debugging. The theory underlying RanK was presented at SAS'10 [15].

The input of RanK is an integer automaton, computed by C2fsm (see Section 5.11), representing the control structure of the program to be analyzed. RanK uses the Aspic tool (see Section 5.10), developed by Laure Gonnord during her PhD thesis, to compute automaton invariants. RanK has been used to discover successfully the worst-case time complexity of many benchmarks kernels of the community (see the WTC benchmark suite at http://compsys-tools.ens-lyon.fr/wtc/index.html). It uses the libraries Piplib (Section 5.2) and Polylib.

RanK has been implemented by Christophe Alias, using the compiler infrastructure PoCo (Section 5.6). It represents more than 3000 lines of C++. The tool has been presented at the CSTVA'13 workshop [16]. An online demonstrator is available at http://compsys-tools.ens-lyon.fr/rank.

5.10. Aspic

Participant: Laure Gonnord.

Aspic is an invariant generator for general counter automata. Used with C2fsm (see Section 5.11), it can be used to derivate invariants for numerical C programs, and also to prove safety. It is also part of the WTC toolsuite (see http://compsys-tools.ens-lyon.fr/wtc/index.html), a set of examples to demonstrate the capability of the RanK tool (see Section 5.9) for evaluating worse-case time complexity (number of transitions when executing an automaton).

Aspic implements the theoretical results of Laure Gonnord's PhD thesis on acceleration techniques and has been maintained since 2007.

5.11. C2fsm

Participant: Paul Feautrier.

C2fsm is a general tool that converts an arbitrary C program into a counter automaton. This tool reuses the parser and pre-processor of Syntol (see Section 5.4), which has been extended to handle `while` and `do while` loops, `goto`, `break`, and `continue` statements. C2fsm reuses also part of the code generator of Syntol and has several output formats, including FAST (the input format of Aspic, see Section 5.10), a rudimentary VHDL generator, and a DOT generator which draws the output automaton. In contrast to the FAST format, an ad hoc format, FLOW, uses a relational representation and retains non-affine constructs. C2fsm is also able to do elementary transformations on the automaton, such as eliminating useless states, transitions and variables, simplifying guards, or selecting cut-points, i.e., program points on loops that can be used by RanK (see Section 5.9) to prove program termination.

5.12. SToP

Participants: Christophe Alias, Guillaume Andrieu [University of Lille], Laure Gonnord [Compsys].

SToP (Scalable Termination of Programs) is the implementation of the modular termination technique presented at the TAPAS'12 workshop [18]. It takes as input a large irregular C program and conservatively checks its termination. To do so, SToP generates a set of small programs whose termination implies the termination of the whole input program. Then, the termination of each small program is checked thanks to RanK (see Section 5.9). In case of success, SToP infers a ranking (schedule) for the whole program. This schedule can be used in a subsequent analysis to optimize the program.

SToP represents more than 2000 lines of C++. The first results are available at http://compsys-tools.ens-lyon.fr/stop.

5.13. Termite

Participants: Laure Gonnord, Gabriel Radanne [ENS Rennes], David Monniaux [CNRS/VERIMAG].

Termite (Termination of C programs) is the implementation of our new algorithm "Counter-example based generation of ranking functions" (see Section 6.4). Based on LLVM and Pagai (a tool that generates invariants), the tool automatically generates a ranking function for each *head of loop*. Its implementation is under consolidation, it will be publicly available soon.

Termite represents 3000 lignes of OCaml.

5.14. Simplifiers

Participant: Paul Feautrier.

The aim of the `simple` library is to simplify Boolean formulas on affine inequalities. It works by detecting redundant inequalities in the representation of the subject formula as an ordered binary decision diagram (OBDD), see details in [23]. It uses PIP (see Section 5.2) for testing the feasibility – or unfeasibility – of a conjunction of affine inequalities.

The library is written in Java and is presented as a collection of class files. For experimentation, several front-ends have been written. They differ mainly in their input syntax, among which are a C like syntax, the Mathematica and SMTLib syntaxes, and an ad hoc Quast (quasi-affine syntax tree) syntax. See the first results at http://compsys-tools.ens-lyon.fr/stop.

6. New Results

6.1. Highlights of the Year

For 2014, from the point of view of organization, funding, collaborations, the main points to highlight are:

- Christophe Alias and Alexandru Plesco have co-founded the XTREMLOGIC start-up in January 2014 (see Section 7.2), following the incubation of Zettice. XTREMLOGIC recently won the "concours région rhône-alpes" grant in November 2014 (40k).
- Tomofumi Yuki was hired as an Inria researcher and became a permanent member of Compsys.
- The 1988 "Array Expansion" seminal paper of Paul Feautrier has been selected for the 25th Anniversary Volume of the ACM International Conference on Supercomputing (ICS) together with 34 other papers selected from the 1800 papers published from 1987 to 2011. A short "reminescence" paper [13] was written for the occasion.
- The team was evaluated in Nov. 2014 by the HCERES (new name of AERES), as part of the LIP lab evaluation. The report has not been received yet.

From a scientific point of view, the shift, in Compsys III, towards the analysis of parallel programs and the extensions of the polyhedral model, both in terms of techniques and applications, is continuing, see the section "New Results", in particular:

- The design (by Christophe Alias and Alexandru Plesco) of a HLS compiler technology (see Section 6.2), patented by Inria [12] and transferred to XTREMLOGIC under an Inria licence (see Section 5.5).
- Two new static analyses: a more precise array bound check analysis [9] (see Section 6.3) and a more scalable termination algorithm for C programs (see Section 6.4).
- A novel equivalence-checking algorithm [7] modulo associativity/commutativity, which is a first step towards semantic program transformations (see Section 6.5).
- A groundbreaking introduction of polyhedral techniques for the analysis of parallel programs, in particular X10 (see [29] and [6]) and OpenStream (see Section 6.6).
- A seminal paper [5] introducing polynomial techniques in program analysis and compilation (see Section 6.7).
- Innovative contributions on parametric tiling [8], [3], [4] as extensions of the polyhedral model (see Sections 6.8 and 6.9).

6.2. Data-Aware Process Networks

Participants: Christophe Alias, Alexandru Plesco [XTREMLOGIC start-up].

Process networks are execution models expressing naturally the parallelism of a computation. They are a natural intermediate representation for high-level synthesis tools, where the front-end extracts the parallelism and produces a process network and the back-end compiles the process network to the target architecture.

In that context, we have defined a new model of process network that fits HLS-specific constraints, the data-aware process network (DPN). Our model makes explicit the communications with the central memory and the parallel access to channels, and is close enough to the hardware constraints to be translated directly to a circuit. We show how to compile an imperative program to a DPN, so as to optimize both the I/O and the parallelism, while using the polyhedral model.

DPNs are used as the intermediate representation for the HLS compiler suite of the XTREMLOGIC start-up. They are generated from C programs by the Dcc compiler (see Section 5.5). The apparatus underlying the DPN synchronizations has been patented by Inria [12].

6.3. Preventing from Out-of-Bound Memory Accesses

Participants: Laure Gonnord, Fernando Pereira [Univ. Mineas Gerais, Brasil].

The C programming language does not prevent out-of-bounds memory accesses. There exist several techniques to secure C programs; however, these methods tend to slow down these programs substantially, because they populate the binary code with runtime checks. To deal with this problem, we designed and tested two static analyses (symbolic region and range analysis), which we combine to remove the majority of these guards.

In addition to the analyses themselves, we brought two other contributions:

- First, we described live-range splitting strategies that improve the efficiency and the precision of our analyses.
- Secondly, we showed how to deal with integer overflows, a phenomenon that can compromise the correctness of static algorithms validating memory accesses.

We validated our claims by incorporating our findings into AddressSanitizer (see https://code.google.com/p/address-sanitizer/). We generated SPEC CINT 2006 code that is 17% faster and 9% more energy efficient than the code originally produced by this tool. Furthermore, our approach is 50% more effective than Pentagons, a state-of-the-art analysis to sanitize memory accesses. This work was published at the OOPSLA 2014 conference [9].

6.4. Scaling Termination Proofs

Participants: Laure Gonnord, Gabriel Radanne [ENS Rennes], David Monniaux [CNRS/VERIMAG], Fernando Pereira [Univ. Mineas Gerais, Brasil], Raphael Rodrigues [Univ. Mineas Gerais, Brasil].

In [15], we presented a new algorithm adapted from scheduling techniques to synthesize (multi-dimensional) affine functions from general flowcharts programs. But, as for other methods, our algorithm tried to solve linear constraints on each control point and each transition, which can lead to quasi-intractable linear programming instances. In contrast to these approaches, we proposed a new algorithm based on the following observations:

- Searching for ranking functions for loop headers is sufficient to prove termination.
- Furthermore, there exist loops such that there is a linear lexicographic ranking function that decreases along each path inside the loop, from one loop iteration to the next, but such that there is no lexicographic linear ranking function that decreases at each step along these paths. For these reasons, it is tempting to treat each path inside a loop as a single transition.

Unfortunately the number of paths may be exponential in the size of the program, thus the constraint system may become very large, even though it features fewer variables. To face this theoretical complexity, even though the number of paths may be large, we argue that, in practice, few of them actually matter in the constraint system (we formalize this concept by giving a characterization as geometric extremal points). Our algorithm therefore builds the constraint system lazily, taking paths into account *on demand*.

In 2014, we consolidated this approach with a work on complexity issues (inspired by [19]) and a new implementation: Termite (see Section 5.13). A corresponding paper is currently under submission for PLDI.

With Fernando Pereira's group in Brazil, we also studied the relevance of fast and simple solutions to compute approximations of the number of iterations of loops (*loop trip count*) of imperative real-world programs. The context of this work is the use of these approximations in compiler optimizations: most of the time, the optimizations yield greater benefits for large trip counts, and are either innocuous or detrimental for small ones. In our paper published at WST'14 [10], we have shown that, most of the time, there is no need to use computationally-expensive state-of-the-art methods to compute (an approximation of) it. We support our position with an actual case study. We show that a fast predictor can be used to speedup the JavaScript JIT compiler of Firefox - one of the most well-engineered runtime environments in use today.

6.5. Equivalence-Checking of Programs with Reductions

Participants: Guillaume Iooss, Christophe Alias, Sanjay Rajopadhye [Colorado State University, USA].

Program equivalence is a well-known problem with a wide range of applications, such as algorithm recognition, program verification, and program optimization. This problem is also known to be undecidable if the class of programs is rich enough, in which case semi-algorithms are commonly used.

We focused on programs represented as systems of affine recurrence equations (SARE), defined over parametric polyhedral domains, a well-known formalism for the *polyhedral model*. SAREs include, as a proper subset, the class of affine control loop programs. Several semi-algorithms for program equivalence were already proposed for this class. Some take into account algebraic properties such as associativity and commutativity. To the best of our knowledge, none of them manage reductions, i.e., accumulations of a *parametric* number of sub-expressions using an associative and commutative operator. Our main contribution has been a new semi-algorithm to manage reductions. In particular, we outlined the ties between this problem and the perfect matching problem in a parametric bipartite graph.

This work was published at the SAS 2014 conference [7].

6.6. Analysis and Transformation of Parallel Programs

Participants: Albert Cohen [Inria/PARKAS], Alain Darte, Paul Feautrier, Abdoulaye Gamatie [CNRS/LIRMM], Laure Gonnord, Alain Ketterlin [Inria/CAMUS], Sanjay Rajopadhye [Colorado State University], Vijay Saraswat [IBM Research], Eric Violard [Inria/CAMUS], Tomofumi Yuki.

While, historically, Compsys has applied polyhedral analysis to sequential programs, it was recently realized that it also applies to parallel programs, with the aim of checking their correctness or improving their performance. The prospect of having to program exascale architectures, with their millions of cores, has led to the development of new programming languages, whose objective is to increase the programmer productivity. Compsys has applied polyhedral techniques to synchronous languages (see [25], [26] and previous activity reports), to IBM's high-productivity language X10, and, in the context of the ManycoreLabs project, to a streaming language, OpenStream, developed by Albert Cohen's group.

X10 is based on the creation of independent *activities* (light-weight threads), which can synchronize either by a generalization of the fork/join scheme, or with *clocks*, an improved version of the familiar barriers. X10 is deadlock-free by construction but it is the programmer responsibility to insure determinism by a proper use of synchronizations. Non-determinism bugs may have a very low occurrence probability thus be very difficult to detect by testing, hence the interest for detecting races at compile time. In collaboration with CSU (S. Rajopadhye, T. Yuki) and IBM (V. Saraswat), we extended array dataflow analysis to polyhedral clock-free X10 programs [29]. We have been working on clocked programs too: race detection becomes undecidable [30], but realistic problems may still be solved by heuristics.

As a side-effect of this work, we have shown in cooperation with Eric Violard and Alain Ketterlin (Inria Team Camus, Strasbourg) that clocks can be removed and replaced by `async/finish` constructs without modifying the program semantics [6]. While this transformation incurs a large overhead for general programs, in the polyhedral case the overhead is negligible, thus improving the program performance.

In contrast to X10, OpenStream is deterministic by construction, but may have deadlocks. A 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, communications use one-dimensional channels and, in a form of linearization, give rise to polynomial dependences for polyhedral OpenStream codes. In a ManycoreLabs project deliverable (see Section 7.1), we have formalized the problem and proved that deadlock detection is undecidable in general.

6.7. Handling Polynomials for Program Analysis and Transformation

Participant: Paul Feautrier.

As shown in the previous section, many problems in parallel programs analysis and verification can be reduced to proving or disproving properties of polynomials in the variables of the program. For instance, 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. This is also the case in OpenStream where nested loops act on one-dimensional streams. 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 – will be presented at the 5th International Workshop on Polyhedral Compilation Techniques (IMPACT'15) [5] in Amsterdam in January 2015.

6.8. Parametric Loop Tiling with Constant Aspect Ratio

Participants: Guillaume Iooss, Christophe Alias, Sanjay Rajopadhye [Colorado State University, USA].

Parametric tiling is a well-known transformation which is widely used to improve locality, parallelism, and granularity (see also the next section for more details). However, parametric tiling is also a non-linear transformation and this prevents polyhedral analysis or further polyhedral transformation after parametric tiling. It is therefore generally applied during the code generation phase.

To address this issue, we presented a method to remain in a polyhedral representation, in a special case of parametric tiling where all the dimensions are tiled and all tile sizes are constant multiples of a single tile size parameter. We call this *Constant Aspect Ratio Tiling*. We showed how to mathematically transform a polyhedron and an affine function into their tiled counterpart, which are the two main operations needed in such a transformation.

The approach is now implemented, and has been tested successfully on several kernels commonly used in the community (matrix multiply, jacobi 1D, jacobi 2D). A corresponding paper was published at the IMPACT 2014 workshop [8].

6.9. Exact and Approximated Data-Reuse Optimizations for Tiling with Parametric Sizes

Participants: Alain Darte, Alexandre Isoard.

Loop tiling is a loop transformation widely used to improve spatial and temporal data locality, to increase computation granularity, and to enable blocking algorithms, which are particularly useful when offloading kernels on computing units with smaller memories. When caches are not available or used, data transfers and local storage must be software-managed, and some useless remote communications can be avoided by exploiting data reuse between tiles. An important parameter of tiling is the sizes of the tiles, which impact the size of the required local memory. However, for most analyses involving several tiles, which is the case for inter-tile data reuse, the tile sizes induce non-linear constraints, unless they are numerical constants. This complicates or prevents a parametric analysis with polyhedral optimization techniques.

We showed that, when tiles are executed in sequence along tile axes, the parametric (with respect to tile sizes) analysis for inter-tile data reuse is nevertheless possible, i.e., one can determine, at compile-time and in a parametric fashion, the copy-in and copy-out data sets for all tiles, with inter-tile reuse, as well as sizes for the induced local memories. When approximations of transfers are performed, the situation is much more complex, and involves a careful analysis to guarantee correctness when data are both read and written. We provide the mathematical foundations to make such approximations possible, thanks to the introduction of the concept of *pointwise functions*. Combined with hierarchical tiling, this result opens perspectives for the automatic generation of blocking algorithms, guided by parametric cost models, where blocks can be pipelined and/or can contain parallelism. Previous work on FPGAs and GPUs already showed the interest and feasibility of such automation with tiling, but in a non-parametric fashion. Our method is currently implemented with the `iscc` calculator of `ISL`, a library for the manipulation of integer sets defined with Presburger arithmetic, a complete implementation within the PPCG compiler is in progress.

We believe that our approximation technique can be used for other applications linked to the extension of the polyhedral model as it turns out to be fairly powerful. Our future work will be to derive efficient approximation techniques, either because the program cannot be fully analyzable, or because approximations can speed-up or simplify the results of the analysis without losing much in terms of memory transfers and/or memory sizes.

A preliminary version of this work has been presented at the IMPACT'14 workshop [3]. A revised version has been accepted for publication at the International Conference on Compiler Construction (CC'15) [4].

6.10. Studying Optimal Spilling in the Light of SSA

Participants: Florian Brandner [ENSTA ParisTech], Quentin Colombet [Apple, Cupertino], Alain Darte.

Recent developments in register allocation, mostly linked to static single assignment (SSA) form, have shown the benefits of decoupling the problem in two phases: a first spilling phase places load and store instructions so that the register pressure at all program points is small enough, a second assignment and coalescing phase maps the variables to physical registers and reduces the number of move instructions among registers. At the end of Quentin Colombet's PhD thesis, we focused on the first phase, for which many open questions remained: in particular, we studied the notion of optimal spilling (what can be expressed?) and the impact of SSA form (does it help?). To identify the important features for optimal spilling on load-store architectures,

we developed a new integer linear programming formulation, more accurate and expressive than previous approaches. Among other features, we can express SSA ϕ-functions, memory-to-memory copies, and the fact that a value can be stored simultaneously in a register and in memory. Based on this formulation, we presented a thorough analysis of the results obtained for the SPECINT 2000 and EEMBC 1.1 benchmarks, from which we drawed the following conclusions: a) rematerialization is extremely important, b) SSA complicates the formulation of optimal spilling, especially because of memory coalescing when the code is not in conventional SSA (CSSA), c) micro-architectural features are significant and thus have to be accounted for, which is not the case with standard cost functions, d) significant savings can be obtained in terms of static spill costs, cache miss rates, and dynamic instruction counts, e) however, cost models based only on static spill costs are not always relevant, in particular when spilling is "at the limit": in this situation, bad interactions with register coalescing and post-pass scheduling can be exacerbated and it may be better to spill a bit more. This important observation indicates that more research is needed to explore alternative cost models that reliably guide spilling.

Parts of this work were already published at CASES 2011. The publication at ACM Transactions on Architecture and Code Optimization [1] contains more detailed discussions, more examples illustrating new concepts and existing approaches, and additional experiments covering the observed worst-case behavior, a new post-latency heuristic, and empiric evidence showing why static spill costs are a poor metric. Three configurations were added: Appel and George under SSA, Koes and Goldstein, and the heuristic of Braun and Hack. This work was partly supported by the Mediacom contract with STMicroelectronics (ended in 2013).

7. Bilateral Contracts and Grants with Industry

7.1. ManycoreLabs Project with Kalray

Compsys is part of a bilateral contract with Kalray called ManycoreLabs, funded by "Investissements d'avenir pour le développement de l'économie numérique". The goal of this project is to allow the company Kalray, based on a collaboration with several partners, to become the European leader of the market of many-core chips for embedded systems. Industrial partners of this project include Bull, CAPS Entreprise, Digigram, Thales, Renault. Academic partners are CEA, Inria (Parkas and Compsys), VERIMAG.

Compsys role is to explore analysis and compilation techniques linked to streaming languages, with the Kalray MPPA platform as long-term target. The research on OpenStream described in Section 6.6 corresponds to the work package WP 2.5.3. This study shows the need for extending polyhedral techniques to polynomials, which is one of the motivation of the work described in Section 6.7. Finally, the work on parametric tiling (Section 6.9), first in the context of FPGA, then of GPUs, is a first step towards the automatic generation of blocking algorithms for multicores such as the Kalray MPPA.

7.2. Technological Transfer: XtremLogic Start-Up

The XTREMLOGIC start-up (former Zettice project) was initiated 3 years ago by Alexandru Plesco and Christophe Alias, after the PhD thesis of Alexandru Plesco under the guidance of Christophe Alias, Alain Darte and Tanguy Risset. The goal of XTREMLOGIC is to build on the disruptive technologies emerging from the polyhedral compilation community, and particularly the results obtained in Compsys to provide the HPC market with efficient and communication-optimal circuit blocks (IP) for FPGA.

The compiler technology transferred to XTREMLOGIC (see Section 6.2) is the result of a tight collaboration between Christophe Alias and Alexandru Plesco. XTREMLOGIC is a unique opportunity to spread the polyhedral technology to industry.

XTREMLOGIC won several prizes and grants: "concours émergence OSEO 2013" at Banque Publique d'Investissement, "most promising start-up award" at SAME 2013, "lean Startup award" at Startup Weekend Lyon 2012, "excel&rate award 2012" from Crealys incubation center.

8. Partnerships and Cooperations

8.1. Regional Initiatives

8.1.1. In Relation with the LYONCALCUL Initiative

Compsys follows or participates 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 organized a thematic quarter on compilation from April 2013 to July 2013 (http://labexcompilation.ens-lyon.fr). A new thematic quarter is in preparation for 2016, initiated by Violaine Louvet (Institute Camille Jordan), with the participation of the LIP teams Avalon, Compsys, and Roma. Also, Alain Darte and Alexandre Isoard have regular exchanges with Violaine Louvet and Thierry Dumont on tiling code optimizations.

8.1.2. Streaming Day with CITI Laboratory

Compsys has some common research interests with the Socrate Inria team from the CITI laboratory (Insa-Lyon), in particular streaming languages. In this context, Socrate (Lionel Morel), with the help of Compsys (Alain Darte), organized in April 2014, a thematic day on the "compilation and execution of streaming programs" in Domaine des Hautannes, St Germain au Mont d'Or, with 7 speakers and 32 participants. See the webpage of the event http://streaming.conf.citi-lab.fr.

8.2. National Initiatives

8.2.1. French Compiler Community

Until 2010, the french compiler community had no official national meetings. Laure Gonnord and Fabrice Rastello decided to motivate the different french actors to meet regularly. All groups whose activities are related to compilation were contacted and the first "compilation day" was organized in September 2010 in Lyon. The next sessions, in a form of 3-days workshops, took place in Aussois (winter 2010), Dinard (spring 2011), Saint-Hippolyte (autumn 2011), Rennes (summer 2012), Annecy (spring 2013, organized by Compsys again), Dammarie-les-lys (winter 2013), and Nice (summer 2014). This effort is a success: the community (http://compilfr.ens-lyon.fr) is now well identified and such an event occurs at least once a year. The community is still animated by Laure Gonnord and Fabrice Rastello, and now also by Florian Brandner (ENSTA), and is now recognized as a sub-group of the CNRS GDRs ASR (Architecture, System, Network) and GPL (Software Engineering and Programming). As a subgroup of GPL, the community is (from 2014) now in charge of organizing one day during the Research school "Ecole des jeunes chercheurs en Algorithmique et Programmation".

8.3. European Initiatives

8.3.1. Collaborations with Major European Organizations: HIPEAC network

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 8.4.1.2), 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, is co-organized and co-chaired by Alain Darte (see http://impact.gforge.inria.fr/impact2015/).

8.4. International Initiatives

8.4.1. Inria International Partners

8.4.1.1. Declared Inria International Partners

- Christophe Alias has a regular collaboration with Sanjay Rajopadhye from the Colorado State University (USA), through the advising of the PhD thesis of Guillaume Iooss. This year, this collaboration led to several publications, see Sections 6.8 and 6.5.
- Laure Gonnord has a regular collaboration with Fernando Magno Quintao Pereira from the University of Mineas Gerais (Brazil). This year, this collaboration led to several results, see Sections 6.4 and 6.3. In Jan.-Feb. 2015, Compsys will host Fernando Pereira as an invited professor.

8.4.1.2. Polyhedral Community

In 2011, as part of the organization of the workshops at CGO'11, Christophe Alias (with C. 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 first very 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 is program chair for the next edition, in Amsterdam (2015). 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 of the first (and for the moment unique) school on polyhedral code analysis and optimizations (http://labexcompilation.ens-lyon.fr/polyhedral-school/). 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.

8.5. International Research Visitors

8.5.1. Visits of International Scientists

8.5.1.1. Internships

- Romain Labolle, a L3 ENS-Lyon student, worked, from June 2014 to July 2014, on the adaptation of parametric tiling with inter-tile data reuse to GPUs (reuse for global memory, reuse for shared memory, reuse for registers, i.e., register tiling), supervised by Alain Darte and Alexandre Isoard.
- Shikhar Makkar, a student from the National Institute of Technology Kurukshetra in India, worked, from June 2014 to August 2014, on the mapping of piece-wise affine functions on FPGAs, supervised by Christophe Alias. His internship was funded by the LIP.
- Amir Teshome Wonjiga, a M1 ENS-Lyon student from Ethiopia, worked, from May 2014 to August 2014, on an implementation of an operational semantics of the X10 language, supervised by Paul Feautrier and Laure Gonnord. His internship was funded by Compsys and the LIP.

9. Dissemination

9.1. Promoting Scientific Activities

9.1.1. Scientific events organization

9.1.1.1. general chair, scientific chair

Laure Gonnord is co-chair of the "Compilation french community", with Florian Frandner (ENSTA) and Fabrice Rastello (Inria Corse).

9.1.1.2. member of the organizing committee

Alain Darte was part of the organization (mainly by Lionel Morel, from Insa-Lyon) of the thematic day on the "compilation and execution of streaming programs" in April 2014, see Section 8.1.2.

9.1.2. Scientific events selection

9.1.2.1. responsible of the conference program committee

Alain Darte was program chair of the topic E2 "compilers for Embedded Systems" of the international conference DATE'15 (see http://www.date-conference.com/group/tpc/members#E), with Rodric Rabbah (IBM), and program chair of the international workshop IMPACT'15 (see http://impact.gforge.inria.fr/impact2015/), with Alexandra Jimborean (Uppsala Univ.).

9.1.2.2. member of the conference program committee

Paul Feautrier was a member of the program committees of the conferences and workshops PECCS'15, PARCO'15, IMPACT'14, and IMPACT'15. Laure Gonnord was a member of the program committee of the WST'14 workshop. Tomofumi Yuki was a member of the program committee of the IMPACT'15 workshop. Alain Darte was member of the program committees of the conferences and workshops IPDPS'14, DATE'14, DATE'15, IMPACT'14, and IMPACT'15.

9.1.2.3. reviewer

In 2014, Christophe Alias has reviewed papers for DATE'15 and MEMOCODE'14. Laure Gonnord has reviewed papers for WST'14, TASE'14, VMCAI'15, and MEMOCODE'14. Tomofumi Yuki has reviewed papers for IMPACT'15. Alain Darte has reviewed papers for MICRO-47 (2014), DATE'15, and IMPACT'15.

9.1.3. Journal

9.1.3.1. member of the editorial board

Paul Feautrier is a member of the editorial board of the journals "International Journal of Parallel Programming" and "Parallel Computing".

9.1.3.2. reviewer

In 2014, Christophe Alias has reviewed papers for ACM TACO (Transactions on Architecture and Code Optimization) and Elsevier IPL (Information Processing Letters). Paul Feautrier has reviewed papers for ACM TOPLAS (Transactions on Programming Languages and Systems) and ACM TODAES (Transactions on Design Automation of Electronic Systems). Tomofumi Yuki has reviewed papers for ACM TOPLAS and ACM TACO. Alain Darte has reviewed papers for ACM TACO and IEEE Transactions on Computers.

9.2. Teaching - Supervision - Juries

9.2.1. Teaching

Licence:

- Christophe Alias: Introduction à la compilation, L3, INSA Centre-Val-de-Loire: Spring 2014 (CM+TD=18h).
- Christophe Alias: Architecture des ordinateurs, L2, Université Lyon 1 Claude Bernard: Spring 2014 (TP=15h).
- Christophe Alias: Concours E3A, épreuve informatique MPSI (shepherding/exams): Spring 2014.
- Laure Gonnord: Architecture des ordinateurs, L2, Université Lyon 1 Claude Bernard: Spring 2014 (TD+TP=40h), and Automn 2014 (Course+TD 40h).

Master:

- Christophe Alias, Compilation avancée (CM 10h), M2, Ecole Normale Supérieure de Lyon.

- Christophe Alias, Compilation (CM 24h), M1, Ecole Normale Supérieure de Lyon.
- Laure Gonnord, Introduction aux systèmes et réseaux (CM/TP 52h), M2 Pro, Université Lyon 1.
- Laure Gonnord, Compilation (TD/TP 24h), M1, Université Lyon 1 Claude Bernard.
- Laure Gonnord, Complexité (TD 15h), M1, Université Lyon 1 Claude Bernard.
- Laure Gonnord, Temps réel (TP 24h), M1, Université Lyon 1 Claude Bernard.
- Laure Gonnord also organized two research schools (24h each) for the M1-ENS students. In 2014, the topic was "Programming embedded systems with synchronous languages" (invited speakers: Julien Forget, Abdoulaye Gamatié, Alain Girault, Nicolas Halbwachs). In 2015, the topic will be "Static analyses in the state-of-the-art compilers" (invited speaker: Fernando Pereira).

E-learning:

- Paul Feautrier has recorded a lecture on the polyhedral model, to be part of a course on auto-parallelization sponsored by the IEEE Computer Society (University of Illinois at Urbana-Champaign, Sept. 5-7) and organized by Hironori Kasahara and David Padua.
- Following the 2013 Spring School on Polyhedral Code Analysis and Optimization organized by Alain Darte, the videos of most courses have been collected and made available in 2014 (see http://labexcompilation.ens-lyon.fr/polyhedral-school/videos/). See also Section 8.4.1.2.

9.2.2. Supervision

- PhD in progress: Guillaume Iooss, "Semantic tiling", started in September 2011, joint PhD ENS-Lyon/Colorado State University, advisors: Christophe Alias and Alain Darte (ENS-Lyon) / Sanjay Rajopadhye (Colorado State University).
- PhD in progress: Alexandre Isoard, "Streaming-related code optimizations", started in September 2012, advisor: Alain Darte.
- PhD in progress: Maroua Maleej, "Low cost static analyses for compilers", started in October 2014, advisors: Laure Gonnord and Alain Darte.

9.2.3. Juries

Paul Feautrier was a member of the PhD defense committees of Michael Kruse (Orsay, Sept. 26, 2014), entitled "Lattice QCD Optimization and Polytopic Representations of Distributed Memory", and of Luc Michel (Grenoble, Dec. 18, 2014), entitled "Contributions à la traduction binaire dynamique: support du parallélisme d'instructions et génération de traducteurs optimisés".

Laure Gonnord was a reviewer for the PhD thesis of Chan N'Go (Rennes, Jul. 2014), entitled "Formal Verification of a Synchronous Data-flow Compiler: From Signal to C".

Alain Darte was member of the PhD defense committee of Romain Brillu (Nantes, Nov. 12, 2014), entitled "Efficient Design and Programming of MPSoC (Multiple Processors System on Chip) Architectures".

9.3. Popularization

- Alain Darte was invited to give a long keynote (1h30), as an introduction to polyhedral techniques ("Polyhedral optimizations? Not even scared!" [2]), to the spanish Conference Jornadas Sarteco, the equivalent of the french COMPAS conference, see http://www.jornadassarteco.org/transparencias-de-las-charlas-invitadas/.
- In Sep. 2014, together with ten other specialists in automatic parallelization, Paul Feautrier recorded a lecture on the "polyhedral model" for a video course on automatic parallelization to be published by the IEEE.

- Paul Feautrier's 1988 "Array Expansion" seminal paper has been selected for the 25th Anniversary Volume of the ACM International Conference on Supercomputing, with 34 other papers, among 1800 papers published from 1987 to 2011. A short "reminescence" paper [13] has been written for the occasion. http://dl.acm.org/citation.cfm?doid=2591635.2591641.
- Alain Darte participated to the "Refresh" Inria Rhône-Alpes initiative, for improving the scientific life of the Inria regional center, as well as to the working group on "team management".

10. Bibliography

Publications of the year

Articles in International Peer-Reviewed Journals

[1] Q. COLOMBET, F. BRANDNER, A. DARTE. *Studying Optimal Spilling in the Light of SSA*, in "ACM Transactions on Architecture and Code Optimization", 2014, pp. 25-34, forthcoming, https://hal.inria.fr/hal-01099016

Invited Conferences

[2] A. DARTE. *Polyhedral Optimizations? Not Even Scared!*, in "Jornadas Sarteco", Valladolid, Spain, Arturo González Escribano and Diego R. Llanos Ferraris, September 2014, Keynote speech, https://hal.inria.fr/hal-01099290

International Conferences with Proceedings

[3] A. DARTE, A. ISOARD. *Parametric Tiling with Inter-Tile Data Reuse*, in "4th International Workshop on Polyhedral Compilation Techniques (IMPACT'14)", Vienna, Austria, S. RAJOPADHYE, S. VERDOOLAEGE (editors), January 2014, https://hal.archives-ouvertes.fr/hal-00915831

[4] 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

[5] 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

[6] P. FEAUTRIER, E. VIOLARD, A. KETTERLIN. *Improving X10 Program Performances by Clock Removal*, in "23rd International Conference on Compiler Construction (CC'14), part of ETAPS'14", Grenoble, France, April 2014, https://hal.inria.fr/hal-00924206

[7] G. IOOSS, C. ALIAS, S. RAJOPADHYE. *On Program Equivalence with Reductions*, in "21st International Static Analysis Symposium (SAS'14)", Munich, Germany, September 2014, https://hal.inria.fr/hal-01096110

[8] G. IOOSS, S. RAJOPADHYE, C. ALIAS, Y. ZOU. *CART: Constant Aspect Ratio Tiling*, in "4th International Workshop on Polyhedral Compilation Techniques (IMPACT'14)", Vienna, Austria, S. RAJOPADHYE, S. VERDOOLAEGE (editors), January 2014, https://hal.archives-ouvertes.fr/hal-00915827

[9] H. NAZARÉ, I. MAFFRA, W. SANTOS, L. OLIVEIRA, F. 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

[10] R. E. RODRIGUES, P. ALVES, F. PEREIRA, L. GONNORD. *Real-World Loops are Easy to Predict: A Case Study*, in "Workshop on Software Termination (WST'14)", Vienne, Austria, July 2014, https://hal.inria.fr/hal-01006208

Research Reports

[11] A. DARTE, A. ISOARD. *Exact and Approximated Data-Reuse Optimizations for Tiling with Parametric Sizes*, LIP - ENS Lyon ; CNRS ; Inria ; UCBL, January 2015, n^{o} RR-8671, 28 p. , https://hal.inria.fr/hal-01103460

Patents and standards

[12] C. ALIAS, A. PLESCO. *Procédé de synthèse de circuits, dispositif et programme d'ordinateur associés*, April 2014, n^{o} FR1453308, https://hal.inria.fr/hal-01096129

Other Publications

[13] P. FEAUTRIER. , U. BANERJEE (editor) *Author Retrospective for Array Expansion, Array Shrinking, or There and Back Again*, ACM, 2014, 1 p. , ACM International Conference on Supercomputing (ICS) 25th Anniversary Volume, https://hal.inria.fr/hal-01099746

[14] A. ISOARD. *Data-reuse Optimizations for Pipelined Tiling with Parametric Tile Sizes*, August 2014, 23rd International Conference on Parallel Architectures and Compilation Techniques (PACT'14), https://hal.inria.fr/hal-01111393

References in notes

[15] C. ALIAS, A. DARTE, P. FEAUTRIER, L. GONNORD. *Multi-dimensional Rankings, Program Termination, and Complexity Bounds of Flowchart Programs*, in "17th International Static Analysis Symposium (SAS'10)", Perpignan, France, ACM press, September 2010, pp. 117-133

[16] C. ALIAS, A. DARTE, P. FEAUTRIER, L. GONNORD. *Rank: A Tool to Check Program Termination and Computational Complexity*, in "International Workshop on Constraints in Software Testing Verification and Analysis (CSTVA'13)", Luxembourg, March 2013, 238 p. , http://hal.inria.fr/hal-00801571

[17] 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

[18] G. ANDRIEU, C. ALIAS, L. GONNORD. *SToP: Scalable Termination Analysis of (C) Programs*, in "International Workshop on Tools for Automatic Program Analysis (TAPAS'12)", Deauville, France, September 2012, (tool presentation), http://hal.inria.fr/hal-00760926

[19] A. M. BEN-AMRAM, S. GENAIM. *Ranking Functions for Linear-Constraint Loops*, in "Journal of the ACM", July 2014, vol. 61, n^{o} 4, pp. 26:1–26:55

[20] P. BOULET, P. FEAUTRIER. *Scanning Polyhedra without DO loops*, in "International Conference on Parallel Architecture and Compilation Techniques (PACT'98)", Paris, France, IEEE Computer Society, October 1998, pp. 4-11

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

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

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

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

[25] P. FEAUTRIER, A. GAMATIÉ, L. GONNORD. *Enhancing the Compilation of Synchronous Dataflow Programs with a Combined Numerical-Boolean Abstraction*, in "CSI Journal of Computing", 2012, vol. 1, n° 4, 8:86 p.

[26] A. GAMATIÉ, L. GONNORD. *Static Analysis of Synchronous Programs in Signal for Efficient Design of Multi-Clocked Embedded Systems*, in "International Conference on Languages, Compilers, and Tools for Embedded Systems (LCTES'11)", Chicago, USA, April 2011

[27] 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

[28] 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

[29] 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

[30] T. YUKI, P. FEAUTRIER, S. RAJOPADHYE, V. SARASWAT. *Checking Race Freedom of Clocked X10 Programs*, arXiv, 2013, n° arXiv.1311.4305, http://hal.inria.fr/hal-00907723