IN PARTNERSHIP WITH:
CNRS

Université Claude Bernard (Lyon 1)

Ecole normale supérieure de Lyon

Activity Report 2013

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

Compsys is an EPC (équipe-projet commune), i.e., a research project-team that is common to several institutions: Inria, Ecole normale supérieure de Lyon (ENS-Lyon), CNRS, and Université Claude Bernard of Lyon (UCB-Lyon). It is located at Ecole normale supérieure de Lyon and exists since January 2002 as part of the computer science laboratory (Laboratoire de l'Informatique du Parallélisme, Lip, UMR CNRS ENS-Lyon UCB-Lyon Inria 5668) and as an Inria pre-project. It became a full Inria project in January 2004. It has been evaluated by Inria in Spring 2007 and extended 4 more years. It has been evaluated by AERES in December 2010 and received the mark A+. It has been evaluated positively again by Inria in Spring 2012 and extended 4 more years. Thus, it should end around 2016.

The goal of Compsys is to develop compilation techniques, more precisely code analysis and code optimization techniques, for programming or designing embedded computing systems. So far, Compsys focused on both low-level (back-end) optimizations for embedded processors and high-level (front-end, mainly source-to-source) transformations, in particular for high-level synthesis of hardware accelerators. Recent activities also include a shift towards dynamic compilation, compilation for GPUs and multicores, and the analysis of parallel languages. The main characteristic of Compsys is its focus on combinatorial optimization problems (graph algorithms, linear programming, polyhedral optimizations) coming from code optimization problems (register allocation, memory optimization, scheduling, automatic generation of interfaces, etc.) and the validation of these techniques in the development of compilation tools.

Creation of the Project-Team: 2004 January 01.

1. Members

Research Scientists

Christophe Alias [Inria, Researcher]
Alain Darte [Team leader, CNRS, Senior Researcher, HdR]
Fabrice Rastello [Inria, Researcher, HdR]

Faculty Members

Paul Feautrier [ENS-Lyon, Professor, Emeritus, HdR]
Laure Gonnord [Univ. Lyon I, Associate Professor, since Sep. 2013 (formerly in Lille University)]

External Collaborators

Adrian Muresan [ENS-Lyon, Zettice funding from Jan. 2013 to Mar. 2013]
Alexandru Plesco [Zettice, Inria funding until Apr. 2013]

Engineers

Florent Bouchez [Inria, ManycoreLabs funding, Caisse des Dépôts et Consignations, from May 2013 to Aug. 2013]
Alexandros Lamprineas [Inria, ManycoreLabs funding, Caisse des Dépôts et Consignations, from Sep. 2013]

PhD Students

François Gindraud [ENS-Lyon grant, from Jan. 2013]
Guillaume Iooss [ENS-Lyon grant, from Sep. 2011]
Alexandre Isoard [ENS-Lyon grant, from Sep. 2012]
Diogo Nunes Sampaio [Brazilian CAPES grant, from Oct. 2013]
Duco Van Amstel [Kalray Cifre grant, from Jan. 2013]

Post-Doctoral Fellow

Lukasz Domagala [Inria, ManycoreLabs funding, Caisse des Dépôts et Consignations, from Jan. 2013]

Visiting Scientist
Raphael Ernani Rodrigues [Univ. Lyon I, Internship, from May 2013 to Jul. 2013]

Administrative Assistants
Laetitia Lecot-Gauthé [Inria]
Maria Immaculada Presseguer [Inria]

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.

The objective of Compsys is to adapt and to extend code analysis and code optimization techniques primarily designed in compilers/parallelizers for high performance computing to the special case of *embedded computing systems*. In particular, Compsys works on back-end optimizations for specialized processors and on high-level program transformations, in particular for the compilation towards or the synthesis of hardware accelerators. The main characteristic of Compsys is its focus on combinatorial problems (graph algorithms, linear programming, polyhedra) coming from code optimizations (register allocation, cache and memory optimizations, scheduling, optimizations for power, automatic generation of software/hardware interfaces, etc.) and the validation of techniques developed in 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 prolongated into Compsys III at the very end of 2012. The geographical move in 2013 of Fabrice Rastello to Grenoble was first to expand the activities of Compsys in the context of Giant, a R&D technology center with several industrial and academic actors. In 2014, this geographical move is a departure from Compsys, Fabrice Rastello will now work on his own. The research directions of Compsys III are nevertheless not modified drastically and are in line with the research directions presented in the synthesis report provided for the 2012 evaluation [1]. The shift towards dynamic compilation, underlined in this report, will be pursued by Fabrice Rastello only, while the shift towards the compilation of streaming programming, the analysis and optimizations of parallel languages, with an even stronger focus on polyhedral optimizations are the heart of Compsys III, as well as the development of the Zettice start-up in which Christophe Alias is involved. The hiring of Laure Gonnord also adds new forces on the code analysis research aspects.

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, referring to the annual reports from 2008 to 2012 for details. Finally, Section 2.5 highlights the main novelties of the past year, i.e., 2013.

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

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

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 can be more aggressive (i.e., take more time to optimize) for optimizing some important parts of applications. This opens a new range of optimizations. Another interesting aspect is the introduction of platform-independent intermediate languages, such as Java bytecode, that is compiled dynamically at runtime (aka just-in-time). Extreme lightweight compilation mechanisms that run faster and consume less memory have to be developed. One of the objectives of Compsys was to revisit existing compilation techniques in the context of embedded computing systems, to deconstruct these techniques, to improve them, and to develop new techniques taking constraints of embedded processors 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. Academic projects also exist 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 will target applications with a large potential parallelism, but we will 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 Alchemy (now Parkas), 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.

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 polytope 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 Spring 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.

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 em-

bedded 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).

2.5. Highlights of the Year

For 2013, from the point of view of organization, funding, collaborations, the main points to highlight are:

- The Zettice startup project, initiated by Alexandru Plesco and Christophe Alias, won the *concours OSEO 2013* grant (Banque Publique d'Investissement, 40 Keuros) and the *"most promising start-up award"* at SAME 2013. See more details in Section 7.3.
- Laure Gonnord was hired as assistant professor at ENS-Lyon, she is now a permanent member of Compsys. Fabrice Rastello has left Compsys and will continue his research in Grenoble.
- The collaborations with Colorado State University (S. Rajopadhye) and Ohio State University (Sadayappan) were very successful. New topics of collaboration with the Inria Parkas and Camus teams have started.
- From April 2013 to July 2013, Compsys organized 4 scientific events on compilation, regrouped in a larger and coherent *thematic quarter on compilation* [2], with international audience and visibility. It was mainly funded by the Labex MILYON, see details in Section 9.1.

From a scientific point of view, the shift, in Compsys III, towards the analysis of parallel programs, the extensions of the polyhedral model, both in terms of techniques and applications, and the code optimizations based on trace analysis has been already fruitful, see the section "New Results", in particular:

- Innovative contributions on parametric tiling [8], [5] as extensions of the polyhedral model.
- A groundbreaking introduction of polyhedral techniques for the analysis of parallel programs, in particular X10 [10], [7].
- Several important contributions (e.g., [2]) that demonstrate the interest of mixing trace analysis and static analysis for code (in particular locality) improvements.

3. Research Program

3.1. Generalities

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.

[2] http://labexcompilation.ens-lyon.fr

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 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. Such a combination 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 multi-cores 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 in this transformation chain, restricting to compilation (transforming a program to a program) for embedded processors 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. 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). In all aspects, the frontier between software and hardware is vanishing. For example, in terms of architecture, customized processors (with processor extension as 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. 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.

The next two sections give an overview of the main directions that were explored by Compsys II and partially extended in 2013 in Compsys III: back-end code optimizations for embedded processors (including aggressive and just-in-time compilation) and high-level program analysis and transformations, primarily for high-level synthesis. For Compsys III, the shifts towards dynamic compilation on one hand and more advanced polyhedral techniques for program analysis and optimization on the other hand are not detailed here but appear clearly in the section "New Results". Indeed, the first axis (dynamic com-

pilation and trace analysis) will not be pursued in 2014, due to the departure of Fabrice Rastello, it will thus be described in his activity report for 2014. The second axis (polyhedral extensions and high-level program analysis) will be detailed more deeply in 2014. But, it is already not limited to high-level synthesis as can be seen from the different contributions on X10, OpenStream, parametric tiling, etc.

3.2. Back-End Code Optimizations for Embedded Processors

Compilation is an old activity, in particular back-end code optimizations. We first give some elements that explain why the development of embedded systems makes compilation come back as a research topic. We then detail the code optimizations that we are interested in, both for aggressive and just-in-time compilation.

3.2.1. Embedded Systems and the Revival of Compilation & Code Optimizations

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 rediscovered the interest of 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), or (quasi) general-purpose processors (DSP for multimedia applications). Our cooperation with STMicroelectronics led us to investigate the last solution, as implemented in the ST100 (DSP processor) and the ST200 (VLIW DSP processor) family for example. Compilation and, in particular, back-end code optimizations find a second life in the context of such embedded computing systems.

At the heart of this progress is the concept of *virtualization*, which is the key for more portability, more simplicity, more reliability, and of course more security. This concept, implemented through binary translation, just-in-time compilation, etc., consists in hiding the architecture-dependent features as far as possible during the compilation process. It has been used for quite a long time for servers such as HotSpot, a bit more recently for workstations, and it is quite recent for embedded computing for reasons we now explain.

As previously mentioned, the definition of "embedded systems" is rather imprecise. However, one can at least agree on the following features:

- Even for processors that are programmable (as opposed to hardware accelerators), processors have some architectural specificities, and are very diverse;
- Many processors (but not all of them) have limited resources, in particular in terms of memory;
- For some processors, power consumption is an issue;
- In some cases, aggressive compilation (through cross-compilation) is possible, and even highly desirable for important functions.

This diversity is one of the reason why virtualization, which starts to be more mature, is becoming more and more common in programmable embedded systems, in particular through CIL (a standardization of MSIL). This implies a late compilation of programs, through just-in-time (JIT), including dynamic compilation. Some people even think that dynamic compilation, which can have more information because performed at run-time, can outperform the performances of "ahead-of-time" compilation.

Performing code generation (and some higher-level optimizations) in a late phase is potentially advantageous, as it can exploit architectural specificities and run-time program information such as constants and aliasing, but it is more constrained in terms of time and available resources. Indeed, the processor that performs the late compilation phase is, *a priori*, less powerful (in terms of memory for example) than a processor used for cross-compilation. The challenge is thus to spread the compilation process in time by deferring some optimizations ("deferred compilation") and by propagating some information for those whose computation is expensive ("split compilation"). Classically, a compiler has to deal with different intermediate representations (IR) where high-level information (i.e., more target-independent) co-exist with low-level information. The split compilation has to solve a similar problem where, this time, the compactness of the information representation,

and thus its pertinence, is also an important criterion. Indeed, the 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 also a reason why static single assignment (SSA) is becoming specific to embedded compilation, even if it was first used for workstations. Indeed, SSA is a sparse (i.e., compact) representation of liveness information. In other words, 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.

3.2.2. Aggressive and Just-in-Time Optimizations of Assembly-Level Code

Compilation for embedded processors is difficult because the architecture and the operations are specially tailored to the task at hand, and because the amount of resources is strictly limited. For instance, 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, but also the special form of applications [19] generate many open problems. Our goal is to contribute to their understanding and their solutions.

As previously explained, compilation for embedded processors include both aggressive and just in time (JIT) optimizations. Aggressive compilation consists in allowing more time to implement costly solutions (so, looking for complete, even expensive, studies is mandatory): the compiled program is loaded in permanent memory (ROM, flash, etc.) and its compilation time is not significant; also, for embedded systems, code size and energy consumption usually have a critical impact on the cost and the quality of the final product. Hence, the application is cross-compiled, in other words, compiled on a powerful platform distinct from the target processor. Just-in-time compilation corresponds to compiling applets on demand on the target processor. For compatibility and compactness, the source languages are CIL or Java. The code can be uploaded or sold separately on a flash memory. Compilation is performed at load time and even dynamically during execution. Used heuristics, constrained by time and limited resources, are far from being aggressive. They must be fast but smart enough.

Our aim is, 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 for example in register allocation, in opcode selection, or in code placement for optimization of the instruction cache. Another example is the problem of removing the multiplexer functions (known as ϕ functions) that are inserted when converting into SSA form. These optimizations are usually done in the last phases of the compiler, using an assembly-level intermediate representation. In industrial compilers, they are handled in independent phases using heuristics, in order to limit the compilation time. Our initial goal was to develop a more global understanding of these optimization problems to derive both aggressive heuristics and JIT techniques, the main tool being the SSA representation.

In particular, we investigated the interaction of register allocation, coalescing, and spilling, with the different code representations, such as SSA. One of the challenging features of today's processors is predication [28], which interferes with all optimization phases, as the SSA form does. Many classical algorithms become inefficient for predicated code. This is especially surprising, since, beside giving a better trade-off between the number of conditional branches and the length of the critical path, converting control dependences into data dependences increases the size of basic blocks and hence creates new opportunities for local optimization algorithms. One has to adapt classical algorithms to predicated code [30] and also to study the impact of predicated code on the whole compilation process.

As mentioned in Section 2.3, a lot of progress has already been done in this direction in our past collaborations with STMicroelectronics. In particular, the goal of the Sceptre project was to revisit, in the light of SSA, some code optimizations in an aggressive context, i.e., by looking for the best performances without limiting, *a priori*, the compilation time and the memory usage. One of the major results of this collaboration was to propose to exploit SSA so as to design a register allocator in two phases, with one spilling phase relatively target-independent, then the allocator itself, which takes into account architectural constraints and optimizes

other aspects (in particular, coalescing). This new way of considering register allocation has shown its interest for aggressive static compilation. But it offered three other perspectives:

- A simplification of the allocator, which again goes toward a more reliable compiler design, based on static single assignment.
- The possibility to handle the hardest part, the spilling phase, as a preliminary phase, thus a good candidate for split compilation.
- The possibility of a fast allocator, with a much higher quality than usual JIT approaches such as "linear scan", thus suitable for virtualization and JIT compilation.

These additional possibilities have been the heart of our research on back-end optimizations in Compsys II. The objective of the Mediacom project with STMicroelectronics was to address them. More generally, in Compsys II, our goal was to continue to develop our activity on code optimizations, exploiting SSA properties, following our two-phases strategy:

- First, revisit 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, exploit 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.

An important challenge was also to consider more code optimizations and more architectural features, such as registers with aliasing, predication, and, possibly in a longer term, vectorization/parallelization.

3.3. High-Level Program Analysis and Transformations

3.3.1. High-Level Synthesis Context

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. We believe that our expertise in compilation and automatic parallelization can contribute to the development of the needed tools.

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

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

Some intermediate tools exist 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. 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. 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. It is 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.

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, are not directly applicable, firstly because they pay poor attention to locality, which is of paramount importance in hardware. Besides, their aim is to extract all the parallelism in the source code; they 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 both resource and locality 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.

In summary, as for our activity on back-end code optimizations, which is decomposed into two complementary activities, aggressive and just-in-time compilation, we focus our activity on high-level synthesis on two 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.
- Developing concepts and techniques in a more global view of high-level synthesis, starting from specification languages down to hardware implementation.

We now give more details on the program optimizations and transformations we want to consider and on our methodology.

3.3.2. Specifications, Transformations, Code Generation for High-Level Synthesis

Before contributing to high-level synthesis, 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 between FPGAs and the host processor.

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 flow from an input channel to the output. Here,

data is 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. 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 (CRP, memory based). It is an interesting fact that several researchers have investigated translation from process networks [20] and to process networks [31], [32].

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 is best 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. In order to bypass this difficulty, one can 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. One can prove that the determinacy property still holds [21]. 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 [23]. Each statement (or group of statements) may be considered a process, and the source computation indicates where to implement communication channels. 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. Here, communications are associated to so-called uncut dependences, i.e., dependences which cross thread boundaries. In both approaches, the main question is whether the communications can be implemented as FIFOs, or need a reordering memory. One of our research directions will be to try to take advantage of the reordering allowed by dependences to force a FIFO implementation.

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 and for minimizing the length of the reuse vector [25], techniques for data allocation in scratch-pad memory, or techniques for folding multi-dimensional arrays [17]. 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? How does one take into account latency or throughput constraints, or bandwidth constraints for input and output channels? 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 tools) or as transformations to generate directly VHDL codes. 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.

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 naïve implementation may destroy the benefits of high-level optimization. There are two possibilities here as suggested before; one may target another high-level synthesis tool, or one may target directly VHDL. Each approach has its advantages and drawbacks. However, in both situations, all such tools require that the input program respects some strong constraints on the code shape, array accesses, memory accesses, communication protocols, etc. Furthermore, to get the tool to do what the user wants requires a lot of program tuning, i.e., of program rewriting. What can be automated in this rewriting process? Semi-automated?

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 systems and processors, 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.

5. 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 Zettice, in general not available, c) the development of algorithms within the back-end compilers of STMicroelectronics and/or Kalray.

Many tools based on the polyhedral representation of codes with nested loops 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 "polytope model" view of codes is now widely accepted: it used by Inria projects-teams Cairn and Alchemy/Parkas, 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. More recently, 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 polytope 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/.

The other activity concerns the developments within the compilers of industrial partners such as STMicroelectronics and Kalray. These are not stand-alone tools, which could be used externally, but algorithms and data structures implemented inside the LAO back-end compiler or other compiler branches, year after year, with the help of STMicroelectronics or Kalray colleagues. They are also completed by important efforts for integration and evaluation within the complete compiler toolchains. They concern exact (ILP-based) methods, algorithms for aggressive optimizations, techniques for just-in-time compilation, code representations, and for improving the design of the compiler.

More recently, an important development activity has been started in the context of the Zettice start-up project (see Section 7.3). An important effort of applied research and software development has been achieved since, which results, in particular, in two major software developments: Dcc (DPN C Compiler) and IceGEN. These tools are outlined in Sections 5.8 and 5.9.

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 Arénaire).

5.3. 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 5.4 and 5.6). 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 [16], 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.

5.4. Cl@k

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

Cl@k (Critical LAttice Kernel) is a stand-alone optimization tool 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. It 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.)

Its application to array contraction has been implemented by Christophe Alias in a tool called Bee (see Section 5.6). Bee uses Rose as a parser, analyzes the lifetimes of the elements of the arrays to be compressed, and builds the necessary input for Cl@k, i.e., the 0-symmetric polytope of conflicting differences. Then, Bee computes the array contraction mapping from the lattice provided by Cl@k and generates the final program with contracted arrays. More details on the underlying theory are available in previous reports. Cl@k can be viewed as a complement to the Polylib suite, enabling yet another kind of optimizations on polyhedra. Initially, Bee was the complement of Cl@k in terms of its application to memory reuse. Now, Bee is a stand-alone tool that contains more and more features for program analysis and loop transformations.

5.5. PoCo

Participant: Christophe Alias.

PoCo is a polyhedral compilation framework providing many features to quickly prototype program analysis and optimizations in the polyhedral model. Essentially, PoCo provides:

- A C front-end extracting the polyhedral representation of the input program. The parser itself is based on EDG (*via* Rose), an industrial C/C++ parser from Edison group used in Intel compilers.
- An extended language of pragmas to feed the source code with compilation directives (a schedule, for example).
- A symbolic layer on polyhedral libraries Polylib (set operations on polyhedra) and Piplib (parameterized ILP, see Section 5.2). This feature simplifies drastically the developer task.
- Some dependence analysis (polyhedral dependence graph, array dataflow analysis), array region analysis, array liveness analysis.
- A C and VHDL code generation based on the ideas of P. Boulet and P. Feautrier [16].

The array dataflow analysis (ADA) of PoCo has been extended to a FADA (Fuzzy ADA) by M. Belaoucha, former PhD student at Université de Versailles. FADALib is available at https://bitbucket.org/mbelaoucha/fadalib. PoCo has been developed by Christophe Alias. It represents more than 19000 lines of C++ code. The tools Bee, Chuba, and RanK presented thereafter make an extensive use of PoCo abstractions.

5.6. Bee

Participants: Christophe Alias, Alain Darte.

Bee is a source-to-source optimizer that contracts the temporary arrays of a program under scheduling constraints. Bee bridges the gap between the mathematical optimization framework described in [17] and implemented in Cl@k (Section 5.4), and effective source-to-source array contraction. Bee applies a precise lifetime analysis for arrays to build the mathematical input of Cl@k. Then, Bee derives the array allocations from the basis found by Cl@k and generates the C code accordingly. Bee is – to our knowledge – the only complete array contraction tool.

Bee is sensitive to the program schedule. This latter feature enlarges the application field of array contraction to parallel programs. For instance, it is possible to mark a loop to be software-pipelined (with an affine schedule) and to let Bee find an optimized array contraction. But the most important application is the ability to optimize communicating regular processes (CRP). Given a schedule for every process, Bee can compute an optimized size for the channels, together with their access functions (the corresponding allocations). We currently use this feature in source-to-source transformations for high-level synthesis (see Section 3.3).

- Bee was made available to STMicroelectronics as a binary.
- Bee has been transferred to the (incubated) start-up Zettice, initiated by Alexandru Plesco.
- Bee has been 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 compiler infrastructure PoCo (see Section 5.5). It represents more than 2400 lines of C++ code.

5.7. 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 optimize 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 [4].

Chuba has been implemented by Christophe Alias, using the compiler infrastructure PoCo (see Section 5.5). It represents more than 900 lines of C++. The reduced size of Chuba is mainly due to the high-level abstractions provided by PoCo.

5.8. Dcc

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

Dcc (DPN C Compiler) is the *front-end* of the HLS tool transferred to the start-up Zettice (see Section 7.3). Dcc takes as input a C program annotated with pragmas and produces an optimized data-aware process network (DPN). A DPN is a regular process network that makes explicit the I/O transfers and the synchronizations. Dcc features throughput optimization, communication vectorization, and automatic parallelization. Furthermore, Dcc applies analysis to build the DPN circuitry: multiplexing, channels sizing and allocation, FSM generation. To do so, Dcc uses extensively the analysis implemented in PoCo (Section 5.5), in particular dataflow analysis and control generation, and Bee (Section 5.6 for buffer sizing. The DPN specific analysis of Dcc is currently under patent deposit.

Dcc represents more than 3000 lines of C++ code.

5.9. IceGEN

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

IceGEN (Integrated Circuit Generator) is the *back-end* of the HLS tool transferred to the start-up Zettice (see Section 7.3). IceGEN takes as input the DPN produced by Dcc (see Section 5.8) and generates:

- a SystemC description relevant for fast and accurate circuit simulation.
- a VHDL description of the circuit, which can be mapped efficiently to an FPGA.

IceGEN makes an extensive use of the pipelined arithmetic operators of the tool FloPoCo [18] developed by Florent De Dinechin, formerly from Inria ARIC team.

IceGEN represents more than 6000 lines of C++ code.

5.10. 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 5.3), which has been greatly 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 5.11), a rudimentary VHDL generator, and a DOT generator which draws the output automaton. 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 5.12) to prove program termination.

5.11. Aspic

Participant: Laure Gonnord.

Aspic is an invariant generator for general counter automata. Used with C2fsm (see Section 5.10), it can be used to derivate invariant for numerical C programs, and also 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 5.12) 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.

5.12. 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, and has already been experimented. The theory underlying RanK was presented at SAS'10 [14].

The input of RanK is an integer automaton, computed by C2fsm (see Section 5.10), representing the control structure of the program to be analyzed. RanK uses the Aspic tool (see Section 5.11), 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 programs of the community (see the WTC benchmark suite 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 5.5). It represents more than 3000 lines of C++. The tool has been presented at the CSTVA'13 workshop [11].

5.13. 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 [15]. 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 5.12). 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++.

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 [22]. 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.

5.15. LAO Developments in Aggressive Compilation

Participants: Benoit Boissinot, Florent Bouchez, Florian Brandner, Quentin Colombet, Alain Darte, Benoît Dupont de Dinechin [Kalray], Christophe Guillon [STMicroelectronics], Sebastian Hack [Former post-doc in Compsys], Fabrice Rastello, Cédric Vincent [Former student in Compsys].

Our past aggressive optimization techniques are all implemented in stand-alone experimental tools (as for example for register coalescing algorithms) or within LAO, the back-end compiler of STMicroelectronics, or both. They concern SSA construction and destruction, instruction-cache optimizations, register allocation. Here, we report only our activities related to register allocation.

Our developments on register allocation within the STMicroelectronics compiler started when Cédric Vincent (bachelor degree, under Alain Darte supervision) developed a complete register allocator in LAO, the assembly-code optimizer of STMicroelectronics. This was the first time a complete implementation was done with success, outside the MCDT (now CEC) team, in their optimizer. This continued with developments made during the master internships and PhD theses of Florent Bouchez, Benoit Boissinot, and Quentin Colombet, and post-doctoral works of Sebastian Hack and Florian Brandner. In 2009, Quentin Colombet started to develop and integrate into the main trunk of LAO a full implementation of a two-phases register allocation. This implementation now includes two different decoupled spilling phases, the first one as described in Sebastian Hack's PhD thesis and a second ILP-based solution. It also includes an up-to-date graph-based register coalescing. Finally, since all these optimizations take place under SSA form, it includes also a mechanism for going out of colored-SSA (register-allocated SSA) form that can handle critical edges and does further optimizations. See details in the "new results" presented in previous Compsys activity reports.

5.16. LAO Developments in JIT Compilation

Participants: Benoit Boissinot, Florian Brandner, Quentin Colombet, Alain Darte, Benoît Dupont de Dinechin [Kalray], Christophe Guillon [STMicroelectronics], Fabrice Rastello.

The other side of our work in the STMicroelectronics compiler LAO has been to adapt the compiler to make it more suitable for JIT compilation. This means lowering the time and space complexity of several algorithms. In particular we implemented our fast out-of-SSA translation method, and we programmed and tested various ways to compute the liveness information. Recent efforts also focused on developing a tree-scan register allocator for the JIT part of the compiler, in particular a JIT conservative coalescing. The technique is to bias the tree-scan coalescing, taking into account register constraints, with the result of a JIT aggressive coalescing. See details in the "new results" presented in previous Compsys activity reports.

5.17. Low-Level Exchange Format (TireX) and Minimalist Intermediate Representation (MinIR)

Participants: Christophe Guillon [STMicroelectronics], Fabrice Rastello, Benoît Dupont de Dinechin [Kalray].

Most compilers define their own intermediate representation (IR) to be able to work on a program. Sometimes, they even use a different representation for each representation level, from source code parsing to the final object code generation. MinIR (Minimalist Intermediate Representation) is a new intermediate representation, designed to ease the interconnection of compilers, static analyzers, code generators, and other tools. In addition to the specification of MinIR, generic core tools have been developed to offer a basic toolkit and to help the connection of client tools. MinIR generators exist for several compilers, and different analyzers are developed as a testbed to rapidly prototype different static analyses over SSA code. This new common format enables the comparison of the code generator of several production compilers, and simplifies the connection of external tools to existing compilers.

MinIR has been extended into TireX, a Textual Intermediate Representation for EXchanging target-level information between compiler optimizers and whole or parts of code generators (a.k.a., compiler back-end). The first motivation for this intermediate representation is to factor target-specific compiler optimizations into a single component, in case several compilers need to be maintained for a particular target (e.g., operating system compiler and application code compiler). Another motivation is to reduce the run-time cost of JIT compilation and of mixed mode execution, since the program to compile is already in a representation lowered to the level of the target processor. Beside the lowering at the target level, the extensions of MinIR include the program data stream and loop scoped information. TireX is currently produced by the Open64/Path64 and the LLVM compilers, with a GCC producer under work. It is used by the LAO code generator.

Detailed information, generic core tools, and LLVM IR based generator for MinIR are available at http://www.assembla.com/spaces/minir-dev/wiki. MinIR was presented at WIR'11 [29].

6. New Results

6.1. Parameterized Construction of Program Representations for Sparse Dataflow Analysiss

Participants: André Tavares [UFMG, Belo Horizonte, Brazil], Benoit Boissinot [Ex-Compsys, Google Zurich], Fernando Magno Quintão Pereira [UFMG, Belo Horizonte, Brazil], Fabrice Rastello.

Data-flow analysis usually associates information with control flow regions. Informally, if these regions are too small like a point between two consecutive statements, we call the analysis dense. On the other hand, if these regions include many such points, then we call it sparse. This work presents a systematic method to build program representations that support sparse analyses. To pave the way to this framework, we clarify the literature about well-known intermediate program representations. We show that our approach, subsumes, up to parameter choices, many of these representations, such as the SSA, SSI, and e-SSA forms. In particular, our algorithms are faster, simpler and more frugal than the previous techniques used to construct SSI (static single information) form programs. We produce intermediate representations isomorphic to Choi *et al.*'s sparse evaluation graphs (SEG) for the family of data-flow problems that can be partitioned by variables. However, contrary to SEGs, we can handle - sparsely - problems that are not in this family. We have tested our ideas in the LLVM compiler, comparing different program representations in terms of size and construction time.

This work is part of the collaboration with UFMG (see Section 8.4) and has been accepted for presentation and publication at CC'14 (Compiler Construction Conference) [9].

6.2. A Framework for Enhancing Data Reuse via Associative Reordering

Participants: Kevin Stock [OSU, Columbus, USA], Louis-Noël Pouchet [UCLA, Los Angeles, USA], Fabrice Rastello, J. Ramanujam [LSU, Houston, USA], P. Sadayappan [OSU, Columbus, USA].

The freedom to reorder computations involving associative operators has been widely recognized and exploited in designing parallel algorithms and to a more limited extent in optimizing compilers. However, the use of associative reordering for enhancing data locality has not been previously explored to our knowledge.

In this work, we develop a novel framework for utilizing associativity of operations in regular loop computations to enhance register reuse. Stencils represent a particular class of important computations where our optimization framework can be applied to enhance performance. We use a multi-dimensional retiming formalism to characterize the space of valid transformations and to generate the transformed code. Experimental results demonstrate the effectiveness of the framework.

This work has been submitted to PLDI'14 and is part of the collaboration with P. Sadayappan from the University of Columbus (OSU) (see Section 8.4).

6.3. Function Cloning Revisited

Participants: Matheus Vilela [UFMG, Belo Horizonte, Brazil], Guilherme Balena [UFMG, Belo Horizonte, Brazil], Guilherme Marques [UFMG, Belo Horizonte, Brazil], Fernando Magno Quintão Pereira [UFMG, Belo Horizonte, Brazil], Fabrice Rastello.

Compilers rely on two main techniques to implement optimizations that depend on the calling context of functions: inlining and cloning. Historically, function inlining has seen more widespread use, as it tends to be more effective in practice. Yet, function cloning provides benefits that inline leaves behind. In particular, cloning gives the program developer a way to fight performance bugs, because it generates reusable code. Furthermore, it deals with recursion more naturally. Finally, it might lead to less code expansion, the inlining's nemesis.

In this work, we revisited function cloning under the light of these benefits. We discuss four independent code specialization techniques based on function cloning, which, although simple, find wide applicability, even in highly optimized benchmarks, such as SPEC CPU 2006. We claim that our optimizations are easy to implement and to deploy. We use Wu and Larus's well-known static profiling heuristic to measure the profitability of a clone. This metric gives us a concrete way to point out to program developers potential performance bugs, and gives us a metric to decide if we should keep a clone or not. By implementing our ideas in LLVM, we have been able to speed up some of the SPEC benchmarks by up to 6% on top of the -O2 optimization level.

This work is part of the collaboration with UFMG (see Section 8.4) and was also done in the context of the collaboration with Kalray and the ManycoreLabs project (see Section 7.2).

6.4. Register Allocation and Promotion through Combined Instruction Scheduling, Loop Splitting and Unrolling

Participants: P. Sadayappan [OSU, Columbus, USA], Fabrice Rastello, Lukasz Domanaga.

Register allocation is a much studied problem. A particularly important context for optimizing register allocation is within loops, since a significant fraction of the execution time of programs is often inside loop code. A variety of algorithms have been proposed in the past for register allocation, but the complexity of the problem has resulted in a decoupling of several important aspects, including loop unrolling, loop fission, register promotion, and instruction reordering.

In this work, we develop an approach to register allocation and promotion in a unified optimization framework that simultaneously considers the impact of loop unrolling, loop splitting, and instruction scheduling. This is done via a novel instruction tiling approach where instructions within a loop are represented along one dimension and innermost loop iterations along the other dimension. By exploiting the regularity along the loop dimension, and a constrained intra-tile execution order, the problem of optimizing register pressure is cast in a constraint programming formalism. Experimental results are provided from thousands of innermost loops extracted from the SPEC benchmarks, demonstrating improvements over the current state of the art.

This work is part of the collaboration with OSU (see Section 8.4) and was also done in the context of the collaboration with Kalray and the ManycoreLabs project (see Section 7.2). It contributes to the developments of the Tirex toolbox (see 5.17). It has also been submitted to PLDI'14.

6.5. Beyond Reuse Distance Analysis: Dynamic Analysis for Characterization of Data Locality Potential

Participants: Naznin Fauzia [OSU, Columbus, USA], Venmugil Elango [OSU, Columbus, USA], Mahesh Ravishankar [OSU, Columbus, USA], J. (ram) Ramanujam [LSU, Houston, USA], Fabrice Rastello, Atanas Rountev [OSU, Columbus, USA], Louis-Noël Pouchet [UCLA, Los Angeles, USA], P. Sadayappan [OSU, Columbus, USA].

Emerging computer architectures will feature drastically decreased flops/byte (ratio of peak processing rate to memory bandwidth) as highlighted by recent studies on Exascale architectural trends. Further, flops are getting cheaper while the energy cost of data movement is increasingly dominant. The understanding and characterization of data locality properties of computations is critical in order to guide efforts to enhance data locality.

Reuse distance analysis of memory address traces is a valuable tool to perform data locality characterization of programs. A single reuse distance analysis can be used to estimate the number of cache misses in a fully associative LRU cache of any size, thereby providing estimates on the minimum bandwidth requirements at different levels of the memory hierarchy to avoid being bandwidth bound. However, such an analysis only holds for the particular execution order that produced the trace. It cannot estimate potential improvement in data locality through dependence preserving transformations that change the execution schedule of the operations in the computation.

In this work, we develop a novel dynamic analysis approach to characterize the inherent locality properties of a computation and thereby assess the potential for data locality enhancement via dependence preserving transformations. The execution trace of a code is analyzed to extract a computational directed acyclic graph (CDAG) of the data dependences. The CDAG is then partitioned into convex subsets, and the convex partitioning is used to reorder the operations in the execution trace to enhance data locality. The approach enables us to go beyond reuse distance analysis of a single specific order of execution of the operations of a computation in characterization of its data locality properties. It can serve a valuable role in identifying promising code regions for manual transformation, as well as assessing the effectiveness of compiler transformations for data locality enhancement. We demonstrate the effectiveness of the approach using a number of benchmarks, including case studies where the potential shown by the analysis is exploited to achieve lower data movement costs and better performance.

This work is part of the collaboration with OSU (see Section 8.4) and has been accepted for publication at ACM TACO [2].

6.6. Characterizing the Inherent Data Movement Complexity of Computations via Lower Bounds

Participants: P. Sadayappan [OSU, Columbus, USA], Venmugil Elango [OSU, Columbus, USA], J. (ram) Ramanujam [LSU, Houston, USA], Louis-Noël Pouchet [UCLA, Los Angeles, USA], Fabrice Rastello.

Technology trends will cause data movement to account for the majority of energy expenditure and execution time on emerging computers. Therefore, computational complexity will no longer be a sufficient metric for comparing algorithms, and a fundamental characterization of data access complexity will be increasingly important. Although the problem of characterizing data access complexity has been modeled previously using the formalism of Hong & Kung's red/blue pebble game [27], applicability of previously-developed approaches has been extremely limited. We improve on prior work in several ways: 1) we develop an approach to composing lower bounds from arbitrary decompositions of computational directed acyclic graphs, thereby eliminating a significant limitation of previous approaches that required homogeneity of analyzed computations, 2) we develop a complementary graph min-cut based strategy to Hong & Kung's S-partitioning approach, and 3) we develop an automated approach to generate concrete I/O lower bounds of arbitrary, possibly irregular computational directed acyclic graphs. We provide experimental results demonstrating the utility of the developed approach.

This work has been submitted to PLDI'14 and is part of an informal collaboration with P. Sadayappan from the University of Columbus (OSU) (see Section 8.4).

6.7. Enhancing the Compilation of Synchronous Dataflow Programs

Participants: Paul Feautrier, Abdoulaye Gamatié [LIRMM, Montpellier], Laure Gonnord.

In this work [12], which is an extension of [26], we propose an enhancement of the compilation of synchronous programs with a combined numerical-Boolean abstraction. While our approach applies to synchronous dataflow languages in general, here, we consider the SIGNAL language for illustration. In the new abstraction, every signal in a program is associated with a pair of the form (`clock`, `value`), where `clock` is a Boolean function and `value` is a Boolean or numeric function. Given the performance level reached by recent progress in satisfiability modulo theory (SMT), we use an SMT solver to reason on this abstraction. Through sample examples, we show how our solution is used to determine absence of reaction captured by empty clocks; mutual exclusion captured by two or more clocks whose associated signals never occur at the same time; or hierarchical control of component activations via clock inclusion. We also show that the analysis improves the quality of the code generated automatically by a compiler, e.g., a code with smaller footprint, or a code executed more efficiently thanks to optimizations enabled by the new abstraction. The implementation of the whole approach includes a translator of synchronous programs towards the standard input format of SMT solvers, and an ad hoc SMT solver that integrates advanced functionalities to cope with the issues of interest in this work. These results have been published in 2013 (but considered as published in 2012) in the CSI Journal of Computing [24].

6.8. Synthesis of Ranking Functions using Extremal Counter-Examples

Participants: David Monniaux [Verimag, Grenoble], Lucas Séguinot [Student at ENS Cachan Bretagne], Laure Gonnord.

In [14], 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-untractable 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*.

We are currently testing our preliminary implementation and submitting a paper on these new results.

6.9. Data-Aware Process Networks

Participants: Christophe Alias, Alexandru Plesco.

The following results concern the applied research activities directly linked to the Zettice start-up (see Section 7.3), which aims at applying polyhedral techniques to high-level circuit synthesis (HLS). Following the guidelines of Inria DTI, as this research aims to be transferred, these results are not published before being "protected" or exploited. An Inria patent deposit is currently processed.

- **Data-aware process networks (DPN).** This is the intermediate representation of the HLS flow. DPN is a parallel execution model fitting the hardware constraints of circuit synthesis, in which the data transfer and the synchronizations are made explicit. We formally described the DPN model and a translation scheme from C programs, and we showed the consistency in the meaning where any terminating sequential program is translated to an equivalent DPN, guaranteed to be deadlock free.
- **Front-end analysis.** We designed many program analyses to produce a quality DPN from a C program:
  - *Throughput optimization.* A I/O scheme has been designed, with the corresponding compiler analysis, to minimize the I/O traffic with the external memory. This allows us to balance efficiently the spilling of temporary value to the memory, and the local buffer size. This scheme impacts the DPN structure itself.
  - *Communication vectorization.* The matrix structure of the memory allows us to load data by chunks. A polyhedral analysis has been designed to solve this issue.
  - *Synchronization scheme.* As parallel units need to communicate intermediate results, synchronizations must be ensured.Unlike KPN, DPN do not use FIFO, but buffers, which required an efficient synchronization mechanism.
- **Back-end analysis.** Once generated, a DPN must be mapped to an FPGA. This raises many interesting issues:
  - *Pipeline completion.* Data paths make an extensive use of pipelined operators, which delays the signal. An algorithm has been designed to enforce the time coherence of signals.
  - *Polyhedral units.* DPNs make an extensive use of piece-wise affine functions, which must be mapped properly to ensure the efficiency of the whole system. A preliminary algorithm has been designed to reach a correct trade-off between critical path size and LUT usage.

All these analyses have been fully implemented. The tool Dcc (DPN C Compiler) implements all the front-end analyses. The tool IceGEN implements the back-end analysis.

6.10. Program Equivalence Modulo A/C (Associativity/Commutativity)

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

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 focus on programs represented as a system of affine recurrence equations (SARE), defined over parametric polyhedral domains, a well-known formalism for the *polyhedral model*, which includes as a proper subset, the class of affine control loop programs. Several semi-algorithms for program equivalence have already been proposed for this class. A few of them take into account algebraic properties such as associativity and commutativity. However, to the best of our knowledge, none of them is able to manage reductions, i.e., accumulations of a parametric number of sub-expressions using an associative and commutative operator.

Our contributions are:

- An equivalence checking algorithm able to manage associativity and commutativity properties. Our method subsumes the previous approaches and is, to the best of our knowledge, the first one able to manage these properties over a parametric number of expressions.
- A semi-algorithm to construct a perfect matching problem on a parametric bipartite graph. We partially solve this problem through a heuristic based on the augmenting path algorithm. This heuristic is able to find a set of non-interfering augmenting paths to improve a proposed maximum matching, as long as these augmenting paths do not have a parametric length.

A preliminary implementation is under development. This work has been submitted to ESOP'14.

6.11. Constant Aspect-Ratio Parametric Tiling

Participants: Guillaume Iooss [PhD student], Sanjay Rajopadhye [Colorado State University], Christophe Alias, Yun Zou [PhD student, Colorado State University].

Parametric tiling is a well-known transformation that is widely used to improve locality, parallelism, and granularity. 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.

This result consists on a method to stay polyhedral in a special case of parametric tiling, where all the dimensions are tiled and all the tile sizes are constant multiples of a single tile size parameter. We call this *Constant Aspect Ratio Tiling*. We show how to mathematically transform a polyhedron and an affine function into their tiled counterpart and show how to obtain good generated code.

This work has been accepted for publication at IMPACT'14 [8].

6.12. Parametric Tiling with Inter-Tile Data Reuse

Participants: Alain Darte, Alexandre Isoard.

Loop tiling is a loop transformation widely used to improve spatial and temporal data locality, increase computation granularity, and enable blocking algorithms, which are particularly useful when offloading kernels on platforms with small memories. When hardware caches are not available, data transfers must be software-managed: they can be reduced by exploiting data reuse between tiles and, this way, avoid some useless external communications. An important parameter of loop tiling is the sizes of the tiles, which impact the size of the necessary local memory. However, for most analyzes that involve 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. In this work, we showed that, actually, parametric tiling with inter-tile data reuse is nevertheless possible.

Our solution is the first parametric solution for generating the memory transfers needed when a kernel is offloaded to a distant accelerator, tile by tile after loop tiling, and when all intermediate results are stored locally on the accelerator. For such computations, there is a complete decoupling between loads and stores, and when a value has been defined in a previous tile, it has to be loaded from the local memory and not from the distant memory as this memory is not yet up-to-date. In other words, inter-tile reuse is mandatory. This also saves external communications. Our solution is parametric in the sense that we derive the set of loads and stores from and to the distant memory with the tile sizes as parameters. Although the direct formulation is quadratic, we can still solve it in an affine way by developing techniques that consider, in the analysis, all (unaligned) possible tiles obtained by translation and not just those that belong to a tiling (partitioning) of the iteration space. We were able to use a similar technique to also parameterize the computations of local memory sizes, thanks to parametric lifetime analysis and folding with modulos, even for pipeline schedules similar to double buffering. Our method is currently implemented with the `iscc` calculator of `ISL`, a library for the manipulation of integer sets defined with Presburger arithmetic.

Also, the whole analysis can handle approximations thanks to the introduction of the concept of pointwise functions, well suited to deal with unaligned tiles. We believe that this 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.

This work has been accepted for publication at IMPACT'14 [5].

6.13. Data Races in the Parallel Language X10

Participants: Tomofumi Yuki [Colorado State University and Inria/IRISA], Paul Feautrier, Sanjay Rajopadhye [Colorado State University], Vijay Saraswat [IBM Research].

Parallel programmers are now required to efficiently utilize the massive amount of parallelism provided by multi-core and many-core systems. Parallel programming is difficult, and the existing tools are mostly low-level extensions to sequential languages or libraries. As an effort to improve this situation, several groups have initiated the design of parallel programming languages, mostly based on the partitioned global address space (PGAS) paradigm. One of these languages is X10, which is developed at IBM Research by a team led by Vijay Saraswat.

While such languages hide the low-level details of parallel programming, they cannot guarantee that the object code will be correct by construction. Parallelism introduces two new types of bugs: non-determinism and deadlocks, and experience shows that it is possible to guarantee the absence of one type but not both. X10 programs are guaranteed deadlock-free but may have non-determinism. Non-determinism can be detected at runtime, but this approach cannot give absolute guarantees. However, it is possible, at least for a restricted class of X10 programs, to check for non-determinism at compile time.

The first step in this direction is to define the *polyhedral fragment* of X10, in which the only control constructs are `for` loops with affine bounds, and the only data structures are arrays with affine subscripts. X10 has many parallel constructs: as a first effort, we focused on `async`, which creates an activity (lightweight thread) and `finish`, which waits for termination of all impending activities. The execution order (or *happens-before relation*) of such a program is an incomplete lexicographic order, in which terms relating operations in different activities are removed. The dataflow analysis method of [23] has to be adapted to a partial execution order, which may have many extrema instead of a unique maximum. Multiple extrema denote data races, thus non-determinism. A detector along these lines has been implemented and presented at PPoPP'13 (Symposium on Principles and Practice of Parallel Programming) [10].

X10 other parallel programming primitive directives are *clocks* and `atomic`. The `at` construct allows downloading a computation to another *place*. Clocks are a dynamic version of barriers. Their analysis involves counting their instances. For polyhedral programs, this can be done using the Ehrhart and Barvinok theories; the results are polynomials. Checking whether clocks remove non-determinism involves finding integer roots and hence is undecidable. However, modern SMT solvers are able to solve most of these problems. The resulting paper [13] has been submitted to the ECOOP conference.

6.14. Clock Removal in X10

Participants: Paul Feautrier, Eric Violard [Inria/Camus], Alain Ketterlin [Inria/Camus].

In the light of the previous work on the determinism of X10, a natural question is: are the parallel programming directives of X10 redundant? The answer is yes, at least for static control programs, i.e., programs in which the set of operations and their execution order do not depend on the input data. The basic idea is that the synchronization which occurs when several activities execute an advance is similar to the synchronization at the end of a finish. If one is able to count advances, one may construct a front by gathering all operations with the same advance count. Each front is executed inside one finish, and fronts are executed sequentially in order of increasing counts. For polyhedral programs, advance counting can be done at compile time. If the counts are affine functions, the restructuring can be done by classical polyhedral code generators like CLooG, and no overhead is incurred. For polynomial counts, one overall enclosing loop must be added, but the resulting program can usually be optimized by simple loop transformations, e.g., pushing guards into enclosing loop bounds. For arbitrary programs, the counts have to be computed dynamically; this is possible only if the program has static control.

This result does not contradict the previous undecidability proof (Section 6.13), as the translation of a polyhedral program is usually not polyhedral. Application of the method to a set of simple kernels has shown significant speedups. The interpretation of this result is that, at least in the present state of the X10 runtime, the implementation of the `async` primitive is more mature than the implementation of clocks. A paper on this topic has been accepted at CC'14 (Compiler Construction Conference) [7].

6.15. Static Analysis of OpenStream Programs

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

The objective of the collaboration between the Compsys and Parkas teams in the ManycoreLabs project (Section 7.2) is to evaluate the possibility of applying polyhedral techniques to the parallel language OpenStream, which is developed by Inria Parkas. When applicable, these techniques are invaluable for compile-time debugging and for improving the target code for a better adaptation to the target architecture.

OpenStream is a two-level language, in which a sequential control code directs the initialization of parallel task instances that communicate through *streams*. OpenStream programs are deterministic by construction, but may have deadlocks. If the control code is polyhedral, one may statically compute, for each task instance, its read and write indices for each stream. These indices may be polynomials of arbitrary degree. When linear, the full power of the polyhedral model may be brought to bear for dependence and dataflow analysis, scheduling and deadlock detection, and program transformations.

In the general case, one can think of two approaches: the first one consists in over-approximating dependences until problems become linear. In the second approach, one first leverages modern developments in SMT solvers, which allow them to solve polynomial problems, albeit with no guarantee of success. Furthermore, the task index functions have special properties that may be used to construct original analysis algorithms. Three preliminary results in this direction:

- the proof that deadlock detection is undecidable in general, thanks to an adaptation of the proof designed for X10 (Section 6.13),
- a characterization of deadlocks in terms of dependence graphs, which implies that streams can be safely bounded as soon as a schedule exists with such sizes,
- a preliminary analysis of some solvable cases.

A document is available as Deliverable 2.5.3 for the ManycoreLabs project.

6.16. Array Contraction in Parallel Programs

Participants: Alain Darte, Alexandre Isoard.

Array contraction is a technique to reuse array elements when they are dead, in a form of array folding. A standard technique for array contraction is to use affine remappings with modulos. When the modulo is equal to 1, this corresponds to the removal of the corresponding array dimension. Array contraction is well-known for sequential programs, after element-wise array liveness analysis. It has also been customized for parallel codes obtained through affine schedules by Lefebvre and Feautrier, and Quilleré-Rajopadhye, both frameworks being generalized by the lattice-based memory allocation framework of Darte, Schreiber, and Villard [17] and the construction of the set of conflicting array indices. We showed how the same framework can be used for a larger range of parallel programs, including programs with outer parallel loops, programs exhibiting pipelining, a subset of X10, etc. The optimality of the construction can be shown, despite a related (but actually non-contradictory here) NP-completeness result for worst-case of register pressure in the context of register allocation. A research report on this topic is in preparation.

7. Bilateral Contracts and Grants with Industry

7.1. Tirex Contract with Kalray

Compsys has a contract with Kalray called Tirex. The goal of this project is to prototype within the TireX toolbox (see Section 5.17) some new profiling/analysis techniques necessary to enable cloning. Because of the current financial problems encountered by Kalray, the efforts related to this project have been frozen until further notice.

7.2. ManycoreLabs Project with Kalray

Compsys is part of a bilateral grant 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.

The cloning/specialization work summarized in Section 6.3 and the generalized register tiling work summarized in Section 6.4 have been done in the context of this grant and correspond to WP 3.3.3. The research on OpenStream described in Section 6.15 corresponds to WP 2.5.3.

7.3. Technological Transfer Towards Zettice Start-Up

Participants: Christophe Alias, Adrian Muresan [Zettice], Alexandru Plesco [Zettice].

The Zettice start-up project has been initiated by Alexandru Plesco and Christophe Alias in March 2011, with the idea of transferring some of the research concepts emerging from the polyhedral model to the context of high-level circuit synthesis. Since, an important amount of applied research has been achieved to propose an effective technology ready for industrial transfer. From an academic perspective, Zettice is a unique opportunity to cover all the aspects of high-level synthesis from the front-end aspects (polyhedral code analysis and optimization) to the back-end aspects (pipelining, retiming, FPGA mapping) providing a global knowledge of relevant industrial issues.

Zettice received in 2012 the *"lean start-up award"* of the startup weekend labs 2012, the *"most exciting start-up mention"* at SAME 2012, and the *concours Crealys Excel&Rate 2012* grant (30 Keuros). In 2013, Zettice won the *concours OSEO 2013* grant (Banque Publique d'Investissement, 40 Keuros) and the *"most promising start-up award"* at SAME 2013.

A patent is under deposit. The research results related to Zettice are presented in Section 6.9. The software tools developed in the context of Zettice are Dcc (see Section 5.8) and IceGEN (see Section 5.9).

8. Partnerships and Cooperations

8.1. Regional Initiatives

Compsys has increased its relationship with the CITI laboratory (Insa-Lyon) and, in particular, the team of Tanguy Risset (Socrate Inria project http://www.citi-lab.fr/team/socrate/). Compsys and Socrate made several common working groups in 2012 and 2013, and are mutually invited to seminars organized by the other team. Streaming languages are a common topic of interest. In this context, Socrate, with the help of Compsys, will organize a thematic day (April 14, 2014) on the "compilation and execution of streaming programs", in Domaine des Hautannes, St Germain au Mont d'Or. Lionel Morel and Laure Gonnord have also common topics of interest.

Compsys has stronger connections with the Grame music/computer laboratory (http://www.grame.fr) in Lyon and, in particular, Yann Orlarey, also due to common interests on streaming languages, in particular the language Faust developed by Grame. Yann Orlarey was one of the invited speaker of the keynotes on parallel languages (see the description the thematic quarter on compilation in Section 9.1.2). Alexandre Isoard's Master 1 training period was on Faust, co-advised by Alain Darte and Yann Orlarey. For 2014, Laure Gonnord and Yann Orlarey proposed a Master research topic on the generation of invariants for the Faust language.

Compsys is also involved in the Labex MILYON (Mathématiques et Informatique Fondamentale de Lyon), which regroups Institut Camille Jordan, and the mathematics and computer science labs of ENS-Lyon. The aim of MILYON is "to strengthen our international relationships, in particular by organizing thematic quarters which will allow world experts of a subject to gather in Lyon and work together in a stimulating environment." In this context, Compsys organized a thematic quarter on compilation from April 2013 to July 2013, see details in Section 9.1.2. Compsys also follows or participates to the activities of LyonCalcul (http://lyoncalcul.univ-lyon1.fr/), a network to federate activities on computing in Lyon.

8.2. National Initiatives

8.2.1. CNRS PEPS

Christophe Alias and Laure Gonnord initiated with the DART/Emeraude team at LIFL Laboratory (University of Lille) a CNRS PEPS ("Projets Exploratoire Premier Soutien") called "HLS and real time" (8 Keuros/year, during two years in 2011-2013). The goal of this project is to investigate how to introduce real-time constraints in the high-level synthesis workflow.

8.2.2. Inria AEN MULTICORE

Fabrice Rastello is part of an Inria Large Scale Initiative (AEN: action d'envergure nationale) called MULTICORE, which regroups researchers from seven teams: Camus, Regal, Alf, Runtime, Algorille, Dali, and thus Compsys on "Large scale multicore virtualization for performance scaling and portability". One of the goals of this project is to enable loop transformations by combining dynamic and static analysis/compilation techniques.

8.2.3. French Compiler Community

The french compiler community is now well identified and is visible through its web-page http://compilation.gforge.inria.fr/. The "journées françaises de la compilation" were initiated in 2010 and are still animated by Fabrice Rastello and Laure Gonnord as a biannual event. Their local organization is handled alternately by the different research teams: Lyon (by Compsys) in Summer 2010, Aussois in Winter 2010, Dinard in Spring 2011, St Hippolyte in Autumn 2011, Rennes in Summer 2012, Annecy (by Compsys again) in Spring 2013, Dammarie-les-lys in December 2013.

8.3. European Initiatives

8.3.1. Collaborations with Major European Organizations

Alain Darte, Paul Feautrier, and Fabrice Rastello are members or affiliate members of the European Network of Excellence on High Performance and Embedded Architecture and Compilation (HiPEAC). Fabrice Rastelloattended the computing system week in may 2013 (Paris), and the computing system week in October 2013 (Tallinn). He participated to the organization of two thematic sessions in Paris: Thread Level Speculation (as chair) and Intermediate Representation (as co-organizer). The thematic quarter on compilation (see Section 9.1.2) was presented in HIPEAC info 35 (July 2013), the HIPEAC quarterly newsletter (http://www.hipeac.net/content/hipeacinfo-35-july-2013) and the keynotes on HPC languages (third event) recognized as an HIPEAC event.

8.4. International Initiatives

8.4.1. Inria International Partners

8.4.1.1. Declared Inria International Partners

- Compsys and, in particular Fabrice Rastello, has a regular collaboration with P. Sadayappan from Ohio State University (USA). This year, this collaboration led to several results, see Sections 6.2, 6.4, 6.5, and 6.6.
- Fabrice Rastello and Laure Gonnord have 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 6.1 and 6.3. Compsys also hosted Raphael Ernani Rodrigues, from the group of F. Pereira, who made part of his master in Lyon supervised by Laure Gonnord and Christophe Alias.
- Compsys and, in particular Christophe Alias, has a regular collaboration with S. Rajopadhye from Colorado State University (CSU). Guillaume Iooss is preparing a PhD through a PhD convention between Ecole normale supérieure de Lyon and Colorado State University, co-advised by Christophe Alias and Sanjay Rajopadhye. In 2013, Guillaume Iooss spent part of the summer at CSU, joined by Christophe Alias for a week. Paul Feautrier and Fabrice Rastello also made regular visits at Colorado State University in the previous years. This year, this collaboration led to several results, see Sections 6.10, 6.11, and 6.13.

8.5. International Research Visitors

8.5.1. Visits of International Scientists

8.5.1.1. Invited Researchers

Fernando Magno Quintão Pereira is visiting Fabrice Rastello for 1.5 month in early 2014. The goal of his visit is to work on dynamic analysis and cloning for loop transformations (so called hybrid compilation).

8.5.1.2. Internships

Raphael Ernani Rodrigues made part of his master Internship in Lyon in June/July 2013 under the supervision of Laure Gonnord and Christophe Alias. He worked on synthesizing preconditions that (may) ensure termination. We are currently pursuing the collaboration with him and his supervisor in Brazil, Fernando Magno Quintao Pereira (Univ. Mineas Gerais).

8.5.2. Visits to International Teams

Fabrice Rastello visited the group of P. Sadayappan (OSU) during two months, in June-July 2013, in addition to shorter stays. He worked on dynamic analysis and generalized tiling.

Alexandre Isoard did an internship at Xilinx, during 2.5 months, from June to September 2013, under the supervision of Stephen Neuendorffer, working on exploring polyhedral tools for Xilinx HLS tool.

9. Dissemination

9.1. Scientific Animation

9.1.1. Program Committees, Editorial Boards, and Reviewing Activities

- Christophe Alias was a member of the steering committee of IMPACT 2013 (International Workshop on Polyhedral Compilation Techniques, Berlin, Germany).
- Christophe Alias, Alain Darte, and Paul Feautrier were members of the program committees of IMPACT 2013 and IMPACT 2014 (Vienna, Austria).

- Christophe Alias was member of the program committee of ODES 2013 (i.e., ODES-10, 10th Workshop on Optimizations for DSP and Embedded Systems, Shenzen, China).
- Fabrice Rastello was member of the program committees of CGO 2014 (International Symposium on Code Generation and Optimization, Orlando, Florida) and CRI 2013 (Conférence de Recherche en Informatique, Yaoundé, Cameroun).
- Alain Darte was member of the program committees of DATE 2013 (Design, Automation, and Test in Europe, Grenoble, France) and DATE 2014 (Dresden, Germany), IPDPS 2013 (International Parallel and Distributed Processing Symposium, Boston, Massachusetts) and IPDPS 2014 (Phoenix, Arizona).
- Alain Darte was member of the editorial board of IEEE TECS (Transactions on Embedded Computing Systems) until end of 2013.
- Christophe Alias was a reviewer for the journals JPDC (Journal of Parallel and Distributed Computing), MICPRO (Microprocessors and Microsystems), PPL (Parallel Processing Letters), ACM TRETS (Transactions on Reconfigurable Technology and Systems), TSI (Technique et Science Informatique), CDT (IET Computers and Digital Techniques), IPL (Information Processing Letters).
- Paul Feautrier was a reviewer for ACM TECS, IJPP, IEEE TPDS, ACM TOPLAS, DATE14, IMPACT 2014.
- Alain Darte was a reviewer for DATE'14, IPDPS'14, IMPACT'14, Parallel Computing, ACM TACO, and ACM TECS.
- Laure Gonnord was a reviewer for MSR'13, DAC'13 and AMT'13.

9.1.2. Thematic Quarter on Compilation

Compsys is part of the Labex MILYON, which regroups Institut Camille Jordan, and the mathematics and computer science labs of ENS-Lyon. One of its goal is "to strengthen our international relationships, in particular by organizing thematic quarters which will allow world experts of a subject to gather in Lyon and work together in a stimulating environment." In this context, Alain Darte, helped by Alexandre Isoard and Laetitia Lecot, organized, from April to July 2013, a thematic quarter on compilation techniques (http://labexcompilation.ens-lyon.fr), with a special focus on the interactions with languages and architectures for high performance computing. This thematic quarter (with a total budget of 100 Keuros), consisted, in addition to the "french compilation days" organized separately in Annecy by Laure Gonnord and Fabrice Rastello (April 4-7, 2013), in three international scientific events organized in Lyon or the vicinity.

- A **spring school on polyhedral code analysis and optimizations** (http://labexcompilation.ens-lyon.fr/polyhedral-school), May 13-17, 2013, in Domaine des Hautannes in St Germain au Mont d'Or, the first international school on the polyhedral model and related optimizations. The school covered scheduling theory, algorithms and modeling with integer sets and relations, abstract interpretation, compilation for distributed platforms, array region analysis, vectorization and SIMD optimizations, through courses given by S. Rajopadhye (Colorado State Univ.), P. Feautrier (Compsys, ENS-Lyon), L.-N. Pouchet (UCLA), S. Verdoolaege (ENS Paris), A. Miné (ENS Paris), U. Bondhugula (IIS Bangalore), A. Darte (Compsys, CNRS), B. Creusillet (Silkan), P. Sadayappan (Ohio State Univ.), N. Vasilache (Reservoir Labs, New York). The school attracted 56 participants, half from France, but also from Germany, the USA, England, Belgium, Spain, China, India, Ireland, and Italy and, interestingly, also from groups that are not familiar with polyhedral optimizations. Roughly half of the participants were PhD students.
- A **dive in languages for high-performance computing** (http://labexcompilation.ens-lyon.fr/hpc-languages), June 29-July 2, 2013 in Résidence Villemanzy in Lyon, organized as a set of long keynotes on CAF (Coarray Fortran), UPC (Unified Parallel C), X10, Chapel, OpenACC & OpenHMPP, Liquid Metal, OmpSs, OpenStream, and some DSL approaches. The keynotes were given by a panel of international experts on compilation for high-performance computing: J. Mellor-Crummey and V. Sarkar (Rice), K. Yelick (Berkeley), R. Schreiber (HP Labs), B. Chamberlain

(Cray), D. Grove and R. Rabbah (IBM Watson), A. Cohen (Inria, ENS Paris), R. Badia (UPC Barcelona), F. Bodin (Univ. Rennes, previously Caps Entreprise), Y. Orlarey (Grame), K. Knobe (Intel, Massachusets), P. Sadayappan (Ohio State Univ.). This event regrouped 71 participants, including speakers, and, as we hoped, also attracted people from industry, and not only computer industry.

- **CPC'13, the 17th international workshop on compilers for parallel computing** (http://labexcompilation.ens-lyon.fr/cpc2013), July 3-5, 2013, in Musée Gadagne, in (old) Lyon, a venue that is held every 18 months in Europe since 1989 and that encompasses all areas of parallelism and optimization linked to compilers. The program consisted in 29 talks, from the international community on compilers for HPC (from Japan & Taiwan to the USA, and of course Europe), with 47 participants.

During this compilation thematic quarter, Paul Feautrier and Alain Darte gave the following talks:

- "Array Dataflow Analysis for Polyhedral X10 Programs" (Paul Feautrier) and "Modèles et algorithmes: comprendre de quoi on parle" (Alain Darte) at the French Compiler Community meeting (April 2-4, 2013),
- "The Care and Feeding of Polyhedra" (Paul Feautrier) and "Array Contraction with Lattice-Based Memory Allocation" (Alain Darte) at the Spring School on Polyhedral Code Analysis and Optimizations (May 13-17, 2013),
- "Determinacy Analysis of Polyhedral X10 Programs" (Paul Feautrier), paper with Alain Ketterlin and Eric Violard, at the CPC Workshop (July 3-5, 2013),

9.2. Teaching - Supervision - Juries

9.2.1. Teaching

Licence:

- Laure Gonnord, Algorithmique et programmation C (60h), L3, Université de Lille 1, Polytech'Lille.
- Laure Gonnord, Architecture des ordinateurs (25h), L3, Université de Lille 1, Polytech'Lille.
- Laure Gonnord, Algorithmique et programmation fonctionnelle et récursive (42h), L1, Université Lyon 1 Claude Bernard.
- Guillaume Iooss, LIF3: algorithmique et programmation fonctionnelle et récursive (28h), L1, Université Lyon 1 Claude Bernard.
- Guillaume Iooss, Programmation 1 (12h), L3, ENS-Lyon.
- Christophe Alias, Architecture des ordinateurs (21h TP), Université Lyon 1.
- Christophe Alias, Compilation (6h CM, 6h TP), ENSI Bourges.
- Christophe Alias, Correction de copies, concours E3A, épreuve informatique MPSI.
- Alexandre Isoard, LIF12: Système et Réseau (32h TP), L3, Université Lyon 1 Claude Bernard.

Master:

- Laure Gonnord, Compilation (24h), M1, Université Lyon 1 Claude Bernard.
- Laure Gonnord, Introduction aux systèmes et réseaux (52h), M2 Pro, Université Lyon 1.
- Christophe Alias, Compilation (24h CM), M1, ENS-Lyon.
- Christophe Alias, Compilation avancée (8h CM), M2, ENS-Lyon.
- Fabrice Rastello, Compilation avancée (6h CM), M2, ENS-Lyon.
- Fabrice Rastello, SSA-based compiler design (2 days), CRI Cameroun.

- Alexandre Isoard, Compilation (24h TP), M1, ENS-Lyon.
- Guillaume Iooss, Image (24h TP), M1, ENS-Lyon.
- Laure Gonnord also organized, for the Computer Science Department of ENS Lyon, a *research school* for Master students, on synchronous programming. The program can be found at the url: http://laure.gonnord.org/pro/research/sync_research_school.html.

9.2.2. Supervision

- PhD in progress: Guillaume Iooss, "Semantic Tiling", started on September 2011, advisors: Christophe Alias and Sanjay Rajopadhye (Associate Professor, Colorado State University).
- PhD in progress: François Gindraud, started on January 2013, advisors Fabrice Rastello, Albert Cohen (Parkas Inria team)
- PhD in progress: Duco Van Amstel, started on January 2013, advisors Fabrice Rastello, Benoit Dupont-de-Dinechin (Kalray)
- PhD in progress: Diogo Nunes Sampaio, started on October 2013, advisor Fabrice Rastello
- PhD in progress: Alexandre Isoard, started in September 2012, advisor Alain Darte

9.2.3. Juries

- Laure Gonnord participated to the Jury of Clement Guy's PhD defense (in Rennes) entitled "Facilités de typage pour l'ingénierie des Langages". This PhD was supervised by J.M. Jézéquiel (Professor, Rennes University), and B. Combemale (Assistant Professor, Rennes University) and S. Derrien (Professor, Rennes university).
- Christophe Alias participated to the Jury of Antoine Morvan's PhD defense (in Rennes) entitled "Utilisation du modèle polyédrique pour la synthèse d'architectures pipelinées". This PhD thesis was supervised by S. Derrien (Professor, Rennes University), and P. Quinton (Professor, Rennes University).
- Fabrice Rastello participated to the jury of Alexandre Carbon's PhD defense, entitled "Accélération matérielle de la compilation à la volée pour les systèmes embarqués".
- Paul Feautrier was a reviewer for the HDR of Stephane Mancini (Grenoble) and for the PhD of Amira Mensi (Paris).
- Alain Darte was a reviewer for the PhD thesis of Cupertino Miranda (Paris 11), entitled "Erbium: Reconciling languages, runtimes, compilation and optimizations for streaming applications" and supervised by Albert Cohen (DR Inria, Parkas team).

10. Bibliography

Publications of the year

Articles in International Peer-Reviewed Journals

[1] F. BRANDNER, Q. COLOMBET. *Elimination of parallel copies using code motion on data dependence graphs*, in "Computer Languages, Systems and Structures", 2013, vol. 39, n° 1, pp. 25 - 47 [*DOI :* 10.1016/J.CL.2012.09.001], http://hal.inria.fr/hal-00768781

[2] N. FAUZIA, V. ELANGO, M. RAVISHANKAR, J. RAMANUJAM, F. RASTELLO, A. ROUNTEV, L.-N. POUCHET, P. SADAYAPPAN. *Beyond Reuse Distance Analysis: Dynamic Analysis for Characterization of Data Locality Potential*, in "Transaction on Architecture and Code Optimization", December 2013, vol. 10, n° 4, http://hal.inria.fr/hal-00920031

[3] L. GONNORD, P. SCHRAMMEL. *Abstract Acceleration in Linear Relation Analysis*, in "Science of Computer Programming", 2013 [*DOI :* 10.1016/J.SCICO.2013.09.016], http://hal.inria.fr/hal-00876627, http://hal.inria.fr/hal-00787212/en

International Conferences with Proceedings

[4] C. ALIAS, A. DARTE, A. PLESCO. *Optimizing Remote Accesses for Offloaded Kernels: Application to High-Level Synthesis for FPGA*, in "Design, Automation, and Test in Europe (DATE'13)", Grenoble, France, 2013, http://hal.inria.fr/hal-00761533

[5] 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), 2014, To be published, http://hal.inria.fr/hal-00915831

[6] B. DIOUF, A. COHEN, F. RASTELLO. *A Polynomial Spilling Heuristic: Layered Allocation*, in "CGO 2013 - International Symposium on Code Generation and Optimization", Shenzhen, China, IEEE, 2013 [*DOI :* 10.1109/CGO.2013.6495005], http://hal.inria.fr/hal-00911887

[7] P. FEAUTRIER, E. VIOLARD, A. KETTERLIN. *Improving X10 Program Performances by Clock Removal*, in "Compiler Construction 2014", Grenoble, France, January 2014, http://hal.inria.fr/hal-00924206

[8] G. IOOSS, S. RAJOPADHYE, C. ALIAS, Y. ZOU. *CART: Constant Aspect Ratio Tiling*, in "IMPACT 2014", Vienna, Austria, January 2014, Not yet published, http://hal.inria.fr/hal-00915827

[9] A. TAVARES, F. RASTELLO, B. BOISSINOT, F. PEREIRA. *Parameterized Construction of Program Representations for Sparse Dataflow Analyses*, in "CC 2014 - 23rd International Conference on Compiler Construction", Grenoble, France, A. COHEN (editor), Springer, 2014, http://hal.inria.fr/hal-00921461

[10] 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

Conferences without Proceedings

[11] C. ALIAS, A. DARTE, P. FEAUTRIER, L. GONNORD. *Rank: a tool to check program termination and computational complexity*, in "Constraints in Software Testing Verification and Analysis", Luxembourg, March 2013, http://hal.inria.fr/hal-00801571

Research Reports

[12] P. FEAUTRIER, A. GAMATIÉ, L. GONNORD. , *Enhancing the Compilation of Synchronous Dataflow Programs with a Combined Numerical-Boolean Abstraction*, July 2013, http://hal.inria.fr/hal-00780521

Other Publications

[13] T. YUKI, P. FEAUTRIER, S. RAJOPADHYE, V. SARASWAT. , *Checking Race Freedom of Clocked X10 Programs*, 2013, 11 p. , http://hal.inria.fr/hal-00907723

References in notes

[14] 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

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

[16] 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

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

[18] F. DE DINECHIN, C. KLEIN, B. PASCA. *Generating High-Performance Custom Floating-Point Pipelines*, in "Field Programmable Logic and Applications", IEEE, August 2009, http://prunel.ccsd.cnrs.fr/ensl-00379154/

[19] B. DUPONT DE DINECHIN, C. MONAT, F. RASTELLO. *Parallel Execution of Saturated Reductions*, in "Workshop on Signal Processing Systems (SIPS'01)", IEEE Computer Society Press, 2001, pp. 373-384

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

[21] P. FEAUTRIER. *Bernstein's Conditions*, in "Encyclopedia of Parallel Programming", D. PADUA (editor), Springer, 2011

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

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

[24] 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^{o} 4, 8:86 p.

[25] A. FRABOULET, K. GODARY, A. MIGNOTTE. *Loop Fusion for Memory Space Optimization*, in "International Symposium on System Synthesis (ISSS'01)", Montréal, Canada, IEEE Press, October 2001, pp. 95–100

[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] J.-W. HONG, H. T. KUNG. *I/O Complexity: The Red-Blue Pebble Game*, in "13th Annual ACM Symposium on Theory of Computing (STOC'81)", ACM, 1981, pp. 326–333

[28] R. JOHNSON, M. SCHLANSKER. *Analysis Techniques for Predicated Code*, in "29th Annual ACM/IEEE International Symposium on Microarchitecture (MICRO-29)", Paris, France, IEEE Computer Society, 1996, pp. 100–113

[29] J. LE GUEN, C. GUILLON, F. RASTELLO. *MinIR, a Minimalistic Intermediate Representation*, in "Workshop on Intermediate Representations (WIR'11), held with CGO'11", Chamonix, F. BOUCHEZ, S. HACK, E. VISSER (editors), April 2011, pp. 5-12

[30] A. STOUTCHININ, F. DE FERRIÈRE. *Efficient Static Single Assignment Form for Predication*, in "34th Annual ACM/IEEE International Symposium on Microarchitecture (MICRO-34)", Austin, Texas, IEEE Computer Society, 2001, pp. 172–181

[31] 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

[32] 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