IN PARTNERSHIP WITH:
CNRS

Université Claude Bernard (Lyon 1)

Ecole normale supérieure de Lyon

Activity Report 2012

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 and Compiling

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.

The goal of Compsys is to develop compilation techniques, more precisely 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 for high-level synthesis of hardware accelerators. Recent activities also include a shift towards dynamic compilation and compilation for GPUs and multicores. 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: January 01, 2004 .

1. Members

Research Scientists
Christophe Alias [Researcher (CR Inria)]
Alain Darte [Senior Researcher (DR CNRS), Team Leader, HdR]
Fabrice Rastello [Researcher (CR Inria), HdR]

Faculty Members
Paul Feautrier [Professor ENS-Lyon, emeritus, HdR]
Laure Gonnord [Associate Professor (Lille University), external collaborator, part-time]

Engineer
Alexandru Plesco [ITI (transfer and innovation engineer), Oct. 2010-...]

PhD Students
Quentin Colombet [Inria, contract Nano2012 Mediacom, Jan. 2010-Sep. 2012]
Guillaume Ioss [ENS-Lyon and Colorado State University, Sep. 2011-...]
Alexandre Isoard [ENS-Lyon, Sep. 2012-...]

Administrative Assistant
Laetitia Lecot [Inria, part-time]

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 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 for 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 evaluated again by Inria in Spring 2012, positively. The move in 2013 of Fabrice Rastello to Grenoble, whose goal is to expand the activities of Compsys in the context of Giant, a R&D technology center with several industrial and academic actors, has not been validated by all concerned institutions (yet). As a consequence, Compsys II has not been formally prolonged yet into Compsys III. This is why the new research directions of Compsys III, presented in the 2012 evaluation report – mainly a shift towards dynamic compilation and the compilation of streaming programming, with an even stronger focus on polyhedral optimizations – are not described in this 2012 annual report. They should appear in the 2013 report.

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 2011 for details. Finally, Section 2.5 highlights the main novelties of the past year, i.e., 2012.

2.2. General Presentation

Classically, an embedded computer is a digital system that is part of a larger system and that is not directly accessible to the user. Examples are appliances like phones, TV sets, washing machines, game platforms, or even larger systems like radars and sonars. In particular, this computer is not programmable in the usual way. Its program, if it exists, is supplied as part of the manufacturing process and is seldom (or never) modified thereafter. As the embedded systems market grows and evolves, this view of embedded systems is becoming obsolete and tends to be too restrictive. Many aspects of general-purpose computers apply to modern embedded platforms. Nevertheless, embedded systems remain characterized by a set of specialized application domains, rigid constraints (cost, power, efficiency, heterogeneity), and its market structure. The term *embedded system* has been used for naming a wide variety of objects. More precisely, there are two categories of so-called *embedded systems*: a) control-oriented and hard real-time embedded systems (automotive, plant control, airplanes, etc.); b) compute-intensive embedded systems (signal processing, multi-media, stream processing) processing large data sets with parallel and/or pipelined execution. Compsys is primarily concerned with this second type of embedded systems, now referred to as *embedded computing systems*.

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

A specific aspect of embedded computing systems is the use of various kinds of processors, with many particularities (instruction sets, registers, data and instruction caches, now multiple cores) and constraints (code size, performance, storage). The development of *compilers* is crucial for this industry, as selling a platform without its programming environment and compiler would not be acceptable. To cope with such a range of different processors, the development of robust, generic (retargetable), though efficient compilers is mandatory. Unlike standard compilers for general-purpose processors, compilers for embedded processors 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 remains external collaborator of the team.

All present and past members of Compsys have a background in automatic parallelization and high-level program 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.

It may be worth to quote Bob Rau and his colleagues (IEEE Computer, sept. 2002):

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

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

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

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

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

2.3. Summary of Compsys I Achievements

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

- The development of a strong collaboration with the compilation group at STMicroelectronics, with important results in aggressive optimizations for instruction cache and register allocation.

- New results on the foundation of high-level program transformations, including scheduling techniques for process networks and a general technique for array contraction (memory reuse) based on the theory of lattices.
- Many original contributions with partners closer to hardware constraints, including CEA, related to SoC simulation, hardware/software interfaces, power models, and simulators.

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

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

2.4. Quick view of Compsys II Achievements

So far, 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 2011.

2.5. Highlights of the Year

For 2012, from the point of view of organization, funding, collaborations, the main points to highlight are the following:

- 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.
- Compsys prepared the installation in 2013 of Fabrice Rastello in the Giant center (Grenoble) with two PhD students and one post-doc, as a second component of Compsys. As already mentioned, this new organization is not fully validated yet.
- Compsys started a new industrial collaboration with Kalray, a multi-core french company, and the Inria team Parkas, through the ManyCoreLabs project coordinated by Kalray. The research activities are linked to compilation for the Kalray platform, in particular back-end code optimizations and compilation related to stream computing.
- Compsys obtained some important funding, mainly from the MI-LYON LaBex, to organize in Lyon a thematic quarter on compilation, languages, and architectures in 2013.

From a scientific point of view, the following points can be highlighted:

- Compsys finalized the developments in static single assignment (SSA) and register allocation, leading to the PhD defense of Quentin Colombet [1] and the habilitation of Fabrice Rastello [2].
- In high-level synthesis (HLS), the research and development efforts within the incubated start-up Zettice have been pursued and Zettice may become a full start-up in 2013.
- Compsys obtained several results in program analysis for parametric communication optimizations, scalable program termination, and dependence analysis for the X10 language.

For a detailed description of these new scientific results, see Section 6 “New Results”.

3. Scientific Foundations

3.1. Introduction

The embedded system design community is facing two challenges:

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

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

All of this dictates that modern embedded systems retain some form of programmability. For increased designer productivity and reduced time-to-market, programming must be done in some high-level language, with appropriate tools for compilation, run-time support, and debugging. This does not mean 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.

3.2. Back-End Code Optimizations for Embedded Processors

Participants: Quentin Colombet, Alain Darte, Fabrice Rastello.

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 [20] 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 [27], 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 [29] 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. Program Analysis and Transformations for High-Level Synthesis

Participants: Christophe Alias, Alain Darte, Paul Feautrier, Laure Gonnord [Compsys/LIFL], Alexandru Plesco [Compsys/Zettice].

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 [21] and to process networks [30], [31].

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 [22]. 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 [24], techniques for data allocation in scratch-pad memory, or techniques for folding multi-dimensional arrays [19]. 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) than 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

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 now concern two activities only: a) the development of tools linked to polyhedra and loop/array transformations, b) the development of algorithms within the back-end compiler of STMicroelectronics.

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.

The other activity concerns the developments within the compiler of STMicroelectronics. These are not stand-alone tools, which could be used externally, but algorithms and data structures implemented inside the LAO back-end compiler, year after year, with the help of STMicroelectronics colleagues. As these are also important developments, it is worth mentioning them in this section. They are also completed by important efforts for integration and evaluation within the complete STMicroelectronics toolchain. They concern exact (ILP-based) methods, algorithms for aggressive optimizations, techniques for just-in-time compilation, and for improving the design of the compiler.

5.2. Pip

Participants: Cédric Bastoul [MCF, IUT d'Orsay], 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 [17], 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). 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 [17].

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 http://www.prism.uvsq.fr/~bem/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 [19] 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 will be transferred to the (incubated) start-up Zettice, initiated by Alexandru Plesco.
- Bee is 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. 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.

Chuba has been implemented by Christophe Alias, using the compiler infrastructure PoCo. 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. IceBuilder

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

IceBuilder is the HLS tool to be transferred in the start-up Zettice. It is a compiler, whose input is a C program annotated with pragmas, and whose output is an equivalent hardware description as synthesizable VHDL. Also, IceBuilder produces a non-synthesizable SystemC description for debugging purpose. As for any compiler, IceBuilder consists into two steps: (i) a front-end, which generates an intermediate representation from the C program, and (ii) a back-end, which translates the intermediate representation into hardware. The intermediate representation of IceBuilder is a data-aware process network (DPN) (see Section 6.3). The front-end does most of the high-level optimizations (communication pipelining, buffer sizing, datapath pipeline scheduling), which are explicitly represented in the DPN. The front-end is implemented as a separate tool, Dcc, so as to be reused with different targets, for instance GPGPUs. Then, the back-end generates the hardware implementation of the DPN. It produces and connects the required buffers, multiplexors, demultiplexors, synchronization channels, finite-state machines, and datapaths.

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

5.9. Dcc

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

Dcc is the front-end of the IceBuilder tool. Dcc takes as input a C program annotated with pragmas and produces an optimized data-aware process network (DPN). To do so, Dcc reuses most of the analysis implemented in PoCo (dataflow analysis and control generation), Chuba (communication pipelining), Cl@k and Bee (buffer sizing). Dcc and DPNs are very critical parts of IceBuilder and will require a patent before any publication.

Dcc represents more than 2500 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, 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), 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 to prove program termination.

5.11. RanK

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

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 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, 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. It uses the libraries Piplib and Polylib.

RanK has been implemented by Christophe Alias, using the compiler infrastructure PoCo. It represents more than 3000 lines of C++.

5.12. SToP

Participants: Christophe Alias, Guillaume Andrieu [LIFL], Laure Gonnord [Compsys/LIFL].

SToP (Scalable Termination of Programs) is the implementation of the modular termination technique presented in Section 6.7. 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. 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++.

5.13. 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). It uses PIP 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.

5.14. 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 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 more recent activities, which concern 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.

5.15. 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.

5.16. 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 (aka 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. Open64/Path64 emitter for TireX and its LAO back-end are available at https://compilation.ens-lyon.fr/. MinIR was presented at WIR'11 [28].

6. New Results

6.1. Enhancing the Compilation of Synchronous Data-Flow Languages

Participants: Paul Feautrier, Abdoulaye Gamatié [LIFL], Laure Gonnord [Compsys/LIFL].

In [25] a new (light) numerical-Boolean abstraction was proposed for an efficient static analysis of synchronous programs that describe multi-clock embedded systems in the language Signal. In this abstraction, relations between clocks and numerical variables are modeled by Boolean-affine formulas. These formulas can easily be extracted from the program text. From the results of a satisfiability test of these formulas, clock properties can be deduced, which, when submitted to the Signal compiler, may improve the resulting target program.

In collaboration with Abdoulaye Gamatié, we proposed an extension of the previous approach to modular programs. This extension necessitates the use of an extended SMT (satisfiability modulo theory) solver – able for instance to deal with quantifier elimination – which has been implemented by Paul Feautrier by reusing some of the Syntol tools. This work is still unpublished but will be soon submitted to a journal.

6.2. Dataflow Analysis of Polyhedral X10 Programs

Participants: Paul Feautrier, Sanjay Rajopadhye [Colorado State Universty], Vijay Saraswat [IBM Research], Tomofumi Yuki [Colorado State University].

X10 is a recent parallel language, developed by IBM Research, whose aim is to increase programmers productivity. It is a descendant of Java and it includes several new parallel constructs, such as `async` and `finish`, which generalize `fork` and `join`, clocks, which generalize barriers, and `at`, which enables the remote execution of program fragments. X10 programs are guaranteed to be deadlock-free, but may exhibit non-deterministic behaviors or *races*.

We have devised a verifier for the `async/finish` fragment of X10 and polyhedral programs. The approach consists in computing the source of each array access. In the presence of parallel constructs, the sequencing predicate is no longer a total order and a read may have several sources, which indicates a race. A proof-of-concept tool has been implemented. This work will be presented at the next *Principles and Practice of Parallel Programming* conference (PPoPP'13 in Shenzen, China) [13].

6.3. Data-Aware Process Networks

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

New techniques were introduced to generate and compile optimized data-aware process networks, from a C program annotated with pragmas (see Section 5.9 and the software tool Dcc). These techniques are essential for the Zettice start-up and are not made publicly available for the moment.

6.4. Optimizing Remote Accesses for HLS

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

Some data- and compute-intensive applications can be accelerated by offloading portions of codes to platforms such as GPGPUs or FPGAs. However, to get high performance for these kernels, it is mandatory to restructure the application, to generate adequate communication mechanisms for the transfer of remote data, and to make good usage of the memory bandwidth. In the context of the high-level synthesis (HLS), from a C program, of hardware accelerators on FPGA, we showed how to automatically generate optimized remote accesses for an accelerator communicating to an external DDR memory. Loop tiling is used to enable block communications, suitable for DDR memories. Pipelined communication processes are generated to overlap communications and computations, thereby hiding some latencies, in a way similar to double buffering. Finally, not only intra-tile but also inter-tile data reuse is exploited to avoid remote accesses when data are already available in the local memory.

We showed how to generate the sets of data to be read from (resp. written to) the external memory just before (resp. after) each tile so as to reduce communications and reuse data as much as possible in the accelerator. The main difficulty arises when some data may be (re)defined in the accelerator and should be kept locally. We proposed an automatic optimized code generation scheme, entirely at source-level, i.e., in C, that allows us to compile all the necessary glue (the communication processes) with the same HLS tool as for the computation kernel. Our method, implemented in the tool Chuba (see Section 5.7) uses advanced polyhedral techniques for program analysis and transformation. Experiments with Altera HLS tools demonstrate how to use our techniques to efficiently map C kernels to FPGA.

This work, astride two different fields (compilation for high-performance computing and high-level synthesis) turned out to be very difficult to publish. It was finally accepted at PPoPP'12 [6], but only as a short paper (2 pages). We requested to retain the copyright of this work to be able to publish a longer version. It was accepted at the IMPACT'12 workshop [7], which makes paper available on the web, but with no copyright. It was finally accepted as a full publication at the DATE'13 conference [8].

6.5. Parametric Inter-Tile Reuse for Kernel Offloading

Participants: Alain Darte, Alexandre Isoard.

The method described in Section 6.4 is not parametric in terms of the tile size, i.e., the tile size needs to be fixed before compiling the program. Furthermore, the size of the required local memory depends on the tile size and is available only after program analysis. As a result, to select the tile size with respect to the size of the local memory, the program first needs to be compiled (actually analyzed) for all tile sizes. A parametric program analysis would be much more convenient. The situation is even worse to get the runtime performances in terms of the tile size. Indeed, so far, Chuba generates a C code that is generic and cannot be immediately compiled by C2H. A few modifications by hand are still needed, such as inserting the adequate pragmas for C2H, transforming array accesses to linearized addresses with the right base addresses, changing some arrays into non-aliasing pointers so that C2H, whose dependence analyzer and software pipeliner are weak, can generate codes with the right initiation intervals, etc. These changes are minor and systematic and take time when performed by hand. A fully-parametric compilation scheme would be a plus.

The formulation proposed in Section 6.4 is unfortunately quadratic in terms of the tile size, which prevents to parameterize it. Indeed, it relies on parametric linear programming, which works only with a linear use of parameters. As part of the Master internship of Alexandre Isoard, we nevertheless succeeded to design a fully-parametric scheme for inter-tile reuse and buffer size computation. The method is much more involved but is still compatible with approximations. These results have still to be implemented and submitted for publication.

6.6. Semantic Program Transformations

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

Traditionally, a program transformation is considered to be *correct* if each data dependence of the original program is respected. In that case, both original and transformed programs perform *exactly* the same computation. We can relax this condition by expecting both programs to perform the same computation, modulo the *semantic* properties of the operators (e.g., associativity, commutativity). Semantic program transformations extend the traditional corpus of program transformations and can reveal new optimization opportunities.

More specifically, we are interested in *semantic loop tiling*, a special case of loop tiling, where the input arrays are tiled, and the program is restructured to use high-level matrix operations between data tiles, instead of the original scalar operations. Surprisingly, it turns out that in most cases, the semantic tiling is simply obtained by substituting the scalar variables by the tiles (matrices), and the original operators by the corresponding matrix operators (e.g., a/b by MatMul(A,Inv(B))). The approach currently investigated consists in two steps: (i) guess the semantic tiling, and (ii) prove the (semantic) equivalence with the original program.

Our current contribution is an heuristic to check the equivalence of two programs modulo associativity/commutativity so as to achieve the step (ii). The two programs should fit in the polyhedral model but can involve explicit reductions. This work is currently under submission, and is part of the PhD thesis of Guillaume Iooss.

6.7. Modular Termination of Large Programs

Participants: Christophe Alias, Guillaume Andrieu [LIFL], Laure Gonnord [Compsys/LIFL].

Program termination is an essential step in program verification. In [16], we showed how to check the termination of programs whose control can be summarized by an integer interpreted automaton. This was done by computing a *ranking function* (kind of schedule) by means of integer linear programming techniques. This approach, though powerful, clearly lacks scalability and cannot handle large programs.

We overcame this limitation by proceeding into two steps. First, we extract, from the program to be analyzed, the part useful for termination, i.e., the smaller program slice with the same control behavior. Then, we show that proving the termination of the whole program (slice) boils down to prove the termination of small programs, which can be handled by the technique of [16]. Experimental results show that many large programs can be handled this way.

This work was part of the engineer internship of Guillaume Andrieu. Our technique has been implemented in a tool called SToP (see Section 5.12) and presented at the workshop TAPAS'12 [9].

6.8. Lower Bounds for the Inherent Data Locality Properties of Computations

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

Data movement will account for most of the energy as well as execution time on upcoming exascale architectures, including data movement between processors as well as data movement across the memory hierarchy within each processor. Therefore a fundamental characterization of the data access complexity of algorithms is increasingly important.

We addressed the problem of data access or I/O complexity in a two-level memory hierarchy, as studied in the seminal work of Hong and Kung [26]. We developed a novel approach based on graph min-cut for deriving lower bounds on I/O complexity with two significant advantages over the S-partitioning model of Hong and Kung: (1) the approach can be used to develop analytical expressions with tighter lower bounds for I/O, and (2) unlike any previous model, our new lower bound approach can be automated for analyzing an arbitrary computational directed acyclic graph. We show tighter analytically-derived lower bounds as well as very promising experimental results thanks to a prototype tool that implements our fully-automated analysis.

This work has been submitted and is part of an informal collaboration with P. Sadayappan from the University of Columbus (CSU).

6.9. A Polynomial Spilling Heuristic: Layered Allocation

Participants: Albert Cohen [Inria, Parkas], Boubacar Diouf [Université Paris Sud, Parkas], Fabrice Rastello.

Register allocation is subdivided into two sub-problems: first, the *allocation* (or its dual problem the *spilling*) selects the set of variables that will reside in registers (resp. in memory) at each point of the program. Then, the *assignment* or *coloring* picks a specific register where a variable will reside. Building on some properties of the static single assignment form (SSA), it is now possible to decouple the allocation from the assignment. Indeed, the interference graph of a program in SSA form is a chordal graph. In this context, MAXLIVE, the maximal number of variables simultaneously live at a program point, is used during the spilling phase as a criterion to guarantee that the forthcoming assignment will be performed without any spill. If MAXLIVE is lower than or equal to R, the number of available registers, then all the variables will be assigned without any spill. This *decoupled* approach was advocated by Fabri, Appel and George, Darte et al., and others.

Existing spilling heuristics rely on a sufficient condition to guarantee register assignment, and incrementally spill until the condition holds. As we just mentioned, for programs under SSA, the condition is necessary and sufficient: MAXLIVE has to be lower than or equal to R. Incremental spilling decisions to satisfy this condition tend to be overly local and suboptimal. Indeed, incremental spilling itself is NP-complete, and heuristics based upon it trade too much their optimality for polynomiality. In contrast to incremental spilling, we proposed to adopt the symmetric approach: incremental allocation. Intuition for it emerges from two observations allowing for more global spilling decisions:

1. Register allocation is pseudo-polynomial in the number of registers, suggesting a heuristic that solves (optimally) roughly $R/step$ allocation problems on $step$ registers each. The final allocation is the layered composition of the stepwise allocations.
2. Stepwise optimality does not guarantee an overall optimal allocation, but experiments show that it comes very close to optimal, even with $step = 1$. Intuition for this comes from recent work by Diouf et al., observing that allocation decisions tend to be a monotonic function of the number of registers.

This work, which will be presented at CGO'13 [11], proposes a new graph-based allocation heuristic, based on a maximum clique cover formulation to define the profitability of spilling variables. It exploits the pseudo-polynomial complexity in the number of registers of the allocation problem under SSA — as opposed to the symmetric, spilling problem, which remains strongly NP-complete. It addresses the spill-everywhere problem in a decoupled context and also proposes an extension to non-decoupled approaches. It introduces *layered allocation* a new strategy that incrementally allocates variables instead of incrementally spilling variables. The evaluation performed on standard benchmarks shows that this new approach is near-optimal.

6.10. Interaction Between Spilling and Scheduling

Participants: Quentin Colombet, Alain Darte, Fabrice Rastello.

As explained in Section 6.9, it is possible to decouple the register allocation problem in two successive phases: a first *spilling* phase places `load` and `store` instructions so that the register pressure at all program points is small enough, a second *assignment* and *coalescing* phase maps the remaining variables to physical registers and reduces the number of `move` instructions among registers. At CASES'11 [18], we presented a new integer linear programming (ILP) formulation, for load-store architectures, to capture "optimal" spilling in a more accurate and more expressive way than previous approaches. Among other features, we can express SSA ϕ-functions, memory-to-memory copies, and the fact that a value can be stored simultaneously in a register and in memory.

We used this ILP formulation to experimentally analyze the impact of the different heuristic strategies and compare them with optimal solutions. While "optimal" solutions show significant improvements for static spill costs, it turned out that runtime performances were disappointing (if not random). We conducted various experiments to understand this behavior and discovered that the interaction with scheduling is actually higher than expected. Micro-architectural features (e.g., memory latencies that can be hidden by prefetching, bundling that can hide cycles) have to be accounted for in the model, which is never done. These experiments and analysis are described in Chapter 4 of Quentin Colombet's PhD thesis [1].

6.11. Elimination of Parallel Copies Using Code Motion on Data Dependence Graphs

Participants: Florian Brandner, Quentin Colombet.

Traditional approaches to copy elimination during register allocation are based on interference graphs and register coalescing. Variables are represented as nodes in a graph, which are coalesced, if they can be assigned the same register. However, decoupled approaches strive to avoid interference graphs and thus often resort to local recoloring.

A common assumption of existing coalescing and recoloring approaches is that the original ordering of the instructions in the program is not changed. We developed an extension of a local recoloring technique called Parallel Copy Motion. We perform code motion on data dependence graphs in order to eliminate useless copies and reorder instructions, while at the same time a valid register assignment is preserved. Our results show that even after traditional register allocation with coalescing our technique is able to eliminate an additional 3% (up to 9%) of the remaining copies and reduce the weighted costs of register copies by up to 25% for the SPECINT 2000 benchmarks. In comparison to Parallel Copy Motion, our technique removes 11% (up to 20%) more copies and up to 39% more of the copy costs.

These results have been accepted for publication at SAC'12 [10] and, in a longer version, in the journal Computer Languages, Systems, and Structures [5].

7. Bilateral Contracts and Grants with Industry

7.1. Mediacom Project with STMicroelectronics

Participants: Benoit Boissinot, Florian Brandner, Quentin Colombet, Alain Darte, Fabrice Rastello.

This contract has started in September 2009 as part of the funding mechanism Nano2012. This is the continuation of the successful previous project Sceptre with STMicroelectronics, which ended in December 2009. Mediacom concerned both aggressive optimizations and the application of the previously-developed techniques to JIT compilation. The project ended this year. Quentin Colombet, whose PhD was funded by this project, defended his PhD in December 2012 [1].

7.2. Creation of the Zettice Start-Up

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

Following his PhD, Alexandru Plesco initiated a start-up on high-level synthesis for FPGAs, named Zettice, and based on the use and extension of tools/techniques developed in Compsys (for high-level code transformations) and Arénaire (for the development of pipelined operators). The results described in Sections 5.7, 5.8, 5.9, and 6.4 are directly linked to this effort.

The incubation of Zettice is supported by Crealys, the "Région Rhône-Alpes", and Inria: Alexandru Plesco is "ingénieur technologie and innovation" (ITI) since October 2011. Zettice should be created around April 2013. Christophe Alias is in charge of the scientific collaboration between Compsys and Zettice.

7.3. ManyCoreLabs with Kalray

Compsys is part of ManyCoreLabs, an academic/industrial project, coordinated by Kalray, a multi-core french company. The project is funded by a "Investissement d'Avenir"/BGLE ("Briques génériques du logiciel embarqué") grant. The goal of this project is to help the Kalray company, 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 include CEA, Inria (Parkas and Compsys teams), VERIMAG.

8. Partnerships and Cooperations

8.1. National Initiatives

8.1.1. 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" (8kEuros/year, during two years). The goal of this project is to investigate how to introduce real-time constraints in the high-level synthesis workflow.

8.1.2. Inria AEN

Compsys is part of an Inria Large Scale Initiative (AEN: action d'envergure nationale) that regroups eight teams: Camus, Regal, Alf, Runtime, Algorille, Parkas, Dali on "Large scale multicore virtualization for performance scaling and portability".

8.1.3. 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 in Summer 2010, Aussois in Winter 2010, Dinard in Spring 2011, St Hippolyte in Autumn 2011, Rennes in Summer 2012, Lyon/Annecy in Spring 2013).

8.2. International Research Visitors

8.2.1. Visits to International Teams (at least one month)

Paul Feautrier has been invited to spend the month of June 2012 at Colorado State University (CSU), Fort Collins, CO, USA, in prof. Sanjay Rajopadhye's team. The work reported in Section 6.2 and accepted at PPoPP'13 [13] was initiated during this stay. Sanjay Rajopadhye and Tomofumi Yuki, both from CSU, have spent a few days in Paris and Lyon in December 2012. During this visit, we have initiated a sequel to this work, which will handle other parallel features of X10.

8.2.2. Informal Collaborations and Short-Term Visitors

Shorter visits (but at least a week) include exchanges (in both directions) with the groups of S. Rajopadhye (Colorado State University), of P. Sadayappan (Ohio State University), of J. Ramanujam (Louisiana State University), of L.-N. Pouchet (UCLA), all related directly or indirectly to polyhedral code optimizations.

Compsys has also regular contacts with Sebastian Hack (Saarland University, Saarbrücken, Germany), Benoît Dupont de Dinechin (Kalray, Grenoble), Christophe Guillon (STMicroelectronics), Fernando M. Q. Pereira (Federal University of Mina Gerais, Brazil) on back-end code optimizations.

Among french academic researchers, Compsys is particularly linked with people such as Albert Cohen (Inria Parkas team), Steven Derrien (Inria Cairn team), Alain Ketterlin (Inria Camus team), François Irigoin (Ecole des Mines de Paris).

Finally, taking the opportunity of the HdR defense of Fabrice Rastello [2] and the PhD defense of Quentin Colombet [1] on December 7, 2012, a "compilation day" was organized in Lyon on December 6, including talks by K. Pingali (University of Texas, Austin), E. Altman (IBM Yorktown), and V. Sarkar (Rice University).

9. Dissemination

9.1. Scientific Animation

Christophe Alias is a member of the steering committee of IMPACT (international workshop on polyhedral compilation techniques). Christophe Alias, Alain Darte, and Paul Feautrier were members of the program committee of IMPACT'13, which will be held in conjunction with HiPEAC'13.

Laure Gonnord and Fabrice Rastello are the co-animators of the french compilation community, and organized the fifth session in June 2012, in Rennes.

Paul Feautrier is an associate editor of *International Journal of Parallel Programming* and of *Prallel Computing*. Alain Darte is an associate editor of ACM Transactions on Embedded Computing Systems.

Laure Gonnord was a member of the program commitee of TAPAS'12 (Tools for Automatic Program Analysis).

In addition to IMPACT'13, Paul Feautrier is a member of the program committee of ParCo'13 (International Parallel Computing Conference). He has reviewed papers for LCTES, DATE and ACM TECS.

In addition to IMPACT'13, Alain Darte was a member of the program committees of DATE'13 (Design, Automation, and Test in Europe), LCTES'12 (Languages, Compilers, Tools, and Theory for Embedded Systems), IPDPS'13 (International Parallel and Distributed Processing Symposium). He has reviewed papers for the DATE'13, LCTES'12, EMSOFT'12, CGO'13 conferences and for the following journals: TSI, Parallel Computing, ACM TECS.

9.2. Workshops and Invited Talks

Alain Darte and Quentin Colombet participated to the 3-days workshop CPC'12 (16th Workshop on Compilers for Parallel Computing), in Padova, Italy, January 11-13, 2012. giving two talks "Kernel offloading with optimized remote accesses" and "Spilling in the light of SSA".

Alain Darte was invited by Jürgen Teich (University of Erlangen-Nuremberg) to give a talk (May 18, 2012) on "Optimizing remote accesses for offloaded kernels, with application to HLS for FPGA", as part of their "invasive computing" project.

Alain Darte was invited to participate to the 4-days workshop ScalPerf'12 (Scalable Approaches to High Performance and High Productivity Computing), in Bertinoro, Italy, September 23-27, 2012. He gave a talk on "Parametric tiling with pipelining or how to automate double-buffering execution style".

9.3. Teaching - Supervision - Juries

9.3.1. Teaching

Licence: Christophe Alias gave a L3 course on "Introduction to compilation" at ENSI Bourges and a L2 lab on "Architecture des ordinateurs" at Université Lyon 1. Laure Gonnord has a full teaching activity in Polytech'Lille where she teaches algorithmics, compilation, and basics of hardware.

Master 1: Christophe Alias gave a Master 1 course on "Compilation" at ENS-Lyon.

Master 2: Alain Darte gave a full Master 2 course on "Advanced compilation and program optimizations" at ENS-Lyon.

Master 1 and 2: Christophe Alias is organizer of the Winter Master School "Vérification et certification du logiciel" that was held in January 2012 at ENS-Lyon, as part of the Master of ENS-Lyon.

Internships: Laure Gonnord and Christophe Alias co-supervised the undergraduate internship of Jean-Marie Vincenti, from the Engineering School Polytech'Lille. This internship lasted two months, from June 2012 to July 2012. The topic was about graphical visualization and step-by-step debugging of communicating processes described with an execution trace. Alain Darte advised the M2 internship of Alexandre Isoard on the parametric tiled kernel offloading (see Section 6.5).

9.3.2. Supervision

PhD in progress: Guillaume Iooss, "Semantic program transformations in the polyhedral model", started in September 2011, at ENS-Lyon and Colorado State University. Advisors: Christophe Alias, Alain Darte, Sanjay Rajopadhye (Colorado State University).

PhD in progress: Alexandre Isoard, "Optimization of remote communications", started in September 2012, at ENS-Lyon. Advisors: Christophe Alias, Alain Darte.

PhD: Quentin Colombet, "Decoupled (SSA-based) Register Allocators: from Theory to Practice, Coping with Just-In-Time Compilation and Embedded Processors Constraints" [1], ENS-Lyon, December 7, 2012. Advisors: Alain Darte, Fabrice Rastello.

HdR: Fabrice Rastello, "On Sparse Intermediate Representations: Some Structural Properties and Applications to Just In Time Compilation" [2], ENS-Lyon, December 7, 2012.

9.3.3. Juries

Fabrice Rastello was member of the jury for Artur Pietrek's PhD thesis (Verimag, Université Joseph Fourier), defended December 2, 2012.

Paul Feautrier is one of the referees for Stephane Mancini's HdR (TIMA, Grenoble), to be defended February 19, 2013.

Alain Darte was reviewer of Mehdi Amini's PhD thesis (Ecole nationale supérieure des Mines de Paris), defended December 13, 2012, and entitled "Source-to-Source Automatic Program Transformations for GPU-like Hardware Accelerators".

As co-advisors, Alain Darte and Fabrice Rastello were of course part of the jury of Quentin Colombet's PhD thesis [1], defended December 7, 2012, at ENS-Lyon, and entitled "Decoupled (SSA-based) Register Allocators: From Theory to Practice, Coping with Just-In-Time Compilation and Embedded Processors Constraints".

Alain Darte was part of the jury of Fabrice Rastello's HdR thesis [2], defended December 7, 2012, at ENS-Lyon, and entitled "On Sparse Intermediate Representations: Some Structural Properties and Applications to Just-In-Time Compilation".

10. Bibliography

Publications of the year

Doctoral Dissertations and Habilitation Theses

[1] Q. COLOMBET. *Decoupled (SSA-based) Register Allocators: From Theory to Practice, Coping with Just-In-Time Compilation and Embedded Processors Constraints*, École normale supérieure de Lyon, December 2012, 224, http://hal.inria.fr/hal-00761572.

[2] F. RASTELLO. *On Sparse Intermediate Representations: Some Structural Properties and Applications to Just-In-Time Compilation*, École normale supérieure de Lyon, December 2012, 154, Habilitation à Diriger des Recherches, Travaux universitaires, http://hal.inria.fr/hal-00761555.

Articles in International Peer-Reviewed Journals

[3] C. ALIAS, B. PASCA, A. PLESCO. *FPGA-Specific Synthesis of Loop Nests with Pipelined Computational Cores*, in "Microprocessors and Microsystems - Embedded Hardware Design", 2012, vol. 36, n^{o} 8, p. 606-619, http://hal.inria.fr/hal-00761515.

[4] B. BOISSINOT, P. BRISK, A. DARTE, F. RASTELLO. *SSI Properties Revisited*, in "ACM Transactions on Embedded Computing Systems", 2012, vol. 11S, n^{o} 1, Article 21, 23 pages, http://hal.inria.fr/hal-00761505.

[5] F. BRANDNER, Q. COLOMBET. *Elimination of parallel copies using code motion on data dependence graphs*, in "Computer Languages, Systems & Structures", April 2013, vol. 39, n^{o} 1, p. 25 - 47, To appear [*DOI :* 10.1016/J.CL.2012.09.001], http://hal.inria.fr/hal-00768781.

International Conferences with Proceedings

[6] C. ALIAS, A. DARTE, A. PLESCO. *Optimizing Remote Accesses for Offloaded Kernels: Application to High-Level Synthesis for FPGA*, in "17th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP'12)", New Orleans, United States, IEEE Computer Society, 2012, p. 285–286, Short paper, http://hal.inria.fr/hal-00761473.

[7] C. ALIAS, A. DARTE, A. PLESCO. *Optimizing Remote Accesses for Offloaded Kernels: Application to High-Level Synthesis for FPGA*, in "2nd International Workshop on Polyhedral Compilation Techniques (IMPACT'12), held with HIPEAC'12", Paris, France, 2012, PPoPP'12 extended version, http://hal.inria.fr/hal-00761477.

[8] 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, To appear, http://hal.inria.fr/hal-00761533.

[9] 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.

[10] F. BRANDNER, Q. COLOMBET. *Copy Elimination on Data Dependence Graphs*, in "27th Annual ACM Symposium on Applied Computing (SAC'12)", Trento, Italy, ACM Press, 2012, p. 1916-1918, http://hal.inria.fr/hal-00761499.

[11] B. DIOUF, A. COHEN, F. RASTELLO. *A Polynomial Spilling Heuristic: Layered Allocation*, in "International Symposium on Code Generation and Optimization (CGO'13)", Shenzhen, China, IEEE Computer Society Press, 2013, To appear, http://hal.inria.fr/hal-00761528.

[12] P. FEAUTRIER. *Approximating the Transitive Closure of a Boolean-Affine Relation*, in "2nd International Workshop on Polyhedral Compilation Techniques (IMPACT'12), held with HIPEAC'12", Paris, France, 2012, http://hal.inria.fr/hal-00761491.

[13] 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, To appear, http://hal.inria.fr/hal-00761537.

Research Reports

[14] G. ANDRIEU, C. ALIAS, L. GONNORD. *Modular Termination of C Programs*, Inria, December 2012, n^{o} 8166, http://hal.inria.fr/hal-00761521.

[15] B. DIOUF, A. COHEN, F. RASTELLO. *A Polynomial Spilling Heuristic: Layered Allocation*, Inria, July 2012, n^{o} RR-8007, 23, http://hal.inria.fr/hal-00713693.

References in notes

[16] 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, p. 117-133.

[17] P. BOULET, P. FEAUTRIER. *Scanning Polyhedra without DO loops*, in "PACT'98", October 1998.

[18] Q. COLOMBET, F. BRANDNER, A. DARTE. *Studying Optimal Spilling in the Light of SSA*, in "International Conference on Compilers, Architectures, and Synthesis of Embedded Systems (CASES'11)", Taipei, Taiwan, IEEE Computer Society, October 2011.

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

[20] 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, p. 373-384.

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

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

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

[24] A. FRABOULET, K. GODARY, A. MIGNOTTE. *Loop Fusion for Memory Space Optimization*, in "IEEE International Symposium on System Synthesis", Montréal, Canada, IEEE Press, October 2001, p. 95–100.

[25] 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.

[26] 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, p. 326–333.

[27] R. JOHNSON, M. SCHLANSKER. *Analysis of Predicated Code*, in "International Workshop on Microprogramming and Microarchitecture (Micro-29)", 1996.

[28] 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, p. 5-12.

[29] A. STOUTCHININ, F. DE FERRIÈRE. *Efficient Static Single Assignment Form for Predication*, in "International Symposium on Microarchitecture", ACM SIGMICRO and IEEE Computer Society TC-MICRO, 2001.

[30] 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, p. 220–229.

[31] 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.