IN PARTNERSHIP WITH:
CNRS

Université Claude Bernard (Lyon 1)

Ecole normale supérieure de Lyon

Activity Report 2011

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: Formal Methods, Compiling, Optimization, Embedded Systems, Hardware Accelerators, Processors

*Compsys is a common research project-team, located at Ecole normale supérieure de Lyon (*ENS-L*yon). It exists since January 2002 as part of Laboratoire de l'Informatique du Parallélisme (Lip,* UMR CNRS ENS-L*yon* UCB-L*yon 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 will be evaluated 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. Compsys focuses 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. The main characteristic of Compsys is its focus on combinatorial optimization problems (graph algorithms, linear programming, polyhedra) 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.

1. Members

Research Scientists

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

Faculty Members

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

Technical Staff

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

PhD Student

Quentin Colombet [Inria, contract Nano2012 Mediacom, Jan. 2010-...]

Post-Doctoral Fellow

Florian Brandner [Inria, contract Nano2012 Mediacom, Dec. 2009-Oct. 2011]

Administrative Assistant

Laetitia Lecot [Inria, part-time]

2. Overall Objectives

2.1. Introduction

Keywords: compilation, automatic generation of VLSI chips, code optimization, scheduling, parallelism, memory optimization, FPGA platforms, VLIW processors, DSP, regular computations, linear programming, tools for polyhedra and lattices.

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, and 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+. It will continue with updated research directions. Section 2.2 defines the general context of the team's activities. Section 2.3 presents the research objectives targeted during the first 4 years (until 2007), the main achievements over this period, and the new research directions that Compsys will follow in the coming years. The last section, Section 2.4, highlights the main achievements of 2011. For 2008, 2009, 2010, see the previous reports.

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) 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. Our objective is 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 in Alchemy and Alf (previously known as Caps), 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. Highlights of the First 4-Years Period

The Compsys team has been evaluated by Inria in April 2007. The evaluation, conducted by Erik Hagersted (Uppsala University), Vinod Kathail (Synfora, inc), J. (Ram) Ramanujam (Baton Rouge University) was positive. Compsys will thus continue for 4 years as an Inria project-team 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, 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 the size reduction of Compsys (from 5 permanent researchers to 3 in 2008, then 4 again in 2009), the team now focuses 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. Highlights

The year 2011 was marked by strong financial difficulties due to the unilateral decision of the government to stop all Nano2012 fundings. For Compsys, involved in the Mediacom and S2S4HLS projects, this has led to a non-anticipated budget cut of about 60% (excluding salaries). The help of Inria to support the salary of Florian Brandner (post-doc/engineer) and pay some registration fees (in particular CGO'11) was crucial. The research activities in Mediacom still continue, but in a restricted form, until the end of Quentin Colombet's PhD. However, Compsys had to stop its participation to S2S4HLS.

Compsys continued its activities on static single assignment (SSA) and register allocation, as well as on high-level synthesis (HLS) for FPGA. The main achievements in 2011 are:

- The design of a tree-scan allocator was continued and two related contributions were published at SCOPES'11 and CASES'11.
- Our new algorithm for liveness analysis under SSA and a comparison with existing methods were finalized and published at APLAS'11.
- An analysis, based on an integer linear programming formulation, of "optimal spilling" was made with full experiments and published at CASES'11.
- The automatic generation of double-buffered pipelined versions of computation kernels for FPGA was improved and will be presented at PPoPP'12. A simplifier for Boolean affine formulas was designed that should improve the code generation part.
- Alexandru Plesco, following his PhD, initiated a project of start-up, Zettice, supported by Inria and ENS-Lyon, combining the experience of Compsys on compilation and HLS, and the expertise of Arénaire on floating-point pipelined operators for FPGA. A publication at ARC'11 illustrates this effort.

Also, in 2011, Compsys was very active in the organization of important events for our scientific community:

- Fabrice Rastello, after re-activating the french community in compilation, was very involved in the organization of the main international conference in code generation (CGO'11), in Chamonix, and the organization and advertising of its workshops.
- Christophe Alias was the main organizer of IMPACT'11 (international workshop on polyhedral compilation techniques), held in conjunction with CGO'11. This workshop is the very first international event on this topic. Laure Gonnord was co-organizer of the workshop ACCA'11 (analyze to compile, compile to analyze), also part of CGO'11.

BEST PAPER AWARD :
[14] **22nd IEEE International Conference on Application-specific Systems, Architectures and Processors (ASAP'11)**. F. DE DINECHIN, J.-M. MULLER, B. PASCA, A. PLESCO.

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 term 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 commonplace 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 placing & routing 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. Beside, 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: Florian Brandner, 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 leads 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 [33] 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 find 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. We want 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 want to investigate 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 [39], which interferes with all optimization phases, as the SSA form does. Many classical algorithms become inefficient for predicated code. This is especially surprising, since, besides 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 first to adapt classical algorithms to predicated code [40], but 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 show that it is possible to exploit SSA 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 offers 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 not been fully studied or developed yet. The objective of our new contract with STMicroelectronics, called Mediacom, is to address them. More generally, we want 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.

We want to consider more code optimizations and more architectural features, such as registers with aliasing, predication, and, possibly in a longer term, vectorization/parallelization again.

3.3. Program Analysis and Transformations for High-Level Synthesis

Participants: Christophe Alias, Alain Darte, Paul Feautrier, Laure Gonnord, Alexandru Plesco.

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 report 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 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. Beside, 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 sometime compositions of the basic models. For instance, a systolic array follows the streaming model, and sometime 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 [34] and to process networks [41], [42].

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 [35]. 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 [36]. 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 [38], techniques for data allocation in scratch-pad memory, or techniques for folding multi-dimensional arrays [32]. 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, including VHDL but not only, 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? Our partnership with STMicroelectronics (synthesis) should help us answer such a question, considering both industrial applications and industrial HLS tools.

4. Application Domains

4.1. Application Domains

Keywords: embedded computing systems, compilation, high-level synthesis, compilation, program optimizations.

The previous sections describe 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 methods (ILP-based), 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. Pip 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

Participants: Hadda Cherroun [Former PhD student in Compsys], 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 [31], 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. See previous reports for more details on the underlying theory. 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:

- 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.
- Extended language of pragmas to feed the source code with compilation directives (a schedule, for example).
- Symbolic layer on polyhedral libraries POLYLIB (set operations on polyhedra) and PIPLIB (parameterized ILP). This feature simplifies drastically the developer task.
- Dependence analysis (polyhedral dependence graph, array dataflow analysis), array region analysis, array liveness analysis.
- C and VHDL code generation based on the ideas of P. Boulet and P. Feautrier [31].

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 [32] 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.

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. 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.

5.9. RanK

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

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 5.8), representing the control structure of the program to check. 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++.

5.10. 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. It uses PIP for testing the feasibility – or unfeasibility – of a conjunction of affine inequality.

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 syntax.

5.11. 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. Since then, new developments are constantly done, in particular by Florent Bouchez, advised by Alain Darte and Fabrice Rastello, as part of his master internship and PhD thesis. 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 new ILP-based solution (see Section 6.2). It also includes an up-to-date graph-based register coalescing. Finally, since all these optimizations take place under SSA form, it includes also a mechanism for going out of colored-SSA (register-allocated SSA) form that can handle critical edges and does further optimizations (see for example Section 6.3).

5.12. LAO Developments in JIT Compilation

Participants: Benoit Boissinot, Florian Brandner, 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 as described in Section 6.6. Recent efforts (see Section 6.4) 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.

5.13. 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. Besides 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 consumed 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 [17].

6. New Results

6.1. Introduction

This section presents the results obtained by Compsys in 2011. For clarity, some earlier results are also recalled, when they were continued or extended during the year 2011.

6.2. Studying Optimal Spilling in the Light of SSA

Participants: Florian Brandner, Quentin Colombet, Alain Darte.

Recent developments in register allocation, mostly linked to static single assignment (SSA) form, have shown that it is possible to decouple the 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. We focused on the first phase, for which many open questions remained: in particular, we studied the notion of optimal spilling (what can be expressed?) and the impact of SSA form (does it help?).

To identify the important features for optimal spilling on load-store architectures, we developed a new integer linear programming formulation, more accurate and expressive than previous approaches. Among other features, we can express SSA ϕ-functions, memory-to-memory copies, and the fact that a value can be stored simultaneously in a register and in memory. We implemented this formulation in LAO and analyzed in details the static and dynamic results obtained for the SPEC INT 2000 and EEMBC 1.1 benchmarks. We can draw, among others, the following conclusions: a) rematerialization is extremely important, b) SSA complicates the formulation of optimal spilling, especially because of memory coalescing when the code is not in CSSA, c) micro-architectural features are significant and thus have to be accounted for, d) significant savings can be obtained in terms of static spill costs, cache miss rates, and dynamic instruction counts.

This work has been presented at the CASES'11 conference [8].

6.3. Copy Elimination on Data Dependence Graphs

Participants: Florian Brandner, Quentin Colombet.

Register allocation recently regained much interest due to new decoupled strategies that split the problem into separate phases: spilling, register assignment, and copy elimination.

A common assumption of existing copy elimination approaches is that the original ordering of the instructions in the program is not changed. We worked on an extension of a local recoloring technique that we developed earlier, called Parallel Copy Motion [30]. 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.

This work will be presented at the conference SAC'12 [6].

6.4. Graph-Coloring and Treescan Register Allocation Using Repairing

Participants: Quentin Colombet, Benoit Boissinot, Philip Brisk [University of California, Riverside], Sebastian Hack [Saarland University], Fabrice Rastello.

Graph coloring and linear scan are two appealing techniques for register allocation as the underlying formalism are extremely clean and simple. Our previous work advocated the use of a decoupled approach that first lowers the register pressure by spilling variables, then performs live-range splitting, coalescing, and coloring in a separate phase. This enables the design of simpler, cleaner, and more efficient register allocators.

In this context, we introduced a new and more general approach to deal with register constraints. This approach, called repairing, does not require any preliminary live-range splitting and does not introduce additional spill code. It ignores register constraints during coloring/coalescing and repairs the violated constraints afterwards. We applied this method to develop both a graph-based and a scan-based decoupled approach: one based on the iterated register coalescer (IRC) and the other on a scan algorithm (the treescan) that uses static single assignment (SSA) properties.

Our experimental evaluation shows that, for the graph-based approach, we reduced the number of vertices (edges) in the interference graph by 26% (33%) without compromising the quality of the generated code. The treescan algorithm improved the compile time of the allocation process by $6.97\times$ over IRC while providing comparable results for the quality of the generated code.

This work was part of a collaboration with the Saarland University and the University of California, Riverside. It has been presented at the conference CASES'11 [7].

6.5. Decoupled Graph-Coloring Register Allocation with Hierarchical Aliasing

Participants: Andre Tavares [UFMG, Brazil], Quentin Colombet, Mariza Bigonha [UFMG, Brazil], Christophe Guillon [STMicroelectronics], Fernando Pereira [UFMG, Brazil], Fabrice Rastello.

Decoupling spilling from register assignment, as mentioned in previous sections, has the main advantage of simplifying the implementation of register allocators. However, the decoupled model faces many problems when dealing with register aliasing, a phenomenon typical in architectures usually seen in embedded systems, such as ARM.

We introduced the semi-elementary form, a program representation that brings decoupled register allocation to architectures with register aliasing. The semi-elementary form is much smaller than program representations used by previous decoupled solutions, which leads to register allocators that perform better in terms of time and space. Furthermore, this representation reduces the number of copies that traditional allocators insert into assembly programs. We have empirically validated our results by showing that how our representation improves two well-known graph-coloring-based allocators, namely the iterated register coalescer (IRC) and Bouchez et al.'s brute force (BF) method, both augmented with Smith et al. extensions to handle aliasing. Running our techniques on SPEC CPU 2000, we have reduced the number of nodes in the interference graphs by a factor of 4 to 5, hence speeding-up the allocation time by a factor of 3 to 5. In addition, the semi-elementary form reduces by 8% the number of copies that IRC leaves uncoalesced.

This work is part of a collaboration with the Federal University of Minas Gerais. It has been presented at SCOPES'11 [13].

6.6. A Non-Iterative Data-Flow Algorithm for Computing Liveness Sets in Strict SSA Programs

Participants: Benoit Boissinot, Florian Brandner, Alain Darte, Benoît Dupont de Dinechin [Kalray], Fabrice Rastello.

We revisited the problem of computing liveness sets, i.e., the sets of variables live-in and live-out of basic blocks, for programs in strict static single assignment (SSA). In strict SSA, aka SSA with dominance property, the definition of a variable always dominates all its uses. We exploited this property and the concept of loop-nesting forest to design a fast two-phase data-flow algorithm: a first pass traverses the control-flow graph (CFG), propagating liveness information backwards, a second pass traverses the loop-nesting forest, updating liveness sets within loops. The algorithm is proved correct even for irreducible CFGs.

We analyzed its algorithmic complexity and evaluated its efficiency on SPEC INT 2000. Compared to traditional iterative data-flow approaches, which perform updates until a fixed point is reached, our algorithm is 2 times faster on average. Other approaches are possible that propagate from uses to definitions, one variable at a time, instead of unioning sets as in data-flow analysis. Our algorithm is 1.43 times faster than the fastest alternative on average, when sets are represented as bitsets and for optimized programs, i.e., when there are more variables and larger live-sets and live-ranges.

This work has been presented at the conference APLAS'11 [5].

6.7. SSI Revisited: A Program Representation for Sparse Dataflow Analyses

Participants: Andre Tavares [UFMG, Brazil], Mariza Bigonha [UFMG, Brazil], Roberto Bigonha [UFMG, Brazil], Benoit Boissinot, Fernando Pereira [UFMG, Brazil], Fabrice Rastello.

Dataflow analyses usually associate information about variables to program regions. Informally, if these regions are too small, e.g., a point between two consecutive statements, we call the analysis dense. On the other hand, if these regions include many such points, then we call it sparse. We developed a systematic method to build program representations that support forward and/or backward sparse analyses. To pave the way that leads to this framework, we first clarified the literature on intermediate program representations. We revisited the static single information (SSI) form introduced in the 90s and showed how to simplify the construction of program representations for unidirectional dataflow analyses. We showed how to cope with live-ranges that have multiple uses/definitions without losing the equivalence property with the initial dataflow analysis problem. This allows us to simplify, for unidirectional problems, the SSI construction algorithm.

We also showed that our approach, up to a parameter choice, subsumes other program representations such as the SSA, SSI, and e-SSA forms. We can produce intermediate representations isomorphic to the sparse evaluation graphs (SEGs) of Choi et al. This data structure enables sparse solutions to the class of dataflow problems called partitioned dataflow analysis (PDA). However, contrary to SEGs, we can handle - sparsely - problems that are not PDA. We have implemented this framework in the LLVM compiler and have empirically compared different program representations in terms of size and construction time.

This work is part of a collaboration with the Federal University of Minas Gerais 8.2 and is under reviewing process for the journal Science of Computer Programming.

6.8. Incremental Spilling

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

This work addresses the minimization of the spill code overhead in the contexts of both coupled and decoupled register allocation. We devised a heuristic approach called *stacking*; it incrementally allocates clusters of variables, as opposed to the conventional incremental spilling approach. We describe two polynomial methods, a stacking-optimal allocator and a greedy stacking-independent-set allocator. The first method is very close to the optimal allocation; the second method outperforms state-of-the-art heuristics for just-in-time compilation.

This work has been submitted for publication.

6.9. Program Analysis and Communication Optimizations for HLS

Participants: Christophe Alias, Alain Darte, Alexandru Plesco.

High-level synthesis (HLS) tools are now getting more mature for generating hardware accelerators with an optimized internal structure, thanks to efficient scheduling techniques, resource sharing, and finite-state machines generation. However, interfacing them with the outside world, i.e., integrating the automatically-generated hardware accelerators within the complete design, with optimized communications, so that they achieve the best throughput, remains a very hard task, reserved to expert designers. The goal of our research on HLS is to study and to develop source-to-source strategies to improve the design of these interfaces, trying to consider the HLS tool as a back-end for more advanced front-end transformations.

Using the C2H HLS tool from Altera, which can synthesize hardware accelerators communicating to an external DDR-SDRAM memory, we showed that it is possible to automatically restructure the application code, to generate adequate communication processes in C, and to compile them all with C2H, so that the resulting application is highly-optimized, with full usage of the memory bandwidth.

These transformations and optimizations, which combine techniques such as double buffering, array contraction, loop tiling, software pipelining, among others, were incorporated in an automatic source-to-source transformation tool, called CHUBA (see Section 5.7), based on the polyhedral model representation. Our study shows that HLS tools can indeed be used as back-end optimizers for front-end optimizations, as it is the case for standard compilation with high-level transformations developed on top of assembly-code optimizers. We believe this is the way to go for making HLS tools viable. The complete automation of the process will be presented at PPoPP'12 [3] and Impact'12 [15].

We also showed how to extend this method to programs with irregular control and array accesses. The main difficulty arises when some data may be redefined in the accelerator but this is not sure. We showed that techniques based on parametric polyhedral optimizations can be used to generate the sets of data to be loaded (resp. stored) just before (resp. after) each tile. An interesting feature is that the previous method appears nicely as a particular case when no approximation is needed. This work is fully described in a research report [23], but is not yet published in a conference or journal.

6.10. Compilation of Hardware Accelerators with Pipelined Arithmetic

Participants: Christophe Alias, Bogdan Pasca [PhD student, Arénaire Inria Team], Alexandru Plesco.

By nature, source-level optimizations cannot perform *fine* optimizations of the datapath and the control, sometimes mandatory to obtain performances. In high-level synthesis, the circuit generated must be efficient and must produce quality results. This last point is the specialty of the Arénaire Inria-team, which develops the tool FLOPOCO, an open-source FPGA-specific generator of pipelined floating-point arithmetic operators, from a functional description as $(x, y, z) \mapsto e^x + y * \cos(z)$. The user can specify the precision (mantissa, exponent) and the maximum frequency, then FLOPOCO generates the corresponding operator.

We have developed an algorithm to automatically generate, from a C program, an hardware accelerator that efficiently uses these pipelined operators. The main issue is to reschedule the initial program execution in order to keep the operator's pipeline as busy as possible, while minimizing memory access. Then, the new execution schedule is used to generate the VHDL code of finite state machines (FSM) controlling the data-flow through the arithmetic operator. This work has been published at the conference ARC'11 [4].

We also showed how our method can be used as a tool to generate control FSMs of multiple parallel computing cores accelerating the same application [25]. This work has been submitted to the journal Microprocessors and Microsystems.

This is still a work in progress and many issues need to be addressed, for example how to extend the program model to general nested loops with more general dependences. These extensions will require to handle properly the communications between the operators and temporary buffers. We believe that the array contraction technique developed in Compsys can be helpful in this context too.

6.11. FPGA Optimized Table Maker's Dilemma Architecture

Participants: Alexandru Plesco, Florent De Dinechin [Assistant Prof., ARENAIRE Inria Team], Jean-Michel Muller [Research Director, ARENAIRE Inria Team], Bogdan Pasca [PhD student, ARENAIRE Inria Team].

In this work, with some members of the Arénaire team, we developed an algorithm that enables to perform the table maker's dilemma on a very regular architecture such as an FPGA. The algorithm is crucially different from the algorithm implemented on a standard PC and exploits efficiently the FPGA optimized high-performance arithmetic operators.

The core component (TaMaDi core) of our design is the polynomial evaluator based on the tabulated differences method that uses Remez algorithm. We instantiated multiple TaMaDi cores that work on 2^m disjoint intervals obtained from the splitting of the input interval. The TaMaDi cores are connected using a pipelined communication architecture based on the credit communication methodology. We focused on the communication architectures internal to a FPGA that can scale with limited impact on the frequency of the generated data processing architecture. We defined a new pipelined credit based communication interface that leads to a full-rate pipeline data transmission with the cost of small transfer initialization latency. This design has been proven to be effective on parallel design of TaMaDi FPGA specific algorithms. This strategy can be reused in any parallel FPGA design.

This work has been published at ASAP'11 [14] where it received a best paper award.

6.12. Termination of Big Programs

Participants: Christophe Alias, Laure Gonnord, Guillaume Andrieu [Undergraduate Internship, Polytech'Lille].

In a previous work with Alain Darte and Paul Feautrier, we showed how to prove the termination of a certain class of irregular programs with while loops [29]. Our technique amounts to compute a sequential schedule – called ranking function – for the program, reinvesting most of the techniques from [37] to schedule static loops. Our termination method is based on the resolution of a linear programming instance, and does not scale.

In most of big programs, all the information is not relevant for proving termination. Only a few slices need a termination proof. Moreover, inside a loop nest, termination can be proved incrementally. Inner loops are proved to terminate, and then are replaced by a summary. Our contribution is thus a reduction to prove a single loop termination, the price being the approximation made while computing the summary of a given (nest of) loop(s). This method can also be used to schedule irregular programs, which is clearly a need for program optimization.

A prototype has been designed and is currently under testing. A paper is in preparation. This method has been developed during the undergraduate internship of Guillaume Andrieu, from the Engineering School Polytech'Lille.

6.13. Simplification of Boolean Affine Formulas

Participant: Paul Feautrier.

Boolean Affine Formulas, in which affine inequalities are combined by boolean connectives, are ubiquitous in computer science: static analysis, code and hardware generation, symbolic model checking and many other techniques use them as a compact representation of large or infinite sets. Common algorithms tend to generate large and highly redundant formulas, hence the necessity of a simplifier for keeping the overall complexity under control. Simplification is a difficult problem, at least as hard as SMT solving, with a worst case complexity exponential in the number of affine inequalities. Paul Feautrier has proposed a new method, based on path cutting in Ordered Binary Decision Diagrams, which is able to take advantage of any regularity in the subject formula to speed up simplification. The method has been implemented and tested on benchmarks from several application domains.

The method has been presented at the 4th "Rencontres de la communauté française de compilation". A detailed description has been submitted for publication; see also [28].

6.14. Retiming for Faust

Participants: Alain Darte, Alexandre Isoard [Master 1 student, ENS-Lyon], Yann Orlarey [Grame].

Faust (*Functional Audio Stream*) is a formal specification stream-like language designed for real-time signal processing and synthesis. Faust programs are compiled into equivalent C++ programs and optimized for parallel execution. One of the core optimizations in Faust is a preliminary phase called *normalization*, which amounts to change the delays between operators into an equivalent normalized form used for, among others, redundancy elimination.

We showed that this normalization is actually a special form of retiming, a well-known technique in circuit design and loop transformations. However, an important subtlety needs to be considered: the problem of initialization of signals, which is usually not even mentioned in the retiming literature. We proved that the problem comes from "time-dependent" operators in Faust and that all these operators can be decomposed into combinations of regular operators (in the sense that standard retiming applies) and a single elementary time-dependent operator, called *init*, which transfers any signal with no modification for time $t \geq 0$ and outputs a constant for time $t < 0$.

This collaboration between Compsys and Grame did not go beyond the Master internship of A. Isoard yet and did not lead to an actual implementation within Faust.

7. Contracts and Grants with Industry

7.1. MEDIACOM: Nano2012 Project with STMicroelectronics on SSA, Register Allocation, and JIT Compilation

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. This new project concerns both aggressive optimizations and the application of the previously-developed techniques to JIT compilation. Related activities are described in Sections 6.2 to 6.7.

Unfortunately, due to a unilateral decision of the government, all fundings related to Nano2012 have been cancelled, or at least frozen, in 2011. The salary of PhD students (such as Quentin Colombet) was guaranteed by contract but neither the salary of engineers and post-docs, nor the regular budget. Thanks to Inria who covered his salary, we succeeded to keep Florian Brandner until October 2011. However, the whole travelling budget was cut. Mediacom survived and will continue, as planned, until the end of 2012, but in a less ambitious format.

7.2. S2S4HLS: Nano2012 Project with STMicroelectronics on Source-to-Source Transformations for High-Level Synthesis

Participants: Christophe Alias, Alain Darte, Paul Feautrier, Alexandru Plesco.

This contract has started in January 2009 as part of the funding mechanism Nano2012. This is a joint project with the Cairn Inria project-team and STMicroelectronics, whose goal is the study and development of source-to-source program transformations, in particular loop transformations, that are worth applying on top of HLS tools. This includes restructuring transformations, program analysis, memory optimizations and array reshaping, etc. Some of the work presented in Section 6.9 is part of this project.

The fact that Nano2012 was cancelled or only frozen (depending on the day) forced us to quit the project. We were indeed about to hire a post-doc on this topic and could not do it due to this unexpected governmental decision. This contract was therefore closed in 2011.

7.3. Creation of the Start-Up Zettice

Following his PhD, Alexandru Plesco initiated a start-up on high-level synthesis for FPGAs, named Zettice, and 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, 6.9, 6.10, and 6.11 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. Christophe Alias is in charge of the collaboration between Compsys and Zettice.

8. Partnerships and Cooperations

8.1. National Initiatives

- 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", initiated in 2010 and officially animated by Fabrice Rastello and Laure Gonnord, are now well-established 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).
- Christophe Alias and Paul Feautrier have been active participants in an effort to structure the french high-level synthesis community, including both actors from academia (TIMA, IRISA, LasTIC, ASIM) and industry (Thales, Bull). The aim of this effort was to submit an ANR proposal for the Arpege initiative. A first version was submitted in 2010, but was rejected mostly on the ground that the project leader should have been from industry rather than academia. A revised proposal, under the leadership of the Magillem company, was submitted in March 2011 and rejected too. It seems evident in retrospect that the HLS community has yet to find a clearer balance between new research and industrial development, and that a new submission must wait for a more mature approach.

8.2. Participation in International Programs

- Fabrice Rastello has obtained a FAPEMIG-INRIA (Brazil-France) funding to collaborate with Mariza A. S. Bigonha, Fernando M. Q. Pereira, and Roberto S. Bigonha from the Federal University of Mina Gerais (UFMG) in Brazil. The work on static single information form described in Section 6.7, and the work on register allocation to handle aliasing described in Section 6.5 are part of this collaboration.
- From July 2010 till July 2011, Fabrice Rastello was in a sabbatical year at Colorado State University within the group of Sanjay Rajopadhye, and in connection with the PathScale compiler company.

8.3. Informal Contacts

- Compsys has regular contacts with Sebastian Hack at Saarland University (Saarbrücken, Germany), Philip Brisk at University of California, Riverside (Riverside, USA), and Benoît Dupont de Dinechin (Kalray, Grenoble) on back-end code optimizations.
- Compsys has regular contacts with P. Sadayappan (Ohio State University), J. (Ram) Ramanujam (Lousiana State University), and Sanjay Rajopadhye (Colorado State University), on polyhedral code transformations. Fabrice Rastello was in sabbatical in 2010-2011 in Sanjay Rajopadhye's group. Christophe Alias is co-advising a PhD with Sanjay Rajopadhye, with an agreement to be signed between ENS-Lyon and Colorado State University.
- In France, Compsys is particularly linked with researchers such as Albert Cohen (Parkas team, Inria), Steven Derrien (Cairn team, Inria), Alain Greiner (LIP6, Paris), Alain Ketterlin (Camus team, Inria), Benoît Dupont de Dinechin (Kalray), Christophe Guillon (STMicroelectronics).
- Compsys, as some other Inria projects, is involved in the network of excellence HiPEAC (High-Performance Embedded Architecture and Compilation, http://www.hipeac.net/). Compsys is also a (distant) partner of the network of excellence Artist2 to keep an eye on the developments of MPSoC.
- Florian Brandner is collaborating with the group of Andreas Krall at the Vienna University of Technology on topics related to the processor description language xADL and on compilation for explicitly parallel processors (EPICOpt, http://www.complang.tuwien.ac.at/epicopt/). He is additionally working with Martin Schöberl from the Technical University of Denmark (DTU) on topics evolving around time-predictable computing.
- Alain Darte is in contact with Yann Orlarey from the Grame team (Lyon, "Centre National de Création Musicale"). They co-advise a Master 1 internship on some features in the development of Faust, a compiled language for real-time audio signal processing.

8.4. Visits of Research Scientists

Since Autumn 2010, several researchers visited Compsys and gave talks in our working groups.

- Amir Ben Amram (Tel Aviv University, Israël).
- Sebastian Hack (Saarland University, Germany).
- Andreas Krall and Gergö Barany (Vienna University, Austria).
- J. Ramanujam (Baton Rouge University, Louisiana)
- Antoniu Pop (Ecole des Mines, Paris).
- Benoît Dupont de Dinechin (Kalray, Grenoble).
- Alain Ketterlin (Camus Inria team, Strasbourg).
- Albert Cohen (Parkas Inria team, Paris).

8.5. Internships

In Spring 2011, three internships were advised in Compsys.

- Guillaume Andrieu (Polytech'Lille engineering school, Master level): termination of big programs.
- Alexandre Isoard (ENS-Lyon, M1 Master level): retiming for Faust.
- François Gindraud (ENS-Lyon, M1 Master level): ψ-SSA, gated-SSA, and variants.

9. Dissemination

9.1. Animation of the Scientific Community

- Fabrice Rastello was the local chair of the CGO'11 conference (ACM/IEEE International Conference on Code Generation and Optimization), which took place in Chamonix, France, in April 2011. He was also part of the organization of CGO'11 satellite workshops.
- Christophe Alias was the main organizer of IMPACT'11 (international workshop on polyhedral compilation techniques), held in conjunction with CGO'11. This workshop was the very first international event on this topic. Alain Darte gave the keynote of this first edition [2]. Christophe Alias is now a member of the steering committee of IMPACT, whose second edition will took place within the conference HIPEAC'12 (January 2012). Also part of CGO'11, the workshop ACCA'11 (analyze to compile, compile to analyze) was co-organized by Laure Gonnord.
- Paul Feautrier has been an active participant in the elaboration of the Encyclopedia of Parallel Programming, (David Padua ed.), to be published by Springer Verlag. He has been a member of the scientific committee, and has contributed four entries, on array layout for parallel computing [19], Bernstein's conditions [20], dependences [21], and the polyhedral model (with Christian Lengauer) [22]. Alain Darte also contributed a chapter on the parallelism detection in loops [18].
- Paul Feautrier is associate editor of Parallel Computing and the International Journal of Parallel Computing. He has been a member of the program committee of the ParCo'2011 conference and of the IMPACT'2012 workshop (as Christophe Alias). He has reviewed many submissions to various conferences and journals.
- Alain Darte is associate editor of ACM Transactions on Embedded Computing Systems. He was member of the program committees of the conferences CASES'11 (Compilers, Architectures, and Systems for Embedded Systems), DATE'12 (Design and Test in Europe), and CC'12 (Compiler Construction).
- Paul Feautrier has been a reviewer for the PhD thesis of Nicolas Pouillon (Université Pierre et Marie Curie, September 2011), and a member of the defense committee for the PhD of Bogdan Pasca (ENS Lyon, September 2011).
- Fabrice Rastello has been a member of the defense committee for the PhD thesis of Antoniu Pop (CRI, MINES ParisTech, September 2011), and a member of the defense committee for the PhD thesis of Boubacar Diouf (Paris-Sud University, December 2011).
- In collaboration with participants of the SSA'09 workshop, Fabrice Rastello is the editor of a book on SSA (Static Single Assignment) and its application in compilation, optimization, and program analysis. In a joint work with Diego Novillo (Google), Florian Brandner contributed a chapter on sparse dataflow analysis to this book.
- Fabrice Rastello and Laure Gonnord are in charge of the "steering committee" of the "journées françaises de la compilation" that gathers, twice a year, around fifty participants.
- In 2011, Alain Darte was member of the hiring committee for a professor position at ENS-Lyon.
- Paul Feautrier has taken charge of the organization of Compsys' workgroups.

9.2. 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.
- Master 1: Christophe Alias gave a Master 1 course on "Compilation" at ENS-Lyon.
- Master 2: Alain Darte gave a Master 2 course on "Advanced code optimizations" at ENS-Lyon. Laure Gonnord and Paul Feautrier contributed to this course too (2 hours each).

- Master 1 and 2: Christophe Alias is organizer of the Winter Master School “Vérification et certification du logiciel” that will take place 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 Guillaume Andrieu, from the Engineering School Polytech’Lille. This internship lasted four months, from May 16th, 2011 to August 15th, 2011. The subject was about termination of big C programs. Alain Darte advised the M1 internship of Alexandre Isoard on retiming techniques for the streaming language Faust. Florian Brandner advised the M1 internship of François Gindraud on intermediate representations related to extensions of SSA.

  PhD in progress: Quentin Colombet, “Advanced topics in register allocation”, started in January 2010, at ENS-Lyon. Advisors: Alain Darte, Fabrice Rastello.

  PhD in progress : Guillaume Iooss, “Program analysis and optimization in the polyhedral model”, started in September 2011, at ENS-Lyon and Colorado State University. Advisors: Sanjay Rajopadhye (Colorado State University), Christophe Alias, Alain Darte.

10. Bibliography

Publications of the year

Articles in International Peer-Reviewed Journal

[1] B. BOISSINOT, P. BRISK, A. DARTE, F. RASTELLO. *SSI Properties Revisited*, in "ACM Transactions on Embedded Computing Systems", 2011, Special Issue on Software and Compilers for Embedded Systems, to appear.

Invited Conferences

[2] A. DARTE. *Approximations in the Polyhedral Model*, in "1st International Workshop on Polyhedral Compilation Techniques (IMPACT’11), CGO’11 Workshop", Chamonix, April 2011, Invited talk (keynote).

International Conferences with Proceedings

[3] 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, USA, IEEE Computer Society, February 2012, Short paper.

[4] C. ALIAS, B. PASCA, A. PLESCO. *Automatic Generation of FPGA-Specific Pipelined Accelerators*, in "7th International Symposium on Applied Reconfigurable Computing (ARC’11)", Belfast, UK, Springer Verlag, March 2011, p. 53–66.

[5] B. BOISSINOT, F. BRANDNER, A. DARTE, B. DUPONT DE DINECHIN, F. RASTELLO. *A Non-Iterative Data-Flow Algorithm for Computing Liveness Sets in Strict SSA Programs*, in "9th Asian Symposium on Programming Languages and Systems (APLAS’11)", Springer Verlag, December 2011.

[6] F. BRANDNER, Q. COLOMBET. *Copy Elimination on Data Dependence Graphs*, in "Symposium on Applied Computing (SAC’12)", Trento, Italy, ACM Press, March 2012, Short paper.

[7] Q. COLOMBET, B. BOISSINOT, P. BRISK, S. HACK, F. RASTELLO. *Graph Coloring and Treescan Register Allocation Using Repairing*, in "International Conference on Compilers, Architectures, and Synthesis of Embedded Systems (CASES'11)", Taipei, Taiwan, IEEE Computer Society, October 2011.

[8] 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.

[9] B. COMBEMALE, L. GONNORD, V. RUSU. *A Generic Tool for Tracing Executions Back to a DSML's Operational Semantics*, in "7th European Conference on Modelling Foundations and Applications (ECMFA 2011)", Birmingham, United Kingdom, Lecture Notes in Computer Science, Springer Verlag, June 2011, vol. 6698, p. 35–51, http://hal.inria.fr/hal-00593425.

[10] A. GAMATIE, L. GONNORD. *Static Analysis of Synchronous Programs in Signal for Efficient Design of Multi-Clocked Embedded Systems*, in "ACM SIGPLAN/SIGBED Conference on Languages, Compilers, and Tools for Embedded Systems (LCTES'11)", New York, NY, USA, ACM, 2011, p. 71–80, http://doi.acm.org/10.1145/1967677.1967688.

[11] D. MONNIAUX, L. GONNORD. *Using Bounded Model Checking to Focus Fixpoint Iterations*, in "Static analysis (SAS'11)", E. YAHAV (editor), Lecture Notes in Computer Science, Springer Verlag, 2011, vol. 6887, p. 369–385.

[12] M. SCHOEBERL, P. SCHLEUNIGER, W. PUFFITSCH, F. BRANDNER, C. W. PROBST, S. KARLSSON, T. THORN. *Towards a Time-predictable Dual-Issue Microprocessor: The Patmos Approach*, in "Bringing Theory to Practice: Predictability and Performance in Embedded Systems, DATE Workshop PPES'11", Grenoble, France, March 2011, vol. 18, p. 11-21, http://hal.inria.fr/inria-00585320/en.

[13] A. TAVARES, Q. COLOMBET, M. BIGONHA, C. GUILLON, F. M. Q. PEREIRA, F. RASTELLO. *Decoupled Graph-Coloring Register Allocation with Hierarchical Aliasing*, in "14th International Workshop on Software and Compilers for Embedded Systems (SCOPES'11)", St. Goar, Germany, ACM Press, 2011, p. 1–10.

[14] *Best Paper*
F. DE DINECHIN, J.-M. MULLER, B. PASCA, A. PLESCO. *An FPGA Architecture for Solving the Table Maker's Dilemma*, in "22nd IEEE International Conference on Application-specific Systems, Architectures and Processors (ASAP'11)", Santa Monica, CA, IEEE Computer Society, September 2011.

Conferences without Proceedings

[15] 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)", Paris, January 2012.

[16] F. BRANDNER, A. DARTE. *Compiler-driven Optimization of the Worst-Case Execution Time*, in "Workshop "Analyse to Compile, Compile to Analyse" (ACCA'11), held with CGO'11", Chamonix, L. GONNORD, D. MONNIAUX (editors), April 2011.

[17] 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.

Scientific Books (or Scientific Book chapters)

[18] A. DARTE. *Optimal Parallelism Detection in Nested Loops*, in "Encyclopedia of Parallel Programming", D. PADUA (editor), Springer, 2011, to appear.

[19] P. FEAUTRIER. *Array Layout for Parallel Processing*, in "Encyclopedia of Parallel Programming", D. PADUA (editor), Springer, 2011, to appear.

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

[21] P. FEAUTRIER. *Dependences*, in "Encyclopedia of Parallel Programming", D. PADUA (editor), Springer, 2011, to appear.

[22] P. FEAUTRIER, C. LENGAUER. *The Polyhedron Model*, in "Encyclopedia of Parallel Programming", D. PADUA (editor), Springer, 2011, to appear.

Research Reports

[23] C. ALIAS, A. DARTE, A. PLESCO. *Kernel Offloading with Optimized Remote Accesses*, INRIA, July 2011, n^{o} RR-7697, http://hal.inria.fr/inria-00611179/en.

[24] C. ALIAS, A. DARTE, A. PLESCO. *Program Analysis and Source-Level Communication Optimizations for High-Level Synthesis*, INRIA, June 2011, n^{o} RR-7648, http://hal.inria.fr/inria-00601822/en.

[25] C. ALIAS, B. PASCA, A. PLESCO. *FPGA-Specific Synthesis of Loop-Nests with Pipelined Computational Cores*, INRIA, July 2011, n^{o} RR-7674, http://hal.inria.fr/inria-00606977/en.

[26] F. BRANDNER, B. BOISSINOT, A. DARTE, B. DUPONT DE DINECHIN, F. RASTELLO. *Computing Liveness Sets for SSA-Form Programs*, INRIA, April 2011, n^{o} RR-7503, http://hal.inria.fr/inria-00558509/en.

[27] F. BRANDNER, Q. COLOMBET. *Parallel Copy Elimination on Data Dependence Graphs*, INRIA, September 2011, n^{o} RR-7735, http://hal.inria.fr/inria-00625131/en.

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

References in notes

[29] 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.

[30] F. BOUCHEZ, Q. COLOMBET, A. DARTE, C. GUILLON, F. RASTELLO. *Parallel Copy Motion*, in "13th International Workshop on Software & Compilers for Embedded Systems (SCOPES'10)", St. Goar, Germany, ACM Press, June 2010, p. 1–10.

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

[32] 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.

[33] 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.

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

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

[36] 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.

[37] P. FEAUTRIER. *Some Efficient Solutions to the Affine Scheduling Problem, Part II, Multidimensional Time*, in "International Journal of Parallel Programming", December 1992, vol. 21, n^{o} 6.

[38] 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.

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

[40] 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.

[41] 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.

[42] 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.