INRIA

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Project-Team Compsys

Compilation and Embedded Computing Systems

Rhône-Alpes

THEME COM

Activity Report

2006

Table of contents

1. Team

Compsys exists since January 2002 as part of Laboratoire de l'Informatique du Parallélisme (Lip, UMR CNRS ENS-Lyon UCB-Lyon Inria 5668), located at ENS-Lyon, and as an Inria pre-project. It is now a full Inria project since January 2004. The objective of Compsys is to adapt and extend optimization techniques, primarily designed for high performance computing, to the special case of embedded computing systems.

Head of the team

Alain Darte [Research Director (DR) CNRS, HdR]

Administrative assistant

Isabelle Antunes [Part-time]

Research scientists

Paul Feautrier [Professor (Pr) ENS-Lyon, HdR]
Antoine Fraboulet [Assistant Professor (MC) Insa-Lyon, Inria delegation from September 2005 to August 2006]
Fabrice Rastello [Research Associate (CR) Inria]
Tanguy Risset [Professor (Pr) Insa-Lyon, HdR]

Post doc

Christophe Alias [February 2006-]
Sebastian Hack [December 2006-December 2007]
Ouassila Labbani [December 2006-December 2007]

PhD Students

Benoit Boissinot [ENS-Lyon grant, 2006-...]
Florent Bouchez [ENS-Lyon grant, 2005-...]
Nicolas Fournel [MESR grant, 2004-...]
Philippe Grosse [Grant CEA-LETI, Grenoble, 2004-...]
Alexandru Plesco [MESR grant, 2006-...]
Clément Quinson [BDI CNRS and STMicroelectronics, 2005-...]
Antoine Scherrer [BDI CNRS and STMicroelectronics, 2003-...]

2. Overall Objectives

2.1. Overall Objectives

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

An embedded computer is a digital system, part of a larger system (appliances like phones, TV sets, washing machines, game platforms, or larger systems like radars and sonars), which is not directly accessible to the user. In particular, this computer is not programmable in the usual way. Its program, if it exists, has been loaded as part of the manufacturing process, and is seldom (or never) modified.

The objective of Compsys is to adapt and extend code optimization techniques, primarily designed for high performance computing, to the special case of embedded computing systems. Compsys has four research directions, centered on compilation methods for simple or nested loops. These directions are:

- code optimization for specific processors (mainly DSP and VLIW processors);
- platform-independent transformations (including loop transformations for memory optimization);
- silicon compilation and hardware/software codesign
- development of polyhedra manipulation tools.

These research activities are supported by a marked investment in polyhedra manipulation tools, with the aim of constructing operational software tools, not just theoretical results. Hence the fourth research theme is centered on the development of these tools. We expect that the Compsys experience on key problems in the design of parallel programs (scheduling, loop transformations) and the support of our respective parent organizations (Inria, CNRS, Ministry of Education) will allow us to contribute significantly to the European research on embedded computing systems.

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*: (1) control-oriented and hard real-time embedded systems (automotive, nuclear plants, airplanes, etc.) and (2) compute-intensive embedded systems (signal processing, multi-media, stream processing). The Compsys team is primarily concerned with the second type of embedded systems, which is now referred as *embedded computing systems*. The design and compilation methods proposed by the team will be efficient on compute intensive processing with big sets of data processed in a pipelined way.

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.

Compsys' aims are to develop new compilation and optimization techniques for embedded systems. The field of embedded computing system design is large, and Compsys does not intend to cover it in its entirety. We are mostly interested in the automatic design of accelerators, for example optimizing a piece of (regular) code for a DSP or designing a VLSI chip for a digital filter. Compsys' specificity is the study of code transformations intended for optimization of features that are specific to embedded systems, like time performances, power consumption, die size. Our project is related to code optimization (like some of the research in the Inria projects Alchemy and Caps) and to high-level architectural synthesis (like the Inria action R2D2).

Our priority towards embedded software is motivated by the following observations:

- The embedded system market is expanding. Among many factors, one can quote pervasive digitalization, low cost products, appliances, etc.
- Software engineering for embedded systems is not well developed in France, especially if one considers the importance of actors like Alcatel, STMicroelectronics, Matra, Thales, and others.
- Since embedded systems have an increasing complexity, new problems are emerging: computer aided design, shorter time-to-market, better reliability, modular design, and component reuse.

Recently, several tools for high-level synthesis have appeared in the synthesis/compiler community. These tools are mostly based on C or C++ (SystemC [1], VCC, and others). The support for parallelism in these tools is minimal, but academic projects are more concerned: Flex [2] and Raw [3] at MIT, Piperench [4] at Carnegie-Mellon University, PiCo from HP Labs and now at the Synfora [5] start-up, Compaan [6] at the University of Leiden, and others.

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 incidence 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 between the compilation and chip design communities is needed.

[1] http://www.systemc.org/
[2] http://www.flex-compiler.lcs.mit.edu/
[3] http://www.cag.lcs.mit.edu/raw/
[4] http://www.ece.cmu.edu/research/piperench/
[5] http://www.synfora.com/
[6] http://embedded.eecs.berkeley.edu/compaan/

Computer aided design of silicon systems is a wide field. The main expertise in Compsys 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 for embedded systems. This is an essential part of Compsys activities and will be a test of the project success.

3. Scientific Foundations

3.1. Introduction

Twenty years ago, the subject of compilation was considered to be mature enough to become an industry, using tools like Lex and Yacc for syntax analysis, and Graham-Glanville code-generator generators. The subject was reactivated by the emergence of parallel systems and the need for automatic parallelizers. The hot topic is now the intermediate phase between syntax analysis and code generation, where one can apply optimizations, particularly those that exploit parallelism, either in an autonomous way or with the help of the programmer. In fact, there is parallelism in all types of digital systems, from supercomputers to PCs to embedded systems.

Compilation consists in a succession of code transformations. These transformations are applied to an intermediate representation that may be very similar to the source code (high-level optimization), or very similar to machine code (assembly code and even register transfer level (RTL) for circuit specification). Almost always, the main constraint is that the meaning (or semantics) of the source program must not be altered. Depending on the context, one may have to express the fact that the degree of parallelism must not exceed the number of available resources (processors, functional units, registers, memories). Finally, the specification of the system may enforce other constraints, like latency, bandwidth, and others. In the case of a complex transformation, one tries to express it as a constrained optimization problem.

For instance, in automatic parallelization, the French community has mainly targeted loop optimization. If the source program obeys a few regularity constraints, one can obtain linear formulations for many of the constraints. In this way, the optimization problem is reduced to a linear program to be solved either over the rationals, or, in few cases, over the integers. These are well-known techniques, based on the theory of convex polyhedra – hence the name *polyhedral model* that is often affixed to the method. Based on this theory, efficient software tools have been developed. One-dimensional and multi-dimensional scheduling techniques [38], [39] are an outcome of this research and are ubiquitously used for handling nested loop programs (regular circuit synthesis, process networks for instance).

Extending these methods to embedded systems is difficult because the objective function is complex to express. Performance, for instance, is no longer an objective but a constraint, the goal being to minimize the “cost” of the system, which may be a complex mixture of the design, the manufacturing, and the operation costs. For instance, minimizing the silicon area improves the yield and hence decreases the manufacturing cost. Power consumption is an important factor for mobile systems. Computer scientists are used to a paradigm in which the architecture is fixed and the only free variable is the program. The critical problem is thus to extend our optimization methods to handle many more free variables, mostly of a discrete nature.

In parallel with compiler research, the circuit design community has developed its own design procedures. These techniques have as input a structural specification of the target architecture, and use many heavy-weight tools for synthesis, placement, and routing. These tools mainly use sophisticated techniques for boolean optimization and do not consider loops. When trying to raise the level of abstraction, circuit designers have introduced the terms *architectural synthesis* and *behavioral synthesis*, but the tools did not follow, due to the above mentioned problems (increasing complexity of the constraints, increasing number of free variables).

Technological advances in digital electronics have motivated the emergence of standards for design specifications and design methodologies. Languages like VHDL, Verilog, and SystemC have been widely accepted. The concepts of off-the-shelf components (intellectual property or IP) and of platform-based design are gaining importance. However, the problem remains the same: how to transform a manual design process into a compilation process?

The first proposal was to use several tools together. For instance, the hardware-software partitioning problem is handled by architecture explorations, which rely on rough performance estimates, and the degree of automation is low. But since the complexity of systems on chip still increases according to Moore's law, there is a pressing need to improve the design process and to target other architectures, like DSP, or reconfigurable FPGA platforms. The next generation of systems on chip will probably mix all the basic blocks of today's technology (DSP, Asic, FPGA, network, and a memory hierarchy with many levels). We intend to participate in the design and programming of such platforms.

Our vision of the challenges raised by these new possibilities is the following: one has to *understand* the technological constraints and the existing tools in order to *propose* innovative, efficient, and realistic compilation techniques for such systems. Our approach consists in modeling the optimization process as precisely as possible and then in finding powerful techniques to get an optimal solution. Past experience has shown that taking simultaneously all aspects of a problem into account is nearly impossible.

Compsys has four research directions, each of which is a strong point in the project. These directions are clearly not independent. Their interactions are as follows: "Platform-Independent Code Transformations" (Section 3.3) is on top of "Optimization for Special Purpose Processors" (Section 3.2) and "Hardware and Software System Integration" (Section 3.4), since its aim is to propose architecture-independent transformations. "Federating Polyhedral Tools" (Section 3.5) is transversal because these tools are useful in all other research axes.

3.2. Optimization for Special Purpose Processors

Participants: Alain Darte, Fabrice Rastello, Paul Feautrier.

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. 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).

3.2.1. Optimization 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 predication, 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, but also the special form of applications [35] generate many open problems. Our goal is to contribute to the understanding and the solution of these problems.

Compilation for embedded processors is either aggressive or just in time (JIT). 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 so 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 compile applets on demand on the target processor. For compatibility and compactness, used languages are CLI 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 the ST100 and ST200). 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 ("Static Single Assignment" [49]). 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 [42], 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 [50], but also to study the impact of predicated code on the whole compilation process. What is the best time and place to do the if conversion? Which intermediate representation is the best one? Is there a universal framework for the various styles of predication, as found in VLIW and DSP processors?

3.2.2. Scheduling under Resource Constraints

The degree of parallelism of an application and the degree of parallelism of the target architecture do not usually coincide. Furthermore, most applications have several levels of parallelism: coarse-grained parallelism as expressed, for instance, in a process network (see Section 3.3.1), loop-level parallelism, which can be expressed by vector statements or parallel loops, instruction-level parallelism as in "bundles" for Epic or VLIW processors. One of the tasks of the compiler is to match the degree of parallelism of the application and the architecture, in order to get maximum efficiency. This is equivalent to finding a schedule that respects dependences and meets resource constraints. This problem has several variants, depending on the level of parallelism and the target architecture.

For instruction-level parallelism, the classical solution, which is found in many industrial compilers, is to do software pipelining using heuristics like modulo scheduling. This can be applied to the innermost loop of a nest and, typically, generates code for an Epic, VLIW, or super-scalar processor. The problem of optimal software pipelining can be exactly formulated as an integer linear program, and recent research has allowed many constraints to be taken into account, as for instance register constraints. However the codes amenable to these techniques are not fully general (at most one loop) and the complexity of the algorithm is still quite high. Several phenomena are still not perfectly taken into account. Some examples are register spilling and loops with a small number of iterations. One of our aims is to improve these techniques and to adapt them to the STMicroelectronics processors.

It is not straightforward to extend the software pipelining method to loop nests. In fact, embedded computing systems, especially those concerned with image processing, are two-dimensional or more. Parallelization methods for loop nests are well known, especially in tools for automatic parallelization, but they do not take resource constraints into account. A possible method consists in finding totally parallel loops, for which the degree of parallelism is equal to the number of iterations. The iterations of these loops are then distributed among the available processors, either statically or dynamically. Most of the time, this distribution is the responsibility of the underlying runtime system (consider for instance the "directives" of the OpenMP library). This method is efficient only because the processors in a supercomputer are identical. It is difficult to adapt it to heterogeneous processors executing programs with variable execution time. One of today's challenges is to extend and merge these techniques into some kind of multi-dimensional software pipelining or resource-constrained scheduling.

3.3. Platform-Independent Code Transformations

Participants: Christophe Alias, Alain Darte, Paul Feautrier, Antoine Fraboulet, Tanguy Risset.

Embedded systems generate new problems in high-level code optimization, especially in the case of loop optimization. During the last 20 years, with the advent of parallelism in supercomputers, the bulk of research in code transformation was mainly concerned with parallelism extraction from loop nests. This resulted in automatic or semi-automatic parallelization. It was clear that there were other factors governing performance, as for instance the optimization of locality or a better use of registers, but these factors were considered to be less important than parallelism extraction, at least to understand the foundations of automatic parallelization. Today, we have realized that performance is a consequence of many factors, and, especially in embedded systems, everything that has to do with data storage is of prime importance, as it impacts power consumption and chip size.

In this respect, embedded systems have two main characteristics. First, they are mass produced. This means that the balance between design costs and production costs has shifted, giving more importance to production costs. For instance, each transformation that reduces the physical size of the chip has the side-effect of increasing the yield, hence reducing the manufacturing cost. Similarly, if the power consumption is high, one has to include a fan, which is costly, noisy, and unreliable. Another point is that many embedded systems are powered from batteries with limited capacity. Architects have proposed purely hardware solutions, in which unused parts of the circuits are put to sleep, either by gating the clock or by cutting off the power. It seems that the efficient use of these new features needs help from the operating system. However, power reduction can be obtained also when compiling, e.g., by making better use of the processors or of the caches. For these optimizations, loop transformations are the most efficient techniques.

As the size of the needed working memory may change by orders of magnitude, high-level code optimization also has much influence on the size of the resulting circuit. If the system includes high performance blocks like DSPs or Asics, the memory bandwidth must match the requirements of these blocks. The classical solution is to provide a cache, but this goes against the predictability of latencies, and the resulting throughput may not be sufficient. In that case, one resorts to the use of scratch-pad memories, which are simpler than a cache but require help from the programmer and/or compiler to work efficiently. The compiler is a natural choice for this task. One then has to solve a scheduling problem under the constraint that the memory size is severely limited. Loop transformations reorder the computations, hence change the lifetime of intermediate values, and have an influence on the size of the scratch-pad memories.

The theory of scheduling is mature for cases where the objective function is, or is related to, the execution time. For other, non-local objective functions (i.e., when the cost cannot be directly allocated to a task), there are still many interesting open problems. This is especially true for memory-linked problems.

3.3.1. Modular Scheduling and Process Networks

Kahn process networks (KPN) were introduced thirty years ago [43] as a notation for representing parallel programs. Such a network is built from processes that communicate via perfect FIFO channels. One can prove that, under very general constraints, the channel histories are deterministic. Thanks to this property, 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 Asics 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. As an added benefit, a communication system in which the receive operation is not destructive is closer to the expectations of system designers.

The challenge with this model is that a modicum of control is necessary for complex applications like wireless networks or software radio. There is an easy conservative solution for intra-process control and channel reads. Conditional channel writes, on the other hand, raise difficult analysis and design problems, which sometimes verge on the undecidable.

The scheduling techniques of MMAlpha and Syntol (tools that we develop) are complex and need powerful solvers using methods from operational research. One may argue that compilation for embedded systems can tolerate much longer compilation times than ordinary programming, and also that Moore's law will help in tackling more complex problems. However, these arguments are invalidated by the empirical fact that the size and complexity of embedded applications increase at a higher rate than Moore's law. Hence, an industrial use of our techniques requires a better scalability, and in particular, techniques for modular scheduling. Some preliminary results have been obtained at École des Mines de Paris (especially in the framework of inter-procedural analysis), and in MMAlpha (definition of structured schedules). The use of process networks is another way of tackling the problem.

The scheduling of a process network can be done in three steps:

- In the first step, which is done one process at a time, one deduces the constraints on the channel schedules that are induced by the data dependences inside the process.
- In the second step, one gathers these constraints and solves them for the channel schedules.
- Lastly, the scheduling problem for each process is solved using the previous results.

This method has several advantages: each of the scheduling problems to be solved is much smaller than the global problem. If one modifies a process, one only has to redo step one for this process, and then redo the second and third steps completely. Lastly, this method promotes good programming discipline, allows reuse, and is a basic tool for the construction of libraries.

Off-the-shelf components pose another problem: one has to design interfaces between them and the rest of the system. This is compounded by the fact that a design may be the result of cooperation between different tools; one has to design interfaces, this time between elements of different design flows. Part of this work has been done inside MMAlpha; it takes the form of a generic interface for all linear systolic arrays. Our intention is to continue in this direction, but also to consider other solutions, like Networks on Chip and standard wrapping protocols such as VCI from VSIA [7].

3.3.2. *Theoretical Models for Scheduling and Memory Optimizations*

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 [40], and techniques for data allocation in scratch-pad memory. 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?

Theoretical studies here search for new scheduling techniques, with objective functions that are no longer linear. These techniques may be applied to both high-level applications (for source-to-source transformations) and low-level applications (e.g., in the design of a hardware accelerator). Both cases share the same computation model, but objective functions may differ in detail.

[7] http://www.vsia.org

One should keep in mind that theory will not be sufficient to solve these problems. Experiments are required to check the pertinence 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.

Alain Darte, who was cooperating on a regular basis with the PiCo project at HP Labs (now in Synfora), has already proposed some solutions to the memory minimization problem. These ideas may be implemented in the PiCo compiler in order to find their strengths and weaknesses. The interaction between Compsys and companies such as STMicroelectronics, Philips, or Thales on high-level synthesis will also be important to make sure that we focus on the right practical problems.

3.4. Hardware and Software System Integration

Participants: Alain Darte, Paul Feautrier, Antoine Fraboulet, Tanguy Risset.

Embedded systems have a very wide range of power and complexity. A circuit for a game gadget or a pocket calculator is very simple. On the other hand, a processor for digital TV needs a lot of computing power and bandwidth. Such performances can only be obtained by aggressive use of parallelism.

The designer of an embedded system must meet two challenges:

- one has to specify the architecture of the system, which should deliver the required performance, but no more than that;
- when this is done, one has to write the required software.

These two activities are clearly dependent, and the problem is how to handle their interactions.

The members of Compsys have a long experience in compilation for parallel systems, high-performance computers, and systolic arrays. In the design of embedded computing systems, one has to optimize new objective functions, but most of the work done in the polyhedral model can be reinvested. Our first aim is thus to adapt the polyhedral model to embedded computing systems, but this is not a routine effort. As we will see below, a typical change is to transform an objective function into a constraint or vice-versa. The models of an embedded accelerator and of a compute-intensive program may be similar, but one may have to use very different solution methods because the unknowns are no longer the same, and this is why this topic is challenging.

3.4.1. Design of Accelerators for Compute-Intensive Applications

The advent of high-level synthesis techniques allows one to create specific design for reconfigurable architectures, for instance with MMAlpha [8] (for regular architectures) or with lower-level tools such as HandelC, SiliconC, and others. Validating MMAlpha as a rapid prototyping tool for systolic arrays on FPGA will allow designers to use it with a full knowledge of its possibilities. To reach this goal, one has first to firm up the underlying methodology and then to try to interface it with tools for control-intensive applications.

Towards this goal, the team will use the know-how that Tanguy Risset has acquired during his participation in the Cosi Inria project (before 2001) and also the knowledge of some members of the Arénaire Inria project (Lip). This work is a natural extension of the “high-level synthesis” action in the Inria project Cosi. We want to show that, for some applications, we can propose, in less than 10 minutes, a correct and flexible design (including the interfaces) from a high-level specification (in C, Matlab, or Alpha). We also hope to demonstrate an interface between our tool, which is oriented towards regular applications, and synchronous language compilers (Esterel, Syndex), which are more control oriented.

[8] http://www.irisa.fr/cosi/ALPHA/

Another important issue is to understand what are the needs in program transformations to be able to use, in practice, high-level tools for synthesizing hardware accelerators. All such tools, including MMAlpha 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 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.

3.4.2. Hardware Interfaces and On-Chip Traffic Analysis

Connecting the various components of a machine on the same interconnect is a challenge, and the most probable solution is the use of an on-chip network instead of the classical on-chip bus. In order to set the parameters of this on-chip network as soon as possible, fast simulation of the interconnection network is needed early in the design flow. To achieve this, we propose to replace some components by stochastic traffic generators. The design of the traffic generators has to be as fast as possible, in order to prototype rapidly different parameters of the network on chip.

We are actively working in the SoCLib group (http://soclib.lip6.fr). We have developed a deep understanding of SoCLib simulation models and we have started collaborations with hardware designers in the LIP6 laboratory (Paris) and Lester laboratory (Lorient). Our aim is to adapt the MMAlpha tool to generate simulation models that are compatible with SoCLib. We will particularly concentrate on the data-flow interface generator, which should be adapted to IPs produced by the Gaut high-level synthesis tool (Lester). These developments will allow fast prototyping of SoC in SoCLib, particularly when a data-flow hardware accelerator is needed for compute-intensive treatments.

3.4.3. Optimization for Low Power

Present-day general-purpose processors need much more power than was usual a few years ago: about 150W for the latest models, or more than twice the consumption of an ordinary TV set. The next generation will need even more power, because leakage currents, which are negligible at present, will increase exponentially as the feature size decreases.

At the other end of the spectrum, for portable appliances, a lower power consumption translates into extended battery life. But the main tendency is the advent of power scavenging devices, which have no external power source, and extract power from the outside world, in the form of light, heat, or vibrations. Here the power budget is more of the order of milliwatts than hundreds of watts. Hence the present-day insistence on low-power digital design.

Low power can be achieved in four ways:

- One can search for low-power technologies and low-power architectures. Reducing the size of the die, or lowering the clock frequency or supply voltage are all techniques that decrease the power consumption.
- One can search for low-power algorithms. Since, for most processors, the energy consumption is proportional to the number of executed operations, this amounts, most often, to find low complexity algorithms.
- One can act at the level of the compiler. The rule here is to classify operations in terms of their power need, and to avoid, as far as possible, those with the highest need. For instance, an external memory access costs much more than a cache access, hence the need for maximizing the hit ratio of the cache. The same reasoning applies to registers.
- Lastly, one can combine the hardware and software approaches. The latest generation of processors and custom devices for embedded systems gives the software some degree of control on power consumption, either by controlling the clock frequency and source voltage, or by disconnecting unused blocks. The best solution would be to let the software or operating system be responsible for these controls.

The Compsys group works in cooperation with CEA-LETI in Grenoble in the field of hardware and software power modeling and optimization.

3.5. Federating Polyhedral Tools

Participants: Christophe Alias, Fabrice Baray [Mentor, Former Compsys Post-Doc], Alain Darte, Antoine Fraboulet, Paul Feautrier, Tanguy Risset.

Present-day tools for embedded system design 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. The French community is a leader in the field, mainly as one of the long-term results of the C^3 cooperative research program.

The polyhedral model is now widely accepted (Inria projects Cosi and A3, now R2D2 and Alchemy, PIPS at École des Mines de Paris, Suif from Stanford University, Compaan at Berkeley and Leiden, PiCo from the HP Labs, the DTSE methodology at Imec, etc.). Most of these groups are research projects, but the increased involvement of industry (Hewlett Packard, Philips) is a favorable factor. Polyhedra are also used in test and certification projects (Verimag, Lande, Vertecs). A very recent development is the interest, shown by several compiler groups, in polyhedral methods (the GCC group, Reservoir Labs in the USA).

Two basic tools that have emerged from this early period are Pip [37] and Polylib [51]. They are currently the only available tools since maintenance has stopped on Omega (Maryland). Their functionalities are parametric integer programming and manipulations of unions of polyhedra. Granting that the showroom effect is important for us (these tools are used in many foreign laboratories), we nevertheless think that maintaining, improving, and extending these tools is a proper research activity. One of our goals must also be the design of new tools for new scheduling/mapping techniques.

In the following, we distinguish between the development of existing tools and the conception and implementation of new tools. These tasks are nevertheless strongly related. We anticipate that most of the new techniques will be evolutions of the present day tools rather than revolutionary developments.

3.5.1. Developing and Distributing the Polyhedral Tools

Recently, we have greatly increased the software quality of Pip and Polylib. Both tools can now use exact arithmetic. A CVS archive has been created for cooperative development. The availability for one year of an ODL software engineer has greatly improved the Polylib code. An interface bridge for combined use of the two tools has been created by Cédric Bastoul (former PhD student of Paul Feautrier). These tools have been the core of new code generation tools [33], [45] widely used in prototyping compilers. Paul Feautrier is the main developer of Pip, while Tanguy Risset has been in charge of coordinating the development of Polylib for several years. Other participants are at Irisa (Rennes) and ICPS (Strasbourg), and also in Lyon and Leiden. In the near future, we contemplate the following actions:

- For Pip, algorithmic techniques for a better control of the size of intermediate values; comparison with commercial tools like Cplex, for the non-parametric part.
- For Polylib, a better handling of $\mathbb{Z}$-polyhedra, which are needed for handling loops with non unit increments.
- For higher-level tools, Cédric Bastoul has developed CLooG, an improved loop generation tool along the lines of Fabien Quilleré's system, which is now available on the Web and is being incorporated into experimental compilers.
- For all these tools, we want to strengthen the user community by participating in the Polylib forum and organizing meetings for interested parties.

3.5.2. New Models

Industry is now conscious of the need for special programming models for embedded systems. Scholars from the University of Berkeley have proposed new models (process networks, SDL, etc.). This has culminated in the use of Kahn process networks, for which a complete overhaul of parallelization techniques is necessary (see Section 3.3.1). Optimizations for memory reduction are also very important. We are developing a tool, based on operations on integral lattices (including Minkowski's successive minima), named Cl@k, that can be used to derive affine mappings with modulo operations for memory reuse (see more details in Section 4.7).

Besides, our community has focused its attention on linear programming tools. For embedded systems, the multi-criteria aspect is pervasive, and this might require the use of more sophisticated optimization techniques (non-linear methods, constraint satisfaction techniques, "pareto-optimal" solutions).

Here again, our leadership in polyhedral tools will make our contributions in these areas easier. We nevertheless expect that, as sometimes in the past, the methods we need have already been invented in other fields like operational research, combinatorial optimization, or constraint satisfaction programming, and that our contribution will be in the selection and adaptation (and possibly the implementation) of the relevant tools.

4. Software

4.1. Introduction

This section lists and briefly describes the software developments conducted within Compsys. Most are tools that we extend and maintain over the years.

4.2. Polylib

Participant: Tanguy Risset.

Polylib (available at http://www.irisa.fr/polylib/) is a C library for polyhedral operations. The library operates on objects such as vectors, matrices, (convex) polyhedra, unions of polyhedra, lattices, $\mathbb{Z}$-polyhedra, and parametric polyhedra. Tanguy Risset has been responsible for the development of Polylib for several years. An ODL software engineer has firmed up the basic infrastructure of the library. The development is now shared between Compsys, the Inria project R2D2 in Rennes, the ICPS team in Strasbourg, and the University of Leiden. This tool is being used by many groups all over the world.

4.3. Pip

Participants: Paul Feautrier, Cédric Bastoul [MCF, IUT d'Orsay].

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. Most of the recent work on Pip has been devoted to solving integer overflow problems by using better algorithms. This has culminated in the implementation of an exact arithmetic version over the GMP library.

Pip is freely available under the GPL at http://www.prism.uvsq.fr/~cedb/bastools/piplib.html. 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).

4.4. MMAlpha

Participant: Tanguy Risset.

Tanguy Risset is the main developer of MMAlpha since 1994 (http://www.irisa.fr/cosi/ALPHA/). The design and development of this software tool was the heart of several PhD theses, and MMAlpha is one of the few available tools for very high-level hardware synthesis (including the design of parallel Asics).

This tool is now in the public domain and has been used in many places (England, Canada, India, USA). Its development is shared between Compsys in Lyon and R2D2 in Rennes. MMAlpha is being evaluated by STMicroelectronics and has been a basis for Compsys participation to European and RNTL calls for proposals.

4.5. Syntol

Participants: Paul Feautrier, Hadda Cherroun [Former member of Compsys].

Syntol is a modular process network scheduler. The source language is C augmented with specific constructs for representing Communicating Regular Process systems (CRP). The present version features a syntax analyzer, a semantic analyzer which is able to identify DO loops in C code, a dependence computer, a modular scheduler, and interfaces for CLooG (see 4.8) and Cl@k (see 4.7). The dependence computer has been extended to handle records (`structures`), the modulo operator in subscripts and conditional expressions. A system for the automatic generation of XML reports has recently been implemented. XML is the preferred format for information exchange between tools.

The next developments should be:

- An extension of the input parser and the scheduler in order to handle a larger subset of C. Cases in point are casts, and disciplined pointers. The introduction of a fixed-point type is also contemplated.
- An extension of the scheduler for handling conditionals.
- Tools for the construction of bounded parallelism schedules – virtual dependences, allocation functions, pseudo arrays.

4.6. SoCLib

Participants: Tanguy Risset, Antoine Scherrer, Antoine Fraboulet.

Compsys has developed a complete network on chip (NOC) prototyping environment in the SoCLib project. This environment can analyze the traffic occurring on the communication medium of a complete SoC, and replay or generate a traffic with similar statistical characteristics. Advanced stochastic analysis have been performed including second order statistics (auto-correlation used for detecting long range dependent behavior). The environment also provides a set of perl script used for fast generation of a SystemC SoC platform and instrumentation of the various SystemC components in order to record network activity. This work is presented in Antoine Scherrer's PhD thesis.

4.7. Algorithms on Integer Lattices and Memory Reuse Module: Cl@k+Bee

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

A few years ago, we identified new mathematical tools useful for the automatic derivation of array mappings that enable memory reuse, in particular the notions of admissible lattice and of modular allocation (linear mapping plus modulo operations). Fabrice Baray, post-doc Inria, developed a tool in 2005-2006, called Cl@k (for Critical LAttice Kernel), that 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.) So far, Cl@k was a stand-alone optimization software tool, with no connections with programs or memory reuse. It has now been plugged by Christophe Alias (also Post-Doc Inria) into ROSE, a source-to-source program transformer, thanks to the development of a lifetime analyzer called Bee. Bee uses ROSE as a high-level parser, analyzes the lifetime of elements of the arrays to be compressed, and builds the necessary input for Cl@k, i.e., the 0-symmetric polytope of conflicting differences. See Section 5.7 for more details.

Cl@k can be viewed as a complement to the Polylib suite, enabling yet another kind of optimizations on polyhedra. Bee is the complement of Cl@k in terms of its application to memory reuse. We believe that Cl@k is going to be used for other not-yet-identified problems, in particular problems for which finding a form of "bounding box" for a polytope is important. As for Bee, it is our first development into ROSE, which may become our future reference platform for implementations of high-level program transformations.

4.8. CLooG: Loop Generation

Participants: Paul Feautrier, Cédric Bastoul [MCF, IUT d'Orsay].

The aim of CLooG is to generate a system of loops that visit once and only once the integer points in the union of several $\mathbb{Z}$-polyhedra. The algorithm is an improved version of a previous effort by Fabien Quilleré (past Inria project Cosi). The code generated by CLooG is compact and quite efficient [26]. The availability of CLooG on the Web as a free software (http://www.cloog.org) has been a triggering factor for a recent increase of interest for the polytope model. Observe for instance the recent implementation of a CLooG/VHDL back-end [34].

4.9. Register Allocation

Participants: Florent Bouchez, Alain Darte, Fabrice Rastello, Cédric Vincent [Former student in Compsys].

Our developments on register allocation with the STMicroelectronics compiler started when Cédric Vincent (bachelor degree, under Alain Darte supervision) developed a complete register allocator in 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. The current implementation explores a two-phases register allocation, with a preliminary spilling phase, followed by a coalescing phase (with the help of SSA). See more details in Section 5.3.

4.10. Procedure Placement

Participants: Fabrice Rastello, Benoit Boissinot.

Within the collaboration with the CEC team at STMicroelectronics, a complete framework of procedure placement for instruction cache optimization have been developed. This tool is decomposed into several parts: first the conflict graph is built from a trace (STMicroelectronics implementation); similarly the affinity graph can also be built from the trace (Fabrice Rastello's implementation); then procedure placement can be performed using graph information (Benoit Boissinot, Fabrice Rastello, and STMicroelectronics implementation); finally the placement can be evaluated using an execution trace or directly on the simulator (STMicroelectronics implementation). All those developments are meant to be open-source. See more details in Section 5.4.

4.11. Modification of the UGH Scheduler

Participants: Alain Darte, Clément Quinson.

The *User Guided High level synthesis* tool is the hardware coprocessor generator of the Disydent framework. This open source environment is developed at the LIP6 laboratory in Paris. It can generate a RTL description of a coprocessor (using the Data Path and Finite State Machine paradigm), from a C description and a *Draft Data Path*. Ugh's principle for high level synthesis (HLS) is that the user is an experienced designer and should be given means to influence the direction the tool takes. It is therefore possible to describe, partially or totally, the datapath organization on which the tool will map scheduled operations.

François Donnet, who developed this scheduler during his PhD, observed that his strategy may fail due to a deadlock. We proved that trying to foresee if a deadlock will appear is a NP-Complete problem. We proposed and implemented an alternative technique, based on register duplication, which solves deadlocks when the scheduler faces one and therefore avoids the complexity of a potentially exponential backtracking or an exponential check for future deadlock appearance. See more details in Section 5.9. We have analyzed the current UGH scheduler and our implementation is almost finished. Our first experiments show that deadlock cases are rare and that performance results from synthesized coprocessors are still acceptable, even with a few duplication.

We plan to include this modified version of UGH's scheduler into the Disydent repository. We are still working on some further developments of this scheduling strategy in order to evaluate the qualities of this approach compared to other lifetime-sensitive scheduling approaches.

5. New Results

5.1. Introduction

This section presents the results obtained by Compsys in 2006. For clarity, some earlier work is also recalled, when results were continued or extended in 2006.

5.2. Optimized Coalescing for Out-of-ssa Conversion

Participants: Florent Bouchez, Benoit Boissinot, Benoit Dupont de Dinechin [STMicroelectronics], Christophe Guillon [STMicroelectronics], Fabrice Rastello.

The SSA form (Static Single Assignment) is an intermediate representation in which multiplexers (called φ functions) are used to merge values at a "join" point in the control graph. The SSA form has very interesting properties that make optimizations like partial redundancy elimination straightforward and efficient. Unfortunately, the SSA form is not machine code and φ functions have to be replaced, at the end of the process, by register-to-register copy instructions on control flow edges. In practice, naive methods for translating out of SSA generate many useless copies (live-range splitting) and additional goto instructions (edge splitting).

Adding copies roughly corresponds to split the live ranges of variables. The reverse transformation is called variable coalescing: coalescing different variables into a common representative variable constrains those variables to be allocated to the same register. Hence, coalescing constrains register allocation and splitting may relax it: in practice, the reduction of register-to-register copies is performed during the register allocation phase. However, we are convinced that coalescing should be performed during the out-of-SSA translation: many more copies can be removed this way with lower complexity; we can keep track of coalesced variables and un-coalesce them when necessary during the register allocator phase.

Coalescing during the out-of-SSA translation has been studied in different contexts. Rastello et al. [47] (*PINNING*) studied the problem for programs represented as native machine instructions, including the use of machine dedicated registers: the heuristic is sophisticated and coalesces variables very aggressively without splitting edges. Budimlić et al. [28] (*D-FOREST*) studied the problem in the context of just in time (JIT) compilers: their "algorithm" is very fast but requires edge splitting; also the coalescing is not very aggressive; finally, their work suffers from a lack of a clean formalization. These two heuristics use a common approach in the sense that they both perform a non-conservative coalescing followed by a repairing phase. Sreedhar et al. [48] (*C*SSA) provided another interesting approach without the above constraints: it is a conservative approach in one pass with basic but efficient ideas that do not require edge splitting. The context of our work is the same as for *D-FOREST*. We have designed a linear heuristic and proved its correctness that aggressively coalesces without splitting edges. For that purpose, we generalized ideas used in all three approaches and developed a new formalization of the problem. Our solution remains to be implemented in the *LAO* code assembly optimizer from STMicroelectronics and compared with other possible approaches.

This work is part of the contract (see Section 6.2) with the CEC team at STMicroelectronics.

5.3. Register Allocation and ssa Form Properties

Participants: Florent Bouchez, Alain Darte, Christophe Guillon [STMicroelectronics], Fabrice Rastello.

The work presented in this section is a joint work with members of the CEC team at STMicroelectronics. It is the natural extension of the work described in Section 5.2. It deals with the following problems related with register allocation: spilling, coalescing, and splitting. Register allocation consists in allocating abstract variables to physical registers, spilling chooses which variables to store in memory when there are too few registers to store all of them, coalescing is used to regroup two variables into one when possible, hence reducing the number of variables, and splitting does the converse. Understanding the interactions between these three techniques is the heart of our work.

Register allocation Register allocation is one of the most studied problems in compilation. It is considered as an NP-complete problem since Chaitin et al., in 1981, modeled the problem of assigning temporary variables to k machine registers as the problem of coloring, with k colors, the interference graph associated to the variables. The fact that the interference graph can be arbitrary proves the NP-completeness of this formulation. However, this original proof does not really show where the complexity of register allocation comes from. Recently, the re-discovery that interference graphs of SSA programs can be colored in polynomial time raised the question: Can we use SSA to do register allocation in polynomial time, without contradicting Chaitin et al's NP-completeness result? To address such a question and, more generally, the complexity of register allocation, we have revisited Chaitin et al's proof to identify the interactions between spilling, coalescing/splitting, critical edges, and coloring. Our study shows that the NP-completeness of register allocation is *not* due to the coloring phase, as may suggest a misinterpretation of the reduction of Chaitin et al. from graph k-coloring. If live-range splitting is taken into account, deciding if k registers are enough or if some spilling is necessary is not as hard as one might think. The NP-completeness of register allocation is due to three factors: the presence of critical edges or not, the optimization of spilling costs (if k registers are not enough) and of coalescing costs, i.e., which live-ranges should be fused while keeping the graph k-colorable. These results have been presented at WDDD [7] and LCPC [8].

Critical edges A critical edge is an edge of the control flow graph whose predecessor block has more than one successor and whose successor block has more than one predecessor. Forbidding live-range splitting on critical edges makes the coloring problem NP-complete. Similarly the presence of critical edges makes the translation out of SSA form difficult (see Section 5.2) and most existing approaches are not able to handle code with critical edges. However, some real codes (supported by some compilers) do contain critical edges that the compiler is not able to split (abnormal edges). Also, splitting an edge has a cost, and it might be preferable not to split an edge of high execution frequency. We have introduced a new technique called parallel-copy motion. Basically, the copies related to a split can be moved backward or forward the edge; a parallel-copy moved backward on the predecessor block should be compensated by a reverse parallel copy on the other outgoing edges of this block; the process has to be iterated and of course could loop and lead to an inconsistency. Determining whether this inconsistency is avoidable or not is precisely where the hardness of coloring comes from. Our experiments have shown that, in practice, for the benchmark suites available from our collaboration with STMicroelectronics, this problem can be solved in polynomial time and critical edges can be efficiently handled this way. We are currently writing a research report presenting these results.

Spilling Minimizing the amount of spilled variables is a highly-studied problem in compiler design; it is an NP-complete problem in general. Besides, an important consequence of our study (see [8] and the previous paragraph on register allocation) is that the goal of the spilling phase simply relies on lowering the register pressure (number of variables alive at this program point) at every program point, and the optimization problem corresponds to minimizing the spilling cost. The question raised by this important remark was: is it easier to solve the spilling problem under SSA? This would lead to register allocation heuristics by transforming a program to SSA, spilling the variables, allocating them with the exact number of registers and then trying to go out of SSA. The spilling can be considered at different granularity levels: the highest, called "spill everywhere", corresponds to consider the live range of each variable entirely: a spilled variable will result in a store after each definition point and a load before each use point. The finer granularity, called "load-store optimization", corresponds to optimize each load and store separately. This accurate formulation, also known as "paging with write back", is NP-complete [36] even for a basic block in SSA form. However, the coarser view (spill everywhere) is much simpler and can be solved in polynomial time [27] for a SSA basic block. What about more general graph structures? We have studied in details the spill everywhere formulation under

SSA form. We have considered different instances of the problem depending on the machine model and also depending on the relative values of register pressure and number of registers. We found that most instances are NP-complete. The complexity almost always comes from the number of registers and the number of arguments by instruction (for RISC machines) but not from the register pressure. We are currently writing a report on these results.

Coalescing To finish with, the last question is related to the coalescing problem: how can we minimize or at least reduce the number of generated copies? We have distinguished several optimizations that occur in most coalescing heuristics: a) aggressive coalescing removes as many moves as possible, regardless of the colorability of the resulting interference graph; b) conservative coalescing removes as many moves as possible while keeping the colorability of the graph; c) incremental conservative coalescing removes one particular move while keeping the colorability of the graph; d) optimistic coalescing coalesces moves aggressively, then gives up about as few moves as possible so that the graph becomes colorable. We have almost completely classified the NP-completeness of these problems, discussing also on the structure of the interference graph: arbitrary, chordal, or k-colorable in a greedy fashion. These results will be presented at CGO'07 [9]. AS for experiments, we already have promising results with heuristics that outperform previously proposed techniques.

These developments are part of two contracts (see Section 6.1 and Section 6.2) with the CEC team at STMicroelectronics and form the theoretical foundations of our implementations in the STMicroelectronics compiler.

5.4. Instruction Cache Optimization

Participants: Thierry Bidault [STMicroelectronics], Benoit Boissinot, Christophe Guillon [STMicroelectronics], Fabrice Rastello, Éric Thierry [Lip, MC2 team].

The instruction cache is a small memory with fast access. All binary instructions of a program are executed from it. In the ST220 processor from STMicroelectronics, the instruction cache is direct mapped: the cache line i can hold only instructions whose addresses are equal to i modulo L, where L is the size of the cache. When a program starts executing a block that is not in the cache, one must load it from main memory; this is called a cache miss. This happens either at the first use of a function (cold miss), or after a conflict (conflict miss). There is a conflict when two functions share the same cache line; each of them removes the other from the cache when their executions are interleaved. The cost of a cache miss is of the order of 150 cycles for the ST220, hence the interest of minimizing the number of conflicts by avoiding line sharing when two functions are executed in the same time slot. For this problem we have considered the three following objective functions:

COL Minimizing the number of conflicts for a given execution trace. This can be reduced to the *Max-K-Cut* and *Ship-Building* problems.

EXP Minimizing the size of the code. This is equivalent to a traveling salesman problem (building an Hamiltonian circuit) on a very special graph called *Cyclic-Metric*.

NBH Maximizing code locality at cache line granularity is the neighboring problem. This can be reduced to the traveling salesman problem.

We have proved several non-approximability and polynomiality results related to the *COL*, *EXP*, and *NBH* problems. For the *COL* problem, Gloy and Smith (GS) proposed a greedy heuristic based on a conflict graph called the temporal relationship graph (TRG). *EXP* is not taken into consideration. Pettis and Hansen (PH) proposed a greedy fusion without code size expansion to enhance locality. As a side effect, for small codes (compared to cache size) PH also reduces cache conflicts. We proposed several algorithmic improvements to these approaches: we have lowered the complexity of GS by a factor varying on our benchmark suite from 50 to 500; in [41] we had proposed a modified version of GS that takes *EXP* into account; we have improved this heuristic (B); we have also developed a heuristic solution (FILLGAP) that provides no code size expansion at all; finally, we have defined an affinity graph that reflects the *NBH* problem and implemented a more aggressive heuristic than PH to optimize the *NBH* problem. Depending on the structure of the conflict graph, we choose one among those three strategies. The code size expansion obtained this way is usually less than 2 percents and the cache conflict rate gains in average 5 percents compared to the best known algorithms.

We also worked on a completely different approach whose goal is to provide a procedure placement that yields *no* code size expansion and takes *NBH* into account. Our algorithm still has to be implemented. Finally, we currently work on the following problem, but with no satisfactory answer yet:

Profiling vs static The GS algorithm is based on a conflict graph built using profiling. In practice, it leads to a much better conflict reduction than any other known static based approach. But a profiling technique has several drawbacks: the compilation process of profiling feedback is too complex for most application developers. Also, using good representative data sets for profiling is an important issue. The problem is then to be able to build a TRG using fully static, fully dynamic, or hybrid techniques.

This work is a continuation of the work done in the previous contract with the CEC team at STMicroelectronics, see Section 6.1.

5.5. On-Chip Traffic Analysis

Participants: Patrice Abry [ENS-Lyon, Physics Lab], Antoine Fraboulet, Tanguy Risset, Antoine Scherrer.

This work is part of the PhD of Antoine Scherrer, co-funded by STMicroelectronics and the CNRS. The goal is to (semi-)automatically generate a traffic generator from an execution trace of a SoC IP. This traffic generator can then be used to prototype NOC performances in various configurations. We work on this topic in relation with STMicroelectronics. We are also working in collaboration with Patrice Abry, from the Physics laboratory of ENS-Lyon. Patrice Abry (http://perso.ens-lyon.fr/patrice.abry/) is a worldwide expert in traffic modeling with self-similar processes. Many results have been published in 2006 on this topic.

First, the analysis and synthesis of long range dependent processes are now fully operational. These tools have been applied to network anomalies detection, a work published in IEEE Transactions on Dependable and Secure Computing (TDSC) [4]. The technique developed for long range dependence synthesis is potentially applicable to many scientific domains.

Second, the complete NOC prototyping environment is now operational. It has been implemented in the cycle-accurate SoC simulation environment SoCLib. It consists in:

- A generic, programmable, SystemC traffic generator that can be used to replace SystemC models of a processor IP;
- A set of `perl` scripts that can quickly generate the SystemC interconnection files used to describe a given MPSOC platform;
- A VCD trace parser used to produce data information from raw traffic traces (mainly network traffic traces: latency, throughput, size, etc.)
- Libraries used to analyze stochastic processes (obtained from network traffic traces) and also to synthesize traffic according to chosen statistical properties (latency with given mean and given second order statistics for instance).
- A *phase detection algorithm* used to split the traffic generated by an IP in several phases, each phase with a stationary stochastic behavior.

All these components have been integrated in a semi-automatic NOC *prototyping environment*. The designer only manually specifies the organization of the platform to be simulated (in a user-friendly format) and the choice of generation: traces are replayed or a stochastic traffic (according to some specified statistical characteristics) is generated. Traffic analysis has been performed on embedded applications (MP3, MPEG, JPEG), running on a multi-processor SoC interconnected with the Spin or DSpin NOC (developed at LIP6). The original features of this environment are:

1. The availability in such a cycle-accurate simulator of a complete set of tools to analyze and synthesize traffic with particular second order statistical properties.
2. The automatic decomposition of a given traffic trace in phases with stationary statistical behavior (using a technique developed for software prototyping).
3. The fact that the traffic generator can be configured *independently of the network* (it is latency aware), which makes possible to program the traffic generator from a trace obtained on a simple platform (no NOC) and use this program for all the NOC configurations to be tested. This features improves the speed of the complete prototyping flow.

These experiments have been published in ASAP [16] and CODES/ISSS [17]. The complete work is described in Antoine Scherrer's PhD thesis.

5.6. Loop Transformations for High Level Synthesis

Participants: Alain Darte, Alexandru Plesco, Clément Quinson, Tanguy Risset.

We have started a study on the use of a loop transformation front-end to high level synthesis HLS tools. The study is based on the Wrapit loop transformation tools developed by the Alchemy team project and integrated into the ORC open-source compiler. This tool allows the user to identify part of C programs which are loops with static control, to indicate many loop transformations (fusion, code motion, strip mining, etc.) and to generate back a C program where the loops have been transformed. The Wrapit tool has been applied to C code, synthesized by the Spark HLS tool, showing important performance improvements of the resulting design (in terms of silicon area and communication optimization). This work was done by Alexandru Plesco during his Master, it is currently under publication.

The topic of using compiling techniques as front-end optimizations to HLS tools is currently very hot. In parallel to this study, Clément Quinson has spent several months at STMicroelectronics (synthesis team in Crolles) to analyze what program transformations are required on source codes so that they can be accepted as input of HLS tools and achieve a good enough *Quality of Result*, i.e., good results both in terms of area and performance when compared to hand-written HDL. He worked with an industrial HLS tool on several applications from STMicroelectronics, that were so far not accepted or that required transformations to get acceptable results. These – still preliminary – experiments give quantitative and qualitative results from a leading edge commercial tool on real-life applications, which represents a valuable feedback for our research. These manually-applied transformations could in the near future be automated or, at least, semi-automated, guided by the user. The “painful” part that can be automated would then be done by a compiler or, more precisely, a high-level source-to-source transformer.

5.7. Memory Reuse and Modular Mappings

Participants: Christophe Alias, Fabrice Baray [Mentor, Former Post-Doc in Compsys], Alain Darte, Rob Schreiber [HP Labs], Gilles Villard [Lip, Arénaire Project].

When designing hardware accelerators, one has to solve both scheduling problems (when is a computation done?) and memory allocation problems (where is the result stored?). This is especially important because most of these designs use pipelines between functional units (this appears in the code as successive loop nests), or between external memory and the hardware accelerator. To decrease the amount of memory, the compiler must be able to reuse it. An example is image processing, for which we might want to store only a few lines and not the entire frame, data being consumed quickly after they are produced. A possibility to reduce memory size is to reuse the memory locations assigned to array elements, following, among others, the work of Francky Catthoor's team [32], of Lefebvre and Feautrier [44], and of Quilleré and Rajopadhye [46].

In our previous work, we introduced a new mathematical formalism for array reuse, using the concept of critical lattice, so as to understand and analyze previous approaches. We found that they are all particular cases of more general heuristics for finding a critical lattice. We also proved that we can compute approximations that do not differ from the optimum more than by a multiplicative constant that depends only on the dimension of the problem. In 2004-2005, we continued our theoretical study. We analyzed in more detail the strengths and weaknesses of the previous approaches for array reuse, we revealed similarities and differences with early 70's and 80's work on data layouts allowing parallel accesses, we explored more deeply the properties of linear mappings [31]. We also developed the algorithms on lattices, successive minima, and critical lattices, needed to implement our memory reuse strategies (see Section 4.7). The resulting tool Cl@k, developed by Fabrice Baray, should impact both the practical problem of designing hardware accelerators and the mathematical problem of finding the critical lattice of an object. It is built on top of our present tools Pip and Polylib, and is a perfect extension for our suite of tools on polyhedral/lattice manipulations.

So far, Cl@k was a stand-alone combinatorial optimization tool, with no connections with memory reuse. In 2006, we designed all the algorithms for the program analysis required to use lattice-based memory allocation in real applications. The resulting tool, developed by Christophe Alias, is called Bee. It uses the source-to-source transformer ROSE, developed by Dan Quinlan at the Livermore National Laboratories, first to collect (as pollen, thus the name Bee) all the necessary information on the lifetime of array elements in the program and, second, to inject the memory allocation found by Cl@k into ROSE and generate the code after memory reduction. Bee is developed in C++ and extensively uses integer linear programming (ILP) and polyhedra manipulations (with Polylib) to extract from the program the set of conflicting differences, which is the input for Cl@k. Our first experiments with benchmarks borrowed from IMEC, thanks to Philippe Clauss, Sven Verdoolaege, and Florin Balasa, show excellent results. Many arrays can be contracted, some can even be transformed into scalars, and this, even without any particular loop re-ordering, which is quite surprising. The second observation is that running times are acceptable, and again surprisingly, that the complex algorithms involved in Cl@k are much cheaper than the program analysis itself. This analysis is very close to standard data dependence analysis but with a slight difference, which turns out to be more important, in practice, than one could think at first sight. A way to compute the set of conflicting differences is to compute for each array location the first write (indeed similar to dependence analysis) and the last read (symmetric situation). But in real-life programs, reads and writes are not symmetric: there are often many more reads than writes, which makes the computation of last reads more expensive. As future work, we need to work more deeply on three aspects:

- We need to develop tricks for particular cases to speed-up the computation of last reads and not rely on ILP for all reads. This is an important practical problem.
- We need to develop strategies for designing *parametric* modular allocations. The computation of the set of conflicting differences is parametric in the current implementation of Bee, but not all algorithms/heuristics in Cl@k are parametric, in particular the computations of successive minima. This is a very challenging theoretical problem.
- For the moment, our method is limited to programs with polyhedral iteration domains and affine array index functions (static control programs). We first need an accurate slicing technique to extract the static control parts that can be handled by our method and an approximation method when index functions are not affine. A more challenging problem is to extend our method to programs with general control flow. This would require a dependence analysis as accurate as possible and perhaps the definition of an approximated "order of computations".

The work on Cl@k and Bee in currently under submission.

5.8. Modular Scheduling

Participant: Paul Feautrier.

Scheduling of ordinary programs is a highly non scalable process. We estimate that the scheduling time grows as the sixth power of the program size. We have designed a new scheduling method, which uses projections instead of linear programming, and which is both scalable and structured. This work has been published in the International Journal of Parallel Programming [1].

Our method is scalable because the scheduling proceeds by successive elimination of statements from the relevant subset of the dependence graph. Hence, it is almost linear in the program size. It is still exponential in the loop nest depths, but these factors are very small integers. It is structured because the application can be split in a hierarchical process network, and because each process can be scheduled independently of the others.

The prototype modular scheduler Syntol is now completed and may be used for experiments. It has been interfaced with CLooG (see Section 4.8) for code generation. We are using Syntol for the following tasks:

- Writing kernels for real-life applications, especially in the field of digital TV. The current parser needs to be extended, especially as regards fixed-point arithmetics and pointers.
- Investigating the question of real-time constraints, in the case of non-unit execution time operations and multidimensional schedules.
- Investigating heuristics for handling resource constraints.
- Creating a VHDL back-end targeted at FPGA chips.
- Imposing additional constraints on schedules in order to improve code generation.

5.9. Scheduling for Synthesis

Participants: Hadda Cherroun, Alain Darte, Paul Feautrier, Alain Darte, Clément Quinson.

While scheduling for high-performance computations can be done at a somewhat abstract level (the back-end compiler takes care of detail), in circuit synthesis all elementary steps (e.g., address computations and memory accesses) must be taken into account. The resulting scheduling problems are typically too large to be solved even by the methods of Section 5.8. Hence the idea of a two-levels scheduling, in which the *fronts* generated by the first-level schedule are refined using combinatorial optimization methods.

We have designed and compared several Branch & Bound algorithms, and also a very fast but approximate greedy scheduling algorithm. The fastest exact Branch & Bound algorithm is based on variants of Bellman-Ford and Dijkstra's algorithm, using a reweighting technique similar in the spirit to Johnson's algorithm. We point out that this technique is also nothing but a “retiming” technique (in the sense of Leiserson and Saxe), technique that we explored in the past for several program optimizations, including software pipelining, loop fusion, and array contraction. The results of this work have been presented at DATE'06, in March 2006 [12] and is the main part of Hadda Cherroun's work. A journal paper is in preparation.

Another work we did on scheduling for synthesis is to examine more deeply the very particular scheduling approach used in UGH, a user-guided synthesis tool developed at LIP6 (see the presentation in Section 4.11). The user constrains the scheduler with a pre-allocation: some scalar variables are identified as registers and the scheduler must respect this allocation. However, the scheduler can relax the output dependences that exist between different writes of a given variable, as long as the semantics of the code is preserved. This strategy has a strong influence on the scheduling process. Anti dependences are not all positioned in the initial dependence graph, they are instead added to the graph after one of the multiple assignments to a register is chosen to occur first. Unfortunately, as some registers are already allocated, dependence and resource constraints may lead to a deadlock.

We proved that determining if a deadlock will appear when choosing one particular assignment is a NP-Complete problem. In other words, even if there exists a positive instance of the register allocation problem (given by the original sequential description of the program), the problem to determine if a modification of the sequence of assignments to registers is hard. This proof confirms that mechanisms already implemented in UGH are not sufficient. We proposed register duplication as a way to solve to the deadlock issue. We do not try to foresee deadlocks, which would be an exponential tests unless P=NP, we do not try to backtrack, which could be exponential too, but we allocate more registers when the scheduler encounters a deadlock, so as to relax constraints. This has been successfully implemented in UGH.

This work still has to be published and completely integrated into the main UGH distribution, which is open source. We plan to extend our results to more general situations, so as to take advantage of an early influence from resource constraints on the scheduling process. Comparisons with alternative lifetime-sensitive scheduling need to be done too.

5.10. Optimization for Low Power

Participants: Yves Durand [CEA-LETi], Paul Feautrier, Nicolas Fournel, Antoine Fraboulet, Philippe Grosse.

There are many contributors to the energy consumption of an embedded system: embedded processors, special purpose accelerators, memory banks, peripherals and the network that interconnect all of them. Our first task has been to investigate the relative magnitude of these contributions, both by simulation (Ph. Grosse) and by physical measurements on an ARM development platform (N. Fournel).

Simulation of a fourth generation radio modem has shown that the most important contributions come from hardware accelerators and external memory. The network contributes for a small constant, and the consumption of the peripherals (mainly the radio interface) is beyond our capabilities to optimize. These results were obtained by augmenting a VHDL simulation with a power estimator and have been presented at the PATMOS workshop [14]. A VHDL description of the appliance is necessary; hence, they cannot be applied to embedded processors.

Direct physical measurements on a VLSI chip need specialized equipment. In contrast, such measurements are easier on a development platform, and were implemented by N. Fournel with help from the electronic laboratory of INSA, Cegely. The result of these measurements is a model of the power consumption of a processor, its cache, scratch pad, and external memory. The influence of the clock frequency has been measured, while the influence of voltage scaling had to be extrapolated. The resulting model has been coupled to an instruction set simulator; this combination allows the prediction of the energy budget of quite complex applications and also of operating system services like task switching or synchronization. Application to realistic image processing applications has shown that the cumulative error – power model error plus simulator approximations – is less than ten percent. Two papers have been submitted on these results.

The application that runs in a fourth-generation modem – software radio – can be modeled as a synchronous data flow (SDF) system of processes. From this model, one can deduce the operating frequency of each process from the throughput and latency constraints of the application. If one assumes that the chip has the necessary controls, one can adjust the per bloc voltage and frequency for minimal energy consumption under performance constraints. We have shown that one can obtain spectacular power reductions in this way, amounting in some cases to a factor of 2. Two papers are in preparation concerning this approach.

6. Contracts and Grants with Industry

6.1. Contract with stmicroelectronics on Register Allocation and Instruction Cache Optimizations

Participants: Alain Darte, Fabrice Rastello, Florent Bouchez, Benoit Boissinot.

This contract funded by STMicroelectronics finished in December 2005, even if our research continues on this topic. Its objective was the improvement of existing techniques for the optimization of the instruction cache use and for the optimization of the register allocation phase. The study targets ST200 architectures, for which the CEC team develops a compiler tool chain. Work on instruction cache optimizations are described in Section 5.4, studies on register allocation in Sections 5.2 and 5.3.

6.2. Minalogic SCEPTRE project with stmicroelectronics on SSA, Register Allocation, and JIT Compilation

Participants: Alain Darte, Fabrice Rastello, Florent Bouchez, Benoit Boissinot.

This contract started in October 2006 is part of the “pôle de compétitivité” MINALOGIC. The collaboration deals with the possible applications of the SSA form for program optimizations, including JIT and dynamic compilation for media processors. It concerns predication, register allocation, and instruction selection. Related work on register allocation and JIT constraints are described in Section 5.2 and 5.3.

6.3. Minalogic Open-TLM project

Participants: Tanguy Risset, Antoine Fraboulet.

Compsys participates to the MINALOGIC Open-TLM project, which was accepted at the end of 2006 (4 years project). MINALOGIC is the “pôle de compétitivité” of Grenoble on Micro- and Nano-technologies. The goal of this project is to build open-source SystemC SoC simulators at the transition level (TLM, i.e., the accurate time is not modeled, only the transactions between the various components are simulated) in order to provide a tool for embedded software prototyping on various SoC architectures. STMicroelectronics, Thompson, and silicomps are participating in this project. Compsys is present in the *Middleware* package and is expected to build a TLM platform emulating an existing multimedia embedded device such as a pocket PC and to study code optimization and middleware integration on such a platform.

7. Other Grants and Activities

7.1. ITEA Project

Participants: Paul Feautrier, Antoine Fraboulet.

Compsys, mainly through Paul Feautrier and Antoine Fraboulet, is involved in the Martes (Model driven Approach to Real-Time Embedded Systems development) ITEA project. The project has been accepted by the EC authority in 2004, and the French Ministry has decided to fund it, starting in 4Q 2005. The French partners have already held several technical meetings. Ouassila Labbani has been hired as a postdoctoral engineer. The Compsys participation to MARTES is focusing on the use of process networks for designing embedded systems. Items of interest are:

- Control in process systems.
- Automatic extraction of data movement information from C programs.

7.2. CNRS Convention with the University of Illinois at Urbana-Champaign (USA)

Participant: Paul Feautrier.

A convention between UIUC and CNRS supports some visits and cooperation between David Padua’s team, Compsys and members of the Alchemy project.

7.3. Inria Support for Collaboration with Federal University, Rio Grande Do Sul (Brasil)

Participant: Antoine Fraboulet.

An Inria grant supported this year a cooperation visit between Luigi Carro’s team and Compsys.

7.4. Informal Cooperations

- Fabrice Rastello has contacts with Saman Amarasinghe's team at the MIT (Boston, USA) to work on StreamIt. He spent one week in December 2006 at MIT in this context.
- Tanguy Risset is in regular contact with the University of Québec at Trois-Rivières (Canada), where MMAlpha is in use.
- Compsys is in regular contact with Sanjay Rajopadhye's team at Colorado State University (USA).
- Compsys is in regular contact with Francky Catthoor's team in Leuwen (Belgium) and with Ed Depreterre's team at Leiden University (the Netherlands).
- Alain Darte has fruitful relations with Rob Schreiber at HP Labs – with several joint patents [30], [29] and publications – and some contacts with the past members of the PiCo team such as Scott Mahlke at the University of Michigan. A more formal collaboration is planned for possible joint work on the high-level synthesis tool developed by Scott Mahlke's team.
- Paul Feautrier has regular contact with Zaher Mahjoub's team in the Faculté des Sciences de Tunis, notably as the co-advisor of a PhD student.
- Compsys is in regular contact with Christine Eisenbeis and Albert Cohen (Inria project Alchemy, Paris), with François Charot and Patrice Quinton (Inria project R2D2, Rennes), with Alain Greiner (Asim, LIP6, Paris), and Frédéric Pétrot (TIMA, Grenoble).
- Compsys participates in the center of microelectronics design Inria/CEA with LETI (CEA Grenoble) on software techniques for power minimization.
- Compsys participates in the EmSoC research project, which is part of the new research clusters of the Rhône-Alpes region.
- Compsys is in regular contact with Luigi Carro's team (UFRGS, Porto Alegre, Brasil).
- Compsys, as some other Inria projects, is involved in the network of excellence HIPEAC (High-Performance Embedded Architecture and Compilation http://www.hipeac.net/).

8. Dissemination

8.1. Introduction

This section lists the various scientific and teaching activities of the team in 2006.

8.2. Conferences and Journals

- In 2006, Alain Darte was member of the program committees of ASAP 2006 (IEEE International Conference on Application-Specific Systems, Architectures and Processors), CASES 2006 (ACM International Conference on Compilers, Architecture, and Synthesis for Embedded Systems), DATE 2006 and DATE 2007 (Design, Automation and Test in Europe), and CGO 2006 (4th Annual International Symposium on Code Generation and Optimization). He is member of the steering committee of the workshop series CPC (Compilers for Parallel Computing). He is member of the editorial board of the international journal ACM Transactions on Embedded Computing Systems (ACM TECS).
- Paul Feautrier is associate editor of Parallel Computing and the International Journal of Parallel Computing. In 2006, he was a member of the program committee of POPL'06 in Charleston, USA.
- Tanguy Risset is a member of the editorial board of Integration: the VLSI Journal. In 2006, he was a member of the program committee of ASAP 2006 and will be member of the program committee of ASAP 2007.

8.3. Teaching and Thesis Advising

- In 2006-2007, Paul Feautrier is teaching a Master 2 course on advanced compilation and program transformations and the compilation course to Master 1 students at ENS-Lyon. Tanguy Risset and Antoine Fraboulet teach a Master 2 course entitled "Design of embedded computing systems" at Insa-Lyon. Tanguy Risset is in charge of the algorithmic and programming course at Insa-Lyon.
- Paul Feautrier is thesis advisor of Yosr Slama (Faculté des Sciences de Tunis, co-advisor Mohamed Jemni, defended in December 2006) and Philippe Grosse (co-advisor Yves Durand, CEA-LETI). Hadda Cherroun has received a French-Algerian grant for an internship of 18 months with Compsys, under Paul Feautrier's supervision, in preparation for an Algerian PhD. Paul Feautrier and Antoine Fraboulet are thesis co-advisors of Nicolas Fournel. Antoine Fraboulet and Tanguy Risset are thesis co-advisors of Antoine Scherrer (defended in December 2006). Alain Darte and Tanguy Risset are thesis co-advisors of Alexandru Plesco. Alain Darte is thesis advisor of Clément Quinson. Alain Darte and Fabrice Rastello are thesis co-advisors of Florent Bouchez. Fabrice Rastello is thesis advisor of Benoit Boissinot.

8.4. Teaching Responsibilities

- Alain Darte is the vice-president of the admission exam to ENS-Lyon, responsible for the "Computer Science" part.
- Paul Feautrier is in charge of the second year of the Computer Science Master of ENS-Lyon.
- Tanguy Risset and Antoine Fraboulet are currently setting up, with some other people, a new course on embedded systems targeted to engineers from different departments: electrical engineering, network, and computer science.

8.5. Animation

- Paul Feautrier is a member of the PhD committee of ENS-Lyon, and of the hiring committees of ENS-Lyon and Université Joseph Fourier.
- Tanguy Risset is in charge of the Polylib mailing-list. This list includes most of the actors on the polyhedral models.
- Alain Darte is member of the national evaluation commission (CE) of INRIA.
- Antoine Fraboulet is a member of the hiring committees of Insa-Lyon.

8.6. Defense Committees

- Tanguy Risset was a member of the defense committee for Tariq Ali Omar (September 2006, Grenoble), Samuel Evain (November 2006, Lorient), and Richard Buchmann (December 2006, Paris)
- Alain Darte was "principal opponent" for the PhD defense of Andrzej Bednarski entitled "Integrated Optimal Code Generation for Digital Signal Processors", Linköping University, Sweden, June 2006.
- Paul Feautrier was reviewer for the HDR (professorial thesis) of Bernard Pottier (Université de Bretagne Occidentale), the PhD thesis of Pascal Gomez (UPMC) and of Rachid Seghir (Strasbourg, December 2006), and a member of the defense committee of Yosr Slama (UVSQ / Faculté des Sciences de Tunis) and Antoine Scherrer (ENS-Lyon).

8.7. Workshops, Seminars, and Invited Talks

(For conferences with published proceedings, see the bibliography.)

- Fabrice Rastello was invited to give talks on register allocation at the UCLA in March 2006 and at the MIT in June 2006.
- Alain Darte gave talks on register allocation in June 2006 at Linköping University, Sweden (Christoph Kessler's group) and at Princeton University, USA (Wayne Wolfe's group) and, in September 2006, in Grenoble, France (Laboratoire ID, Denis Trystram's group).

9. Bibliography

Year Publications

Articles in refereed journals and book chapters

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

[2] P. FEAUTRIER. *Les Compilateurs*, in "Encyclopédie de l'Informatique", J.-E. PIN (editor). , to appear, Vuibert, 2007.

[3] A. FRABOULET, T. RISSET. *Master Interface for On-Chip Hardware Accelerator Burst Communications*, in "Journal of VLSI Signal Processing", to appear, 2007.

[4] A. SCHERRER, N. LARRIEU, P. BORGNAT, P. OWEZARSKI, P. ABRY. *Non Gaussian and Long Memory Statistical Characterisations for Internet Traffic with Anomalies*, in "IEEE Transactions on Dependable and Secure Computing (TDSC)", to appear, 2007, http://perso.ens-lyon.fr/antoine.scherrer/scherrer-tdsc.pdf.

Publications in Conferences and Workshops

[5] P. AMIRANOFF, A. COHEN, P. FEAUTRIER. *Beyond Iteration Vectors: Instancewise Relational Abstract Domains*, in "Static Analysis Symposium (SAS'06), Seoul, Corea", August 2006.

[6] P. BORGNAT, N. LARRIEU, P. OWEZARSKI, P. ABRY, J. AUSSIBAL, L. GALLON, G. DEWAELE, N. NOBELIS, L. BERNAILLE, A. SCHERRER, Y. ZHANG, Y. LABIT. *Détection d'attaques de dénis de service par un modèle non gaussien multirésolution*, in "Colloque francophone sur l'ingénierie des protocoles (CFIP), Tozeur, Tunisie", November 2006.

[7] F. BOUCHEZ, A. DARTE, C. GUILLON, F. RASTELLO. *Register Allocation: What does the NP-Completeness Proof of Chaitin et al. Really Prove?*, in "5th Annual Workshop in Duplicating, Deconstructing, and Debunking (WDDD'06), held in conjunction with the 33rd International Symposium on Computer Architecture (ISCA-33), Boston, USA", July 2006.

[8] F. BOUCHEZ, A. DARTE, C. GUILLON, F. RASTELLO. *Register Allocation: What does the NP-Completeness Proof of Chaitin et al. Really Prove? Or Revisiting Register Allocation: Why and How*, in "19th International Workshop on Languages and Compilers for Parallel Computing (LCPC'06), New Orleans, USA", November 2006.

[9] F. BOUCHEZ, A. DARTE, F. RASTELLO. *On the Complexity of Register Coalescing*, in "International Symposium on Code Generation and Optimization (CGO'07)", to appear, IEEE Computer Society Press, March 2007.

[10] G. CHELIUS, A. FRABOULET, E. FLEURY. *Demonstration of Worldsens: A Fast Prototyping and Performance Evaluation Tool for Wireless Sensor Network Applications & Protocols*, in "Second International Workshop on Multi-hop Ad Hoc Networks: From Theory to Reality (REALMAN), Firenze, Italia", ACM, May 2006, p. 131 – 133.

[11] G. Chelius, A. Fraboulet, E. Fleury. *Worldsens: A Fast and Accurate Development Framework for Sensor Network Applications*, in "The 22nd Annual ACM Symposium on Applied Computing (SAC 2007), Seoul, Korea", to appear, ACM, March 2007.

[12] H. Cherroun, A. Darte, P. Feautrier. *Scheduling under Resource Constraints using Dis-Equalities*, in "Design Automation and Test Europe (DATE'06)", March 2006.

[13] A. Fraboulet, G. Chelius, E. Fleury. *WorldSens: System Tools for Embedded Sensor Networks*, in "Real-Time Systems Symposium (RTSS 2006) (Work in Progress), Rio de Janeiro, Brasil", IEEE, December 2006.

[14] P. Grosse, Y. Durand, P. Feautrier. *Power Modelling of a NoC Based Design for High-Speed Telecommunication Systems*, in "16th PATMOS Workshop - International Workshop on Power And Timing Modeling, Optimization and Simulation, Montpellier, France", September 2006.

[15] S. Rus, G. He, C. Alias, L. Rauchwerger. *Region Array SSA*, in "15th International Conference on Parallel Architectures and Compilation Techniques (PACT'06), Seattle, WA, USA", September 2006.

[16] A. Scherrer, A. Fraboulet, T. Risset. *A Generic Multi-Phase On-Chip Traffic Generation Environment*, in "IEEE 17th International Conference on Application-Specific Systems, Architectures and Processors (ASAP'06), Steamboat Springs, Colorado, USA", September 2006.

[17] A. Scherrer, A. Fraboulet, T. Risset. *Automatic Phase Detection for Stochastic On-Chip Traffic Generation*, in "Proceedings of the 4th International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS), seoul, South Corea", ACM Press, October 2006, p. 88 – 93.

[18] A. Scherrer, N. Larrieu, P. Borgnat, P. Owezarski, P. Abry. *Non Gaussian and Long Memory Statistical Modeling of Internet Traffic*, in "4th International Workshop on Internet Performance, Simulation, Monitoring and Measurement (IPS MOME), Salzbourg, Austria", March 2006.

[19] A. Scherrer, N. Larrieu, P. Borgnat, P. Owezarski, P. Abry. *Une caractérisation non gaussienne et longue mémoire du trafic Internet et de ses anomalies*, in "5th Conference on Security and Network Architectures (SAR), Seignosse, France", June 2006.

Internal Reports

[20] F. Bouchez, A. Darte, F. Rastello. *On the Complexity of Register Coalescing*, Technical report, n^{o} RR2006-15, LIP, ENS-Lyon, France, March 2006, http://www.ens-lyon.fr/LIP/Pub/rr2006.php.

[21] F. Bouchez, A. Darte, F. Rastello. *Register Allocation: What does Chaitin's NP-Completeness Proof really Prove?*, Technical report, n^{o} RR2006-13, LIP, ENS-Lyon, France, March 2006, http://www.ens-lyon.fr/LIP/Pub/rr2006.php.

[22] N. Fournel, A. Fraboulet, P. Feautrier. *Booting and Porting Linux and uClinux on a New Platform*, Technical report, n^{o} RR2006-08, ENSL/LIP, February 2006.

[23] N. Fournel, A. Fraboulet, P. Feautrier. *Porting the Mutek Operating System to ARM Platforms*, 34 pages, Research Report, n^{o} 2006-12, LIP, ENS-Lyon, February 2006.

Miscellaneous

[24] G. CHELIUS, A. FRABOULET, E. FLEURY. *WSNet: A Modular Event-Driven Wireless Network Simulator*, IDDN 06-370013-000, 2006, http://www.worldsens.net.

[25] A. FRABOULET, G. CHELIUS, E. FLEURY. *WSim: A Hardware Platform Simulator*, IDDN 06-370012-000, 2006, http://www.worldsens.net.

References in notes

[26] C. BASTOUL, P. FEAUTRIER. *Adjusting a Program Transformation for Legality*, in "Parallel Processing Letters", vol. 15, n^{o} 1-2, March-June 2005, p. 3-17.

[27] L. A. BELADY. *A study of replacement algorithms for a virtual storage computer*, in "IBM Systems Journal", vol. 5, n^{o} 2, 1966, p. 78–101.

[28] Z. BUDIMLIĆ, K. D. COOPER, T. J. HARVEY, K. KENNEDY, T. S. OBERG, S. W. REEVES. *Fast Copy Coalescing and Live-Range Identification*, in "PLDI '02: Proceedings of the ACM SIGPLAN 2002 Conference on Programming language design and implementation, New York, NY, USA", ACM Press, 2002, p. 25–32.

[29] A. DARTE, B. R. RAU, R. SCHREIBER. *Programmatic Iteration Scheduling for Parallel Processors*, August 2002, US patent number 6438747.

[30] A. DARTE, R. SCHREIBER. *Programmatic Method For Reducing Cost Of Control In Parallel Processes*, April 2002, US patent number 6374403.

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

[32] E. DE GREEF, F. CATTHOOR, H. DE MAN. *Memory Size Reduction Through Storage Order Optimization for Embedded Parallel Multimedia Applications*, in "Parallel Computing", vol. 23, 1997, p. 1811-1837.

[33] E. F. DEPRETTERE, E. RIJPKEMA, P. LIEVERSE, B. KIENHUIS. *Compaan: Deriving Process Networks from Matlab for Embedded Signal Processing Architectures*, in "8th International Workshop on Hardware/Software Codesign (CODES'2000), San Diego, CA", May 2000.

[34] H. DEVOS, K. BEYLS, M. CHRISTIAENS, J. VAN CAMPENHOUT, D. STROOBANDT. *From Loop Transformation to Hardware Generation*, in "Proceedings of the 17th ProRISC Workshop, Veldhoven", 11 2006, p. 249-255.

[35] BENOÎT. DUPONT DE DINECHIN, C. MONAT, F. RASTELLO. *Parallel Execution of the Saturated Reductions*, in "Workshop on Signal Processing Systems (SIPS 2001)", IEEE Computer Society Press, 2001, p. 373-384.

[36] M. FARACH-COLTON, V. LIBERATORE. *On Local Register Allocation*, in "Journal of Algorithms", vol. 37, n^{o} 1, 2000, p. 37-65.

[37] P. FEAUTRIER. *Parametric Integer Programming*, in "RAIRO Recherche Opérationnelle", vol. 22, September 1988, p. 243–268.

[38] P. FEAUTRIER. *Some Efficient Solutions to the Affine Scheduling Problem, Part I, One Dimensional Time*, in "International Journal of Parallel Programming", vol. 21, n^{o} 5, October 1992, p. 313-348.

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

[40] 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.

[41] C. GUILLON, F. RASTELLO, T. BIDAULT, F. BOUCHEZ. *Procedure Placement using Temporal-Ordering Information: Dealing with Code Size Expansion*, in "Journal of Embedded Computing", vol. 1, n^{o} 4, 2005, p. 437-459.

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

[43] G. KAHN. *The Semantics of a Simple Language for Parallel Programming*, in "IFIP'74", N. HOLLAND (editor). , 1974, p. 471-475.

[44] V. LEFEBVRE, P. FEAUTRIER. *Automatic Storage Management for Parallel Programs*, in "Parallel Computing", vol. 24, 1998, p. 649-671.

[45] F. QUILLERÉ, S. RAJOPADHYE, D. WILDE. *Generation of Efficient Nested Loops from Polyhedra*, in "International Journal of Parallel Programming", vol. 28, n^{o} 5, 2000, p. 469–498.

[46] F. QUILLERÉ, S. RAJOPADHYE. *Optimizing Memory Usage in the Polyhedral Model*, in "ACM Transactions on Programming Languages and Systems", vol. 22, n^{o} 5, 2000, p. 773-815.

[47] F. RASTELLO, F. DE FERRIÈRE, C. GUILLON. *Optimizing Translation Out of SSA using Renaming Constraints*, in "International Symposium on Code Generation and Optimization (CGO'04)", IEEE Computer Society Press, March 2004, p. 265-278.

[48] V. C. SREEDHAR, R. D.-C. JU, D. M. GILLIES, V. SANTHANAM. *Translating Out of Static Single Assignment Form*, in "Static Analysis Symposium", A. CORTESI, G. FILÉ (editors). , Lecture Notes in Computer Science, vol. 1694, Springer, 1999, p. 194–210.

[49] V. SREEDHAR, R. JU, D. GILLIES, V. SANTHANAM. *Translating Out of Static Single Assignment Form*, in "Static Analysis Symposium, Italy", 1999, p. 194 – 204.

[50] 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.

[51] D. WILDE. *A Library for Doing Polyhedral Operations*, Technical report, n^{o} 785, Irisa, Rennes, France, 1993, http://hal.inria.fr/inria-00074515.