INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Team Alchemy

Architectures, Languages and Compilers to Harness the End of Moore Years

Futurs

THEME COM

Table of contents

1. Team

Head of project team

Olivier Temam [Research Director (DR) Inria, HdR]

Administrative assistants

Stéphanie Meunier [TR Inria, with Gemo]

Staff members, Inria

Hugues Berry [Research Associate (CR) Inria, on secondment from the Cergy-Pontoise University until september, 2006, then CR1 since october 2006]
Albert Cohen [Research Associate (CR) Inria]
Christine Eisenbeis [Research Director (DR) Inria]
Grigori Fursin [Postdoctoral Fellow]

Staff members, Paris-11 University

Cédric Bastoul [Assistant Professor]
Frédéric Gruau [Assistant Professor]

Technical staff

Hamid Daoud [Expert engineer, FP6 IST grant]
Sylvain Girbal [Expert engineer, FP6 IST Grant]
Sebastian Pop [Expert engineer, since October, 2006, Thalès (Carroll) grant]

Ph. D. students

Mounira Bachir [Inria scholarship, since October, 2006, University of Versailles-Saint-Quentin]
Patrick Carribault [Bull fellowship (Cifre), University of Versailles-Saint-Quentin]
Mohamed Fellahi [Inria scholarship, since October, 2006, University of Paris-Sud]
Fei Jiang [Inria scholarship, with the TAO Inria team, University of Paris-Sud]
Piotr Lesnicki [MENRT scholarship, since October, 2006 University of Paris-Sud]
Zheng Li [Inria scholarship, University of Paris-Sud]
Pierre Palatin [CNRS BDI scholarship, University of Paris-Sud]
Sebastian Pop [École Nationale Supérieure des Mines de Paris, until September, 2006]
Louis-Noël Pouchet [MENRT scholarship, since October, 2006, University of Paris-Sud]
Benoît Siri [Inria scholarship, University of Paris-Sud]
Nicolas Vasilache [MENRT scholarship, University of Paris-Sud]

Student interns

Hamid Daoud [Master of computer science, Paris-13 University, March to September, 2006]
Frédéric De Mesmay [École Polytechnique, March to July, 2006]
Mohamed Fellahi [Master of computer science, University of Paris-Sud, March to September, 2006]
Helena Fulger [École Polytechnique, March to July, 2006]
Fei Jiang [Master of computer science, University of Paris-Sud, March to September, 2006 (with the Tao Inria team)]
Charles-Eric Laporte [Master Paris Centre, March to July, 2006]
Piotr Lesnicki [Master of computer science, University of Paris-Sud, March to September, 2006]
Louis-Noël Pouchet [Master of computer science, University of Paris-Sud, March to September, 2006]
Alina Stoica [École Polytechnique, March to July, 2006]

External collaborators

Pierre Amiranoff [PRAG, IUT d'Orsay]
Denis Barthou [Assistant professor, University of Versailles-Saint-Quentin]
Benjamin Dauvergne [PhD student, Tropics project-team, Inria Sophia-Antipolis]
Sébastien Donadio [PhD student, University of Versailles-Saint-Quentin]
Nathalie Drach [Professor, Paris-6 University]
Sid-Ahmed-Ali Touati [Assistant professor, University of Versailles-Saint-Quentin]

2. Overall Objectives

2.1. Overall Objectives

ALCHEMY is a joint Inria/University of Paris Sud research group.

The general research topics of the ALCHEMY group are architectures, languages and compilers for high-performance embedded and general-purpose processors. ALCHEMY investigates *scalable* architecture and compiler/programming solutions for high-performance general-purpose and embedded processors. ALCHEMY stands for Architectures, Languages and Compilers to Harness the End of Moore Years, referring to both the traditional processor architectures implemented using the current photolithographic processes, and novel architecture/language paradigms compatible with future and alternative technologies. The current emphasis of ALCHEMY is on the former part, and we are progressively increasing our efforts on the latter part.

The research goals of ALCHEMY span from short term to long term. The short-term goals target existing complex processor architectures, and thus focus on improving program performance on these architectures (software-only techniques). The medium-term goals target the upcoming CMPs (Chip Multi-Processors) with a large number of cores, which will result from the now slower progression of core clock frequency due to technological limitations. The main challenge is to take advantage of the large number of cores for a wide range of applications, considering that automatic parallelization techniques have not yet proved an adequate solution. In ALCHEMY, we explore joint architecture/programming paradigms as a pragmatic alternative solution. Finally, even longer term research is conducted with the goal of harnessing the properties of future and alternative technologies for processing purposes.

Most of the research in ALCHEMY attempts to jointly consider the hardware and software aspects, based on the premise that many of the limitations of existing architecture and compiler techniques stem from the lack of cooperation between architects and compiler designers. However, ALCHEMY addresses the aforementioned research goals through two different, though sometimes complementary, approaches. One approach considers that, in spite of their complexity, architectures and programs can still be accurately and efficiently modeled (and optimized) using *analytical* methods. The second approach considers the architecture/program pair already has or will reach a complexity level that will evade analytical methods, and explores a *complex systems* approach; the principle is to accept that the architecture/program pair is more easily understood (and thus optimized) based on its observed behavior rather than infered from its known design.

3. Scientific Foundations

3.1. Scientific Foundations

In the sections below, the different research activities of Alchemy are described, from short-term to long-term goals. For most of the goals, both analytical and complex systems approaches are conducted.

3.1.1. A practical approach to program optimizations for complex architectures

This part of our research work is more targeted at single-core architectures but also applies to multi-cores. The rationale for this research activity is that compilers rely on architecture models embedded in heuristics to drive compiler optimizations and strategy. As architecture complexity increases, such models tend to be too simplistic, often resulting in inefficient steering of compiler optimizations.

3.1.1.1. Iterative optimization

Our general approach consists in acknowledging that architectures are too complex to embed reliable architecture models in compilers, and to explore the behavior of the architecture/program pair through repeated executions. Then, using machine-learning techniques, a model of this behavior is infered from the observations. This approach is usually called *iterative optimization*.

In the recent years, iterative optimization has emerged as a major research trend, both in traditional compilation contexts and in application-specific library generators (like ATLAS or SPIRAL). The topic has matured significantly since the pionneering works of Mike O'Boyle [78] at University of Edinburgh, UK or Keith Cooper [53] at Rice University. While these research works successfully demonstrated the performance *potential* of the approach, they also highlighted that iterative optimization cannot become a *practical* technique unless a number of issues are resolved. Some of the key issues are: the size and structure of the search space, the sensitivity to data sets, and the necessity to build long transformation sequences.

Scanning a large search space. Transformation parameters, the order in which transformations are applied, and even which transformations must be applied and how many times, all form a huge transformation space. One of the main challenges of iterative optimization is to rapidly converge towards an efficient, if not optimal, point of the transformation space. Machine-Learning techniques can help build an empirical model of the transformation space in a simple and systematic way, only based on the observation of transformations behavior, and then rapidly deduce the most profitable points of the space. We are investigating how to correlate static and dynamic program features with transformation efficiency. This approach can speed up the convergence of the search process by one or two orders of magnitude compared to random search [25], [38], [1].

We have also shown that by representing the impact of loop transformations using structured encoding derived from polyhedral program representation, it is possible to reduce the complexity of the search by several orders of magnitude [39]. This encoding is further described in Section 3.1.1.2.

We have also shown that it is possible to further speed up transformation space exploration by exploring several transformations during a single run [60]. Currently, one program transformation is explored for each loop nest, while performance often reaches a stable state soon after the start of the execution. We have shown that, assuming we properly identify the phase behavior of programs, it is possible to explore multiple transformations in each run.

Data set sensitivity. Iterative optimization is based on the notion that the compiler will discover the best way to optimize a program through repeatedly running the same program on the same data set, trying one or a few different optimizations upon each run. However, in reality, a user rarely needs to execute the same data set twice. Therefore, iterative optimization is based on the implicit assumption that the best optimization configuration found will work *well* for *all data sets* of a program. To the best of our knowledge, this assumption has never been thoroughly investigated. Most studies on iterative optimization repeatedly execute the same program/data set pair [52], [61], [58], [70], [42], only recently, some studies have focused on the impact of data sets on iterative optimizations [68], [45].

In order to explore the issue of data set sensitivity, we have assembled a data set suite, of 20 data sets per benchmark, for most of the MiBench [65] embedded benchmarks. We have found that, though a majority of programs exhibit stable performance across data sets, the variability can significantly increase with many optimizations. However, for the best optimization configurations, we find that this variability is in fact small. Furthermore, we show that it is possible to find a compromise configuration across data sets which is often within 5% of the best possible optimization configuration for most data sets, and that the iterative process can converge in less than 20 iterations (for a population of 200 optimization configurations). Overall, the preliminary conclusion, at least for the MiBench benchmarks, is that iterative optimization is a fairly robust technique across data sets, which brings it one step closer to practical usage.

Compositions of program transformations. Compilers impose a certain set of program transformations, an order of application and how many times each transformation is applied. In order to explore what are the possible gains beyond these strict constraints, we have manually optimized kernels and benchmarks, trying to achieve the best possible performance assuming no constraint on transformation order, count or selection [80], [79]. The study helped us clarify which transformations bring the best performance improvements in general. But the main conclusion of that study is that surprisingly long compositions of transformations are sometimes needed (in one case, up to 26 composed loop transformations) in order to achieve good performance. Either because multiple issues must be tackled simultaneously or because some transformations act as enabling operations for other transformations.

As a result, we have started developing a framework facilitating the composition of long transformations. This framework is based on the polyhedral representation of program transformations [51]. This framework also enables a more analytical approach to program optimization and parallelization, beyond the simple composition of transformations. This latter part is further developed in Section 3.1.1.2.

Putting it all together: continuous optimization. Increasingly, we are now moving toward automatizing the whole iterative optimization process. Our goal is to bring together, within a single software environment, the different aforementioned observations and techniques (search space techniques, data set sensitivity properties, long compositions of transformations,...). We are currently in the process of plugging these different techniques within GCC in order to create a tool capable of doing continuous, whole-program optimization, and even collaborative optimization across different users.

Hardware-Oriented applications of iterative optimization. Because iterative optimization can successfully capture complex dynamic/run-time phenomena, we have shown that the approach can act as a replacement for costly hardware structures designed to improve the run-time behavior of programs, such as out-of-order execution in superscalar processors. An iterative optimization-like strategy applied to an embedded VLIW processor [54] was shown to achieve almost the same performance as if the processor was fitted with dynamic instruction reordering support. We are also investigating applications of this approach to the specialization/idiomization of general-purpose and embedded processors [88]. Currently, we are exploring similar approaches for providing thread scheduling and placement information for CMPs without requiring costly run-time environement overhead or hardware support. This latter study is related to the work presented in Section 3.1.2.

CURRENT ACTIVITIES: IST STREP MilePost, FET IP SARC, MEDEA+ ITEA GGCC, IST NoE HiPEAC, RNTL COP.

CURRENT PEOPLE: Albert Cohen, Patrick Carribault, Grigori Fursin, Sylvain Girbal, Piotr Lesnicki, Louis-Noël Pouchet, Olivier Temam, Nicolas Vasilache.

3.1.1.2. Polyhedral program representation: facilitating the analysis and transformation of programs

As loop transformations are utterly important — performancewise — and among the hardest to predictably drive through static cost models, their current support in compilers is disappointing. After decades of experience and theoretical advances, the best compilers can miss some of the most important loop transformations in simple numerical codes from linear algebra or signal processing codes. Performance hits of more than an order of magnitude are not uncommon on single-threaded code, and the situation worsens when automatically parallelizing or optimizing parallel code.

Our previous work on sequences of loop transformations [51] has led to the design of a theoretical framework, based on the polyhedral model [55], [56], [57], [83], [75], [87], and a set of tools based on the advanced Open64 compiler. We have shown that this framework does simplify the problem of building complex transformation sequences, but also that it scales to real-world benchmarks [36], [16], [50], [9], and allows to significantly reduce the size of the search space and better understand its structure [39]. The latter work, for example, is the first attempt at directly characterizing all *legal and distinct* ways to reschedule a loop nest.

After two decades of academic research, the polyhedral model is finally evolving into a mature, production-ready approach to solve the challenges of maximizing the scalability and efficiency of statically-controlled, loop-based computations on a variety of high performance and embedded targets. After Open64, we are now porting these techniques to the GCC compiler [31], applying them to several multi-level parallelization and optimization problems, including vectorization, extraction and exploitation of thread-level parallelism on distributed memory CMPs like the Cell broadband engine from IBM, NXP's CAT-DI scalable signal-processing accelerator and novel STMicroelectronics emerging xStream architecture.

CURRENT ACTIVITIES: IST STREP ACOTES, ANR CIGC PARA, RNTL COP.

CURRENT PEOPLE: Cedric Bastoul, Albert Cohen, Sebastian Pop, Louis-Noël Pouchet, Nicolas Vasilache.

3.1.2. Joint architecture/programming approaches

While Section 3.1.1 is only concerned with transforming programs for a more efficient exploitation of existing architectures, in the longer term, researchers can assume modifications of architectures and/or programs are possible. These relaxed constraints allow to target the root causes of poor architecture/program performance.

The current architecture/program model partly fails because the burden is either excessively on the architecture (superscalar processors), or the compiler (VLIW and now CMPs). And both compiler and architecture optimizations often aim at program *reverse-engineering*: compilers attempt to dig up program properties (locality, parallelism) from the static program, while architectures attempt to retrieve them from program run-time behavior. Now, in many cases, the user is not only aware of these properties but may pass them effortlessly to the architecture and the compiler provided she had the appropriate programming support, provided the compiler would pass this information to the architecture, and the architecture would be fitted with the appropriate support to take advantage of them. For instance, simply knowing that a C structure denotes a tree rather than a graph can provide significant information for parallel execution. Such approaches, while not fully automatic, are practical and would relieve the complexity burden of the architecture and the compiler, while extracting significant amounts of task-level parallelism.

In the paragraphs below we apply this approach of passing more program semantic to the compiler and the architecture, first for domain-specific stream-oriented programs, and then for the parallelization of more general programs.

3.1.2.1. Passing program semantics using a synchronous language for high-performance video processing

While we are currently investigating the aforementioned approach for general-purpose applications, we have started with the investigation of the specific domain of high-end video processing. In this domain, assessing that real-time properties will be satisfied is as important as reaching uncommon levels of compute density on a chip. 150 giga-operations per second per Watt (on pixel components) is the norm for current high-definition TVs, and cannot be achieved with programmable cores at present. The future standards will need an 8-fold increase (e.g., for 3D displays or super-high-definition). Predictability and efficiency are the keywords in this domain, in terms of both architecture and compiler behavior.

Our approach combines the aforementioned iterative optimization and polyhedral modeling research with a predictability- and efficiency-oriented parallel programming language. We focus on warrantable (as opposed to best-effort) usage of hardware resources with respect to real-time constraints. Therefore, this parallel programming language must allow overhead-free generation of tightly coupled parallel threads, interacting through dedicated registers rather than caches, streaming data through high-bandwidth, statically managed interconnect structures, with frequent synchronizations (once every few cycles), and very limited memory resources immediately available. This language also needs to support advanced loop transformations, and its representation of concurrency compatible with the expression of multi-level partitioning and mapping decisions. All these conditions tend to consider a language closer to hardware synthesis languages than general-purpose, von Neumann oriented imperative ones [48], [49].

The synchronous data-flow paradigm is a natural candidate, because of its ability to combine high-productivity in programming complex concurrent applications (due to the determinism and compositionality of the underlying model, a rare feature of a concurrent semantics), direct modeling of computation/communication time, and static checking of non-functional properties (time and resource constraints). Yet generating low-level, tightly fused loops with maximal exposition of fine-grain parallelism from such languages is a difficult problem, as soon as the target processor is not the one being described by the synchronous data-flow program, but a pre-existing target on which we are folding an application program. The two tasks are totally different: although the most difficult decisions are pushed back to the programmer in the hardware synthesis case, application programmers usually rely on the compiler to abstract away the folding of their code in a reasonably portable fashion across a variety of targets. This aspect of synchronous language compilation has largely been overlooked and constitutes the main direction of our work. Another direction lies in the description of hardware resources, at the same level as the application being mapped and scheduled onto them; this unified

representation would allow the expression of the search space of program transformations, and would be a necessary step to apply incremental refinement methods (expert-driven, very popular in this domain).

Technically, we extend the classical clock calculus (a type system) of the *Lucid Synchrone* language, expliciting significantly more information about the program behavior, especially when tasks must be started and will be completed, how information flow among tasks, etc. Our main contribution is the integration of relaxed synchronous operators like jittering and bursty streams within synchronous bounds [26]. This research consists in revisiting the semantics of synchronous Kahn networks in the domain of media streaming applications and reconfigurable parallel architectures, in collaboration with Marc Duranton from Philips Research Eindhoven (now NXP Semiconductors) and with Marc Pouzet from LRI and the Proval Inria project team.

CURRENT ACTIVITIES: IST STREP ACOTES, Marie Curie ToK-IAP PSYCHES.

CURRENT PEOPLE: Albert Cohen, Christine Eisenbeis, Mohammed Fellahi.

3.1.2.2. Passing program semantic using software components

Beyond domain-specific and regular applications (loops and arrays), automatic compiler-based parallelization has achieved only mixed results on programs with complex control and data structures [66]. Writing, and especially debugging, large parallel programs is a notoriously difficult task [69], and one may wonder whether the vast majority of programmers will be able to cope with it. Currently, transactional memory is a popular approach [67] for reducing the programmer burden using intuitive transaction declarations instead of more complex concurrency control constructs. However, it does not depart from the classic approach of parallelizing standard C/C++/Fortran programs, where parallelism can be difficult to extract or manipulate. Parallel languages, such as HPF [73], require more ambitious evolutions of programming habits, but they also let programmers pass more semantic about the control and data characteristics of programs to the compiler for easier and more efficient parallelization. However, one can only observe that, for the moment, few such languages have become popular in practice.

A solution would have a better chance to be adopted by the community of programmers at large if it integrates well with popular practices in *software engineering*, and this aspect of the parallelization problem may have been overlooked. Interestingly, software engineering has recently evolved towards a programming model that can blend well with multi-core architectures and parallelization. Programming has consistently evolved towards more encapsulation: procedures, then objects, then *components* [86]. Essentially for two reasons, because programmers have difficulties grasping large programs and need to think locally, and because encapsulation enables *reuse* of programming efforts. Component-based programming, as proposed in Java Beans, .Net or more ad-hoc component frameworks, is the step beyond C++ or Java objects: programs are decomposed into modules which fully encapsulate code and data (no global variable) and which communicate among themselves through explicit interfaces/links.

Components have many assets for the task of developing parallel programs. (1) Components provide a pragmatic approach for bringing parallelization to the community at large thanks to component reuse. (2) Components provide an implicit and intuitive programming model: the programmer views the program as a "virtual space" (rather than a sequence of tasks) where components reside; two components residing together in the space and not linked or not communicating through an existing link implicitly operate in parallel; this virtual space can be mapped to the physical space of a multi-threaded/multi-core architecture. (3) Provided the architecture is somehow aware of the program decomposition into components, and can manipulate individual components, the compiler (and the user) would be also relieved of the issue of mapping programs to architectures.

In order to use software components for large-scale and fine-grain parallelization, the key notion is to augment them with the ability to split or replicate. For instance, a component walking a binary tree could spawn two components to scan two child nodes and the corresponding subtrees in parallel.

We are investigating a low-overhead component-based approach for fine-grain parallelism where components have the ability to replicate [71], [30]. We investigate both a hardware-supported and software-only approach to component division. We show that a low-overhead component framework, possibly paired with component

hardware support, can provide both an intuitive programming model for writing fine-grain parallel programs with complex control flow and data structures, and an efficient platform for parallel components execution.

CURRENT ACTIVITIES: FET IP SARC, IST NoE HiPEAC, ANR APE.

CURRENT PEOPLE: Olivier Certner, Zheng Li, Pierre Palatin, Olivier Temam.

3.1.3. Spatial computing

The last research direction stems from possible evolutions of technology. While this research direction may seem very long term, processor manufacturers cannot always afford to investigate many risky alternatives way ahead in time. At the same time, for them to accept and adopt radical changes, they have to be anticipated long in advance. Thus, we believe prospective research is a core role for academic researchers, which may be less immediately useful to companies, but which can bring a real addition to their internal research activities, and which also carries the potential of bringing disruptive technology.

Prospective information on the future of CMOS technology suggests that, though the density of transistors will keep increasing, the commuting speed of transistors will not increase as fast, and transistors may be more faulty (either fabrication defects or execution faults). Possible replacement/alternative technologies, such as nanotubes [63] which have received a lot of attention lately, share many of these properties: high density, but slow components (possibly even slower than current components), a large rate of defects/faults, and more difficulty to place them except than in fairly regular structures.

In short, several potential upcoming technologies seem to bring a very large number of possibly faulty and not so fast components with layout issues. For architectures to take advantage of such technology, they would have to rely on *space* much more than *time/speed* to achieve high performance. Large spatial architectures bring a set of new architecture issues, such as controling the execution of a program in a totally decentralized way, efficiently managing the placement of program tasks on the space, and managing the relative movement of these different tasks so as to minimize communications. Furthermore, beyond a certain number of processing elements, it is not even clear whether many applications will embed enough traditional task-level parallelism to take advantage of such large spaces, so applications may have to be expressed (programmed) differently in order to leverage that space. These two research issues are addressed in the two research activities described below.

Blob computing. Blob computing [5] is both a spatial programming and architecture model which aims at investigating the utilization of a vast amount of processing elements. The key originality of the model is to acknowledge that the chip space becomes too large for anything else than purely *local* actions. As a result, all architecture control becomes local. Similarly, the program itself is decomposed into a set of purely local actions/tasks, called Blobs, connected together through links; the program can create/destroy these links during its lifetime.

With respect to architecture control, for instance, the local method for expressing that two tasks frequently communicating through a link must get close together in space so that their communication latency is low is expressed through a simply physical law, emulating spring tension; the more communications, the higer the tension. Similarly, expressing that tasks should move away because too many tasks are grouped in the same physical spot is achieved through a law similar to pressure: as the number of tasks increases, the local pressure on neighbor tasks increases, inducing them to move away. Overall many of these local control rules derive from physical or biological laws which achieve the same goals: controling a large space through simple local interactions.

With respect to programming, the user essentially has to decompose the program into a set of nodes and links. The program can create a static node/link topology that is later used for computations, or it can dynamically change that topology during execution. But the key concept is that the user is not in charge of placing tasks on the physical space, only to express the *potential* parallelism through task division. As can be observed, several of the intuitions of the CAPSULE environment of Section 3.1.2.2 stems from this Blob model.

Bio-Inspired computing. As mentioned above, beyond a certain number of individual components, it is not even clear whether it will be possible to decompose tasks in such a way they can take advantage of a large space. Searching for pieces of solution to this problem has progressively lead us to biological neural networks. Indeed, biological neural networks (as opposed to artificial neural networks, ANNs) are well-known examples of systems capable of complex information processing tasks using a large number of self-organized, but slow and unreliable components. And the complexity of the tasks typically processed by biological neurons is well beyond what is classically implemented with ANNs, because ANNs lack key features of biological neural networks.
Emulating the workings of biological neural networks may at first seem far-fetched. However, the SIA (Semiconductor Industry Association) in its 2005 roadmap addresses for the first time "biologically inspired architecture implementations" [85] as emerging research architectures and focuses on biological neural networks as interesting scalable designs for information processing. More importantly, the computer science community is beginning to realize that biologists have made tremendous progress in the understanding of how certain complex information processing tasks are implemented (programmed) using biological neural networks.
One of the key emerging features of biological neural networks is that they process information by *abstracting* it, and then only manipulate such higher abstractions. As a result, each new input (for image processing for instance) can be analyzed using these learned abstractions directly, thus avoiding to rerun a lengthy set of elementary computations. More precisely, Poggio et al. [81] at MIT have shown how combinations of neurons implementing simple operations such as MAX or SUM, can automatically create such abstractions for image processing, and some computer science researchers in the image processing domain have started to take advantage of these findings.
We are starting to investigate the information processing capabilities of this abstraction programming method [34], [13], [33], [12]. While image processing is also our first application, we plan to later look at a more diverse set of example applications.

A complex systems view of computing systems. More generally, the increased complexity of computing systems at stake, whether due to a large number of individual components, a large number of cores or simply complex architecture program/pairs, suggest that novel design and evaluation methodologies should be investigated that rely less on known design information than on observed behavior of the global resulting system. The main problem here is to be able to extract general characteristics of the architecture on the basis of measurements of its global behavior. For that purpose, we are using tools provided by the physics of complex systems (nonlinear time series analysis, phase transitions, multifractal analysis...).
We have already applied such tools to better understand the performance behavior of complex but traditional computing systems such as superscalar processors [24], [3]. And we are starting to apply them to sampling techniques for performance evaluation [64], [29]. We will be progressively expanding the reach of these techniques in our research studies in the future.

CURRENT ACTIVITIES: ASTICO ACI grant.

CURRENT PEOPLE: Hugues Berry, Christine Eisenbeis, Frédéric Gruau, F. Jiang, Benoît Siri, Olivier Temam

3.1.4. Transversal research activities: simulation and compilation

Since our research group has been involved in both compiler and architecture research for several years, we have progressively given increased attention to tools, partly because we found a lot of productivity was lost in inefficient or hard to reuse tools. Since then, both simulation and compilation platforms have morphed into research activities of their own. Our group is now coordinating the development of the simulation platform of the European HiPEAC network, and it is co-coordinating the development of the compiler research platform of HiPEAC together with University of Edinburgh.

3.1.4.1. Simulation platform

As processor architecture and program complexity increase, so does the development and execution time of simulators. Therefore, we have investigated simulation methodologies capable of increasing our research productivity. The key point is to improve the reuse, sharing, comparison and speed capabilities of simulators. For the first three properties, we are investigating the development of a *modular* simulation platform, and for the latter fourth property, we are investigating sampling techniques and more abstract modeling techniques. Our simulation platform is called UNISIM [41].

What is UNISIM? UNISIM is a structural simulation environment which provides an intuitive mapping from the hardware block diagram to the simulator; each hardware block corresponds to a simulation module. UNISIM is also a library of modules where researchers will be able to download and upload (contribute) modules.

What are the assets of UNISIM over other simulation platforms? UNISIM allows to reuse, exchange and compare simulator parts (and architecture ideas), something that is badly needed in academic research, and between academia and industry. Recently, we did a comparison of 10 different cache mechanisms proposed over the course of 15 years [8], and suggested the progress of research has been all but regular because of the lack of a common ground for comparison, and because simulation results are easily skewed by small differences in the simulator setup.

Also, other simulation environments or simulators advocate modular simulation for sharing and comparison, such as the SystemC environment [40], or the M5 simulator [46]. While they do improve the modularity of simulators, in practice, reuse is still quite difficult because most simulation environments overlook the difficulty and importance of reusing *control*. For instance, SystemC focuses on reusing hardware blocks such as ALUs, caches, and so on. However, while hardware blocks correspond to the greatest share of transistors in the actual design, they often correspond to the least share of simulator lines. For instance, the cache data and instruction banks often correspond to a sizeable amount of transistors, but they merely correspond to array declarations in the simulator; conversely, cache control corresponds to few transistors but most of the source lines of any cache simulator function/module. As a result, it is difficult to achieve reuse in practice, because control code is often not implemented in such a way that it can lend well to reuse.

On the contrary, UNISIM is focused on reuse of control code, provides a standardized module communication protocol and a control abstraction for that purpose. Moreover, UNISIM will later on come with an open library in order to better structure the set of available simulators and simulator components.

Taking a realistic approach at simulator usage. Obviously, many research groups will not accept easily to drop years of investment in their simulation platforms and to switch to a new environment. We take a pragmatic approach and UNISIM is designed from the ground up to be interoperable with existing simulators, from industry and academia. We achieve interoperability by wrapping full simulators or simulator parts within UNISIM modules. We have an example full SimpleScalar simulator stripped of its memory, wrapped into a UNISIM module, and plugged into a UNISIM SDRAM module.

Moreover, we are in the process of developing a number of APIs (for power, GUI, functional simulators, sampling,...) which will allow third-party tools to be plugged into the UNISIM engine. We call these APIs simulator capabilities or services.

With CMPs, communications become more important than cores cycle-level behavior. While the current version of UNISIM is focused on cycle-level simulators, we are developing a more abstract view of simulators called Transaction-Level Models (TLM). Later on, we will also allow hybrid simulators, using TLM for prototyping, and then zooming on some components of a complex system.

Because CMPs also require operating system support for a large part, and because existing alternatives such as SIMICS [74] are not open enough, we are also developing full-system support in our new simulators jointly with CEA. Currently, UNISIM has a functional simulator of a PowerPC750 capable of booting Linux.

Cooperation. While Inria was initially developing its own environment called MicroLib [77],[8],[1] we found many similarities with the Liberty environment developed at Princeton University (David August). After a few

interactions, we decided to merge the two environments into a single one called UNISIM. Since then, UNISIM has been adopted as the official simulation platform of the European HiPEAC network, and has attracted other cooperations. First, from CEA, a French research institution, which is heavily involved in the development of the TLM part of UNISIM, and also with UPC, which is involved in both the TLM part and the development of CMP simulators (they have a UNISIM model of an IBM Cell). Ghent University has recently started to investigate the application of its statistical simulation techniques to UNISIM simulators.

CURRENT ACTIVITIES: European NoE HiPEAC, European IP SARC, Inria MODSIM joint team.

CURRENT PEOPLE: Sylvain Girbal, Olivier Temam, Zheng Li.

3.1.4.2. Compilation platform

The free *GNU Compiler Collection* (GCC) is the leading tool suite for portable developments on open platforms. It supports more than 6 input languages and 30 target processor architectures and instruction sets, with state-of-the-art support for debugging, profiling and cross-compilation. It has long been supported by the general-purpose and high-performance hardware vendors. The last couple of years have seen GCC taking momentum in the embedded system industry, and also as a platform for advanced research in program analysis, transformation and optimization.

GCC 4.2 features more than 170 compilation passes, two thirds of them playing a direct role in program optimization. These passes are selected, scheduled, and parameterized through a versatile pass manager. The main families of passes can be classified as:

- interprocedural analyses and optimizations;
- profile-directed optimization (interprocedural and intraprocedural);
- induction variable analysis, canonicalization and strength-reduction;
- loop optimizations;
- automatic vectorization;
- data layout optimization.

More advanced developments are in progress. We identified three major ones with a direct impact on high-performance embedded systems research:

- link-time optimization (towards just-in-time and dynamic compilation), with emphasis on scalability to whole-program optimization and compatibility with production usage;
- automatic parallelization, featuring full OpenMP 2.5 support and evolving towards automatic extraction of loop and functional parallelism, with ongoing research on speculative forms of parallelism.

The HiPEAC network supports GCC as a platform for research and development in compilation for high-performance and embedded systems. The network activities on the GCC research platform are coordinated by Mike O'Boyle and Albert Cohen. We briefly survey the activities conducted in this context in the Alchemy project team.

Collaborative research and mutual-interest development. Multiple research collaborations have emerged. Those including Alchemy are listed below.

- IBM Haifa and Philips Research (now NXP Semiconductors) are jointly working on automatic vectorization for complex and wide embedded vector architectures. This work makes heavy use of Sebastian Pop's results on induction variables and dependence analysis. A paper has been submitted to a journal;
- STMicroelectronics, Inria Futurs and the University of Edinburgh regularly exchange personnel and ideas on just-in-time and machine learning compilation; one PhD student (Piotr Lesnicki) started in October on these topics, with the option to hire another one early in 2007.

[1] Some of the MicroLib developments were rather heavily disseminated: 3500+ downloads of our PowerPC750 simulator, 7400+ downloads of our parallel simulation environment DIST [62] as of November 2006.

These efforts have an impact on the visibility and influence of European researchers in the GCC developer community (heavily industry backed, with major contributions from IBM, HP, Intel, AMD, RedHat and Apple). 7 participants to the 2006 GCC developer's summit were affiliated to HiPEAC institutions (IBM Haifa, STMicroelectronics, Philips and Inria Futurs).

CURRENT ACTIVITIES: IST NoE HiPEAC, FET IP SARC, IST STREP ACOTES, IST STREP MilePost, MEDEA+ ITEA GGCC.

CURRENT PEOPLE: Albert Cohen, Grigori Fursin, Sebastian Pop, Olivier Temam, Piotr Lesnicki, Hamid Daoud.

4. Software

4.1. Main software developments

4.1.1. Main software developments

COMPILERS & PROGRAM OPTIMIZATION:

Polyhedral transformations in Open64 The WRaP-IT tool (WHIRL Represented as Polyhedra – Interface Tool) is a program analysis and transformation tool implemented on top of the Open64 compiler [44] and of the CLooG code generator [43]. The formal basis of this tool is the polyhedral model for reasoning about loop nests. We introduced a specific polyhedral representation that guarantees strong transformation compositionality properties [51]. This new representation is used to generalize classical loop transformations, to lift the constraints of classical compiler frameworks and enable more advanced iterative optimization and machine learning schemes. WRaP-IT — and its loop nest transformation kernel called URUK (Unified Representation Universal Kernel) — is designed to support a wide range of transformations on industrial codes, starting from the SPEC CPU2000 benchmarks, and recently considering a variety of media and signal processing codes (vision, radar, software radio, video encoding, and DNA-mining in particular, as part of the IST STREP ACOTES, ANR CIGC PARA, and a collaboration with Thales).

Based on this framework, we are also planning an extension of the polyhedral model to handle speculative code generation and transformation of programs with data-dependent control, and a direct search and transformation algorithm based on the Farkas lemma. These developments will take place in the GRAPHITE project: a migration/rewrite of our Open64-based software to the GCC suite. This project is motivated by the maturity — performancewise and infrastructurewise — of GCC 4.x, and on the massive industrial investment taking off on GCC in the recent years, especially in the embedded world. We are heavily involved in fostering research projects around GCC as a common compilation platform, and GRAPHITE is one of those projects.

Grigori Fursin developed the first prototype of an iterative optimization API for GCC, and started using this infrastructure for continuous and adaptive optimization research, in collaboration with the University of Edinburgh.

PROCESSOR SIMULATION:

UNISIM The UNISIM platform has been described in Section 3.1.4.1. As of now, besides the simulation engine, the developments include a shared-memory CMP based on the PowerPC 405, functional simulators for the PowerPC 405 (and cycle-level), PowerPC 750, a functional system simulator of the PowerPC 750 capable of booting Linux, 10 different cache modules corresponding to various research works. The following simulators or tools are currently under development: a functional and cycle-level version of the ARM 9 with full-system capability, a distributed-memory CMP based on the Power 405 core, an ST231 VLIW functional and later on cycle-level simulator.

BeeRS & IDDCA BeeRS [17] is a sampling technique that focuses on practicality by jointly considering warm-up and sampling. Most sampling techniques treat the problem separately which complicates their practical usage. BeeRS also includes the IDDCA clustering technique which has been shown to outperform traditional k-means techniques by an order of magnitude [64].

MicroLib MicroLib [84] is our former version of a modular simulation platform. It includes a library of modular simulator components freely distributed on a web site (www.microlib.org). As of now, it contains generic modules for each of the main components of a superscalar processor, a full superscalar processor model, an embedded processor model (PowerPC 750).

FastSysC FastSysC [76] is an enhanced SystemC engine. SystemC is itself a modular simulation environment which is becoming a de facto standard supported by more than 50 companies in the embedded domain. However, the SystemC engine development is geared toward adding functionalities rather than improving performance. Because performance is critical in processor simulation, due to excessively long traces, we have developed from scratch a new SystemC engine geared toward performance.

DiST As part of our efforts on speeding up simulation execution, we have developed a tool for parallelizing simulators, called DiST [62], requiring little simulator modifications and incurring only a small loss of accuracy. The main asset of the tool is that it can take advantage of multiple computing resources.

5. New Results

5.1. Practical approach to program optimizations

5.1.1. Iterative optimization

Our general approach consists in acknowledging that architectures are too complex to embed reliable architecture models in compilers, and to explore the behavior of the architecture/program pair through repeated executions. Then, using machine-learning techniques, a model of this behavior is infered from the observations. This approach is usually called *iterative optimization*.

In the recent years, iterative optimization has emerged as a major research trend, both in traditional compilation contexts and in application-specific library generators (like ATLAS or SPIRAL). The topic has matured significantly since the pionneering works of Mike O'Boyle [78] at University of Edinburgh, UK or Keith Cooper [53] at Rice University. While these research works successfully demonstrated the performance *potential* of the approach, they also highlighted that iterative optimization cannot become a *practical* technique unless a number of issues are resolved. Some of the key issues are: the size and structure of the search space, the sensitivity to data sets, and the necessity to build long transformation sequences.

Scanning a large search space. Transformation parameters, the order in which transformations are applied, and even which transformations must be applied and how many times, all form a huge transformation space. One of the main challenges of iterative optimization is to rapidly converge towards an efficient, if not optimal, point of the transformation space. Machine-Learning techniques can help build an empirical model of the transformation space in a simple and systematic way, only based on the observation of transformations behavior, and then rapidly deduce the most profitable points of the space. We are investigating how to correlate static and dynamic program features with transformation efficiency. This approach can speed up the convergence of the search process by one or two orders of magnitude compared to random search [25], [38], [1].

We have also shown that by representing the impact of loop transformations using structured encoding derived from polyhedral program representation, it is possible to reduce the complexity of the search by several orders of magnitude [39]. This encoding is further described in Section 3.1.1.2.

We have also shown that it is possible to further speed up transformation space exploration by exploring several transformations during a single run [60]. Currently, one program transformation is explored for each loop nest, while performance often reaches a stable state soon after the start of the execution. We have shown that, assuming we properly identify the phase behavior of programs, it is possible to explore multiple transformations in each run.

Data set sensitivity. Iterative optimization is based on the notion that the compiler will discover the best way to optimize a program through repeatedly running the same program on the same data set, trying one or a few different optimizations upon each run. However, in reality, a user rarely needs to execute the same data set twice. Therefore, iterative optimization is based on the implicit assumption that the best optimization configuration found will work *well* for *all data sets* of a program. To the best of our knowledge, this assumption has never been thoroughly investigated. Most studies on iterative optimization repeatedly execute the same program/data set pair [52], [61], [58], [70], [42], only recently, some studies have focused on the impact of data sets on iterative optimizations [68], [45].

In order to explore the issue of data set sensitivity, we have assembled a data set suite, of 20 data sets per benchmark, for most of the MiBench [65] embedded benchmarks. We have found that, though a majority of programs exhibit stable performance across data sets, the variability can significantly increase with many optimizations. However, for the best optimization configurations, we find that this variability is in fact small. Furthermore, we show that it is possible to find a compromise configuration across data sets which is often within 5% of the best possible optimization configuration for most data sets, and that the iterative process can converge in less than 20 iterations (for a population of 200 optimization configurations). Overall, the preliminary conclusion, at least for the MiBench benchmarks, is that iterative optimization is a fairly robust technique across data sets, which brings it one step closer to practical usage.

Compositions of program transformations. Compilers impose a certain set of program transformations, an order of application and how many times each transformation is applied. In order to explore what are the possible gains beyond these strict constraints, we have manually optimized kernels and benchmarks, trying to achieve the best possible performance assuming no constraint on transformation order, count or selection [80], [79]. The study helped us clarify which transformations bring the best performance improvements in general. But the main conclusion of that study is that surprisingly long compositions of transformations are sometimes needed (in one case, up to 26 composed loop transformations) in order to achieve good performance. Either because multiple issues must be tackled simultaneously or because some transformations act as enabling operations for other transformations.

As a result, we have started developing a framework facilitating the composition of long transformations. This framework is based on the polyhedral representation of program transformations [51]. This framework also enables a more analytical approach to program optimization and parallelization, beyond the simple composition of transformations. This latter part is further developed in Section 3.1.1.2.

Putting it all together: continuous optimization. Increasingly, we are now moving toward automatizing the whole iterative optimization process. Our goal is to bring together, within a single software environment, the different aforementioned observations and techniques (search space techniques, data set sensitivity properties, long compositions of transformations,...). We are currently in the process of plugging these different techniques within GCC in order to create a tool capable of doing continuous, whole-program optimization, and even collaborative optimization across different users.

Hardware-Oriented applications of iterative optimization. Because iterative optimization can successfully capture complex dynamic/run-time phenomena, we have shown that the approach can act as a replacement for costly hardware structures designed to improve the run-time behavior of programs, such as out-of-order execution in superscalar processors. An iterative optimization-like strategy applied to an embedded VLIW processor [54] was shown to achieve almost the same performance as if the processor was fitted with dynamic instruction reordering support. We are also investigating applications of this approach to the specialization/idiomization of general-purpose and embedded processors [88]. Currently, we are exploring similar approaches for providing thread scheduling and placement information for CMPs without requiring

costly run-time environement overhead or hardware support. This latter study is related to the work presented in Section 3.1.2.

CURRENT ACTIVITIES: IST STREP MilePost, FET IP SARC, MEDEA+ ITEA GGCC, IST NoE HiPEAC, RNTL COP.

CURRENT PEOPLE: Albert Cohen, Patrick Carribault, Grigori Fursin, Sylvain Girbal, Piotr Lesnicki, Louis-Noël Pouchet, Olivier Temam, Nicolas Vasilache.

5.1.2. Polyhedral program representation: facilitating the analysis and transformation of programs

As loop transformations are utterly important — performancewise — and among the hardest to predictably drive through static cost models, their current support in compilers is disappointing. After decades of experience and theoretical advances, the best compilers can miss some of the most important loop transformations in simple numerical codes from linear algebra or signal processing codes. Performance hits of more than an order of magnitude are not uncommon on single-threaded code, and the situation worsens when automatically parallelizing or optimizing parallel code.

Our previous work on sequences of loop transformations [51] has led to the design of a theoretical framework, based on the polyhedral model [55], [56], [57], [83], [75], [87], and a set of tools based on the advanced Open64 compiler. We have shown that this framework does simplify the problem of building complex transformation sequences, but also that it scales to real-world benchmarks [36], [16], [50], [9], and allows to significantly reduce the size of the search space and better understand its structure [39]. The latter work, for example, is the first attempt at directly characterizing all *legal and distinct* ways to reschedule a loop nest.

After two decades of academic research, the polyhedral model is finally evolving into a mature, production-ready approach to solve the challenges of maximizing the scalability and efficiency of statically-controlled, loop-based computations on a variety of high performance and embedded targets. After Open64, we are now porting these techniques to the GCC compiler [31], applying them to several multi-level parallelization and optimization problems, including vectorization, extraction and exploitation of thread-level parallelism on distributed memory CMPs like the Cell broadband engine from IBM, NXP's CAT-DI scalable signal-processing accelerator and novel STMicroelectronics emerging xStream architecture.

CURRENT ACTIVITIES: IST STREP ACOTES, ANR CIGC PARA, RNTL COP.

CURRENT PEOPLE: Cedric Bastoul, Albert Cohen, Sebastian Pop, Louis-Noël Pouchet, Nicolas Vasilache.

5.1.3. Iterative optimization meets the polytope model

Participants: Albert Cohen, Sylvain Girbal, David Parello, Olivier Temam, Nicolas Vasilache.

Static cost models have a hard time coping with hardware components exhibiting complex run-time behaviors, calling for alternative solutions. Iterative optimization is emerging as a promising research direction, but currently, it is mostly limited to finding the parameters of program transformations or selecting whole optimization phases. One of the cornerstones of our *Center for Program Tuning* (RNTL project, *Centre d'Optimisation de Programmes*, 2003–2005) is to facilitate the expression and search of compositions of program transformations. Our framework relies on a unified polyhedral representation of loops and statements. The key is to clearly separate the impact of each program transformation on the following three components: the iteration domain, the statements schedule and the memory access functions [50]. Within this framework, composing a long sequence of program transformations induces no code explosion. As a result, searching for compositions of transformations is not hampered by the multiplicity of compositions, and ultimately, it is equivalent to testing different values of the matrices parameters in many cases. Our techniques have been implemented on top of the Open64/ORC compiler. In addition, we have designed a prototype iterative optimization infrastructure for iterative optimization, based on genetic algorithms. This infrastructure distributes simulations, dynamic profiles, compilations, transformations, while interacting with a machine-learning component or with an expert user. Validation of these concepts and application of the tools is beginning on the SPEC CPU2000 benchmarks; showing the ability of our tools and framework to scale

to larger codes is a critical phase in the center for program tuning. Recent research addresses the automatic search of program transformations in a multidimensional space, combining Lagrangian relaxation (e.g., Farkas Lemma), operation research algorithms and iterative optimization.

5.1.4. Iterative compilation and continuous optimizations

Participants: Albert Cohen, Grigori Fursin, Olivier Temam.

Currently we are working on iterative compilation and continuous optimizations. For iterative compilation we attempt to further improve existing compiler infrastructure such as gcc or PathScale to be able to apply a greater variety of program transformations systematically and automatically to considerably improve program performance and/or reduce power consumption for embedded systems. We investigate both static and run-time optimizations and adaptation towards various data inputs. For continuous optimizations we attempt to collect all information available during program optimization and its multiple executions with various datasets. We further intend to use machine-learning techniques to quickly and automatically optimize new programs or adapt towards new datasets by exploiting all previously gathered knowledge.

This year we published 3 papers on the above topics.

The paper [60], ranked first at the International Conference on High Performance Embedded Architectures and Compilers (HiPEAC 2005) shows a method to make iterative optimization practical and usable by speeding up the evaluation of a large range of optimizations. Instead of using a full run to evaluate a single program optimization, we take advantage of periods of stable performance, called phases. For that purpose, we propose a low-overhead phase detection scheme geared toward fast optimization space pruning, using code instrumentation and versioning implemented in a production compiler .We demonstrate that it is possible to search for complex optimizations at run-time without resorting to sophisticated dynamic compilation frameworks. In addition to that, our approach also enables to quickly design self-tuned applications.

The two other papers [72] and [59] (in collaboration with Edinburgh University) explore the ways to search for best transformations in large optimizations spaces. The first paper uses Pugh's Unified Transformation Framework to exploit the performance improvement potential of complex transformation compositions and presents a heuristic search algorithm capable of efficiently locating good program optimizations within such a space. The second paper empirically evaluates source-level transformations and the probabilistic feedback-driven search for "good" transformation sequences within a large optimization space.

5.1.5. Low-level optimization

Participants: Patrick Carribault, Albert Cohen.

This work is done in collaboration with William Jalby from University of Versailles-Saint-Quentin. To achieve the best performance on single processors, optimizations need to target most components of the architecture simultaneously, focusing on the memory hierarchy (including registers), branch prediction, instruction-level parallelism and vector (SIMD) parallelism. Typical examples of good candidates for aggressive optimization technologies include regular and numerical computations from scientific, signal processing or multimedia applications.

More irregular programs can also be data and compute intensive, but less architecture-aware optimizations have been proposed for such programs. Still, speculative and very complex transformations are available for such codes in the context of massively parallel computers. We investigated the applicability and extension/adaptation of some of these techniques for the optimization on uniprocessors, and our results were extremely promising in the case of two approximate string-matching codes (for computational biology). Hybrid static-dynamic optimizations for such programs are also being considered, driving the selection of optimization parameters at run-time through the fine-grain tracking of the behaviour of the application (performance counters).

Finally, we studied even more irregular codes: decision trees in control-intensive emulators, text processors or memory management functions. We showed that, surprisingly, high quality performance predictions could be achieved at compile time, helping the compiler to take the right code generation decisions. This study also led to the design of a new program transformation, called Deep Jam, generalizing the unroll-and-jam optimization to nested irregular loops with conditionals and early exits [47].

5.1.6. Advanced analysis and optimization for the end-user

Participants: Albert Cohen, Sebastian Pop.

This work is done in collaboration with Georges Silber, Pierre Jouvelot and François Irigoin from École Nationale Supérieure des Mines de Paris.

We designed an induction variable analyzer suitable for the analysis of typed, low-level, three address representations in SSA form. At the heart of our analyzer is a new algorithm recognizing scalar evolutions. We define a representation called trees of recurrences that is able to capture different levels of abstractions: from the finer level that is a subset of the SSA representation restricted to arithmetic operations on scalar variables, to the coarser levels such as the evolution envelopes that abstract sets of possible evolutions in loops. Unlike previous work, our algorithm tracks induction variables without prior classification of a few evolution patterns: different levels of abstraction can be obtained on demand [7]. The low complexity of the algorithm fits the constraints of a production compiler, and roots the mainline dependence analysis framework in the Gnu Compiler Collection (GCC), as illustrated by the evaluation of our implementation on standard benchmark programs [82].

5.2. Joint architecture/programming approaches

5.2.1. Passing program semantics using a synchronous language for high-performance video processing

While we are currently investigating the aforementioned approach for general-purpose applications, we have started with the investigation of the specific domain of high-end video processing. In this domain, assessing that real-time properties will be satisfied is as important as reaching uncommon levels of compute density on a chip. 150 giga-operations per second per Watt (on pixel components) is the norm for current high-definition TVs, and cannot be achieved with programmable cores at present. The future standards will need an 8-fold increase (e.g., for 3D displays or super-high-definition). Predictability and efficiency are the keywords in this domain, in terms of both architecture and compiler behavior.

Our approach combines the aforementioned iterative optimization and polyhedral modeling research with a predictability- and efficiency-oriented parallel programming language. We focus on warrantable (as opposed to best-effort) usage of hardware resources with respect to real-time constraints. Therefore, this parallel programming language must allow overhead-free generation of tightly coupled parallel threads, interacting through dedicated registers rather than caches, streaming data through high-bandwidth, statically managed interconnect structures, with frequent synchronizations (once every few cycles), and very limited memory resources immediately available. This language also needs to support advanced loop transformations, and its representation of concurrency compatible with the expression of multi-level partitioning and mapping decisions. All these conditions tend to consider a language closer to hardware synthesis languages than general-purpose, von Neumann oriented imperative ones [48], [49].

The synchronous data-flow paradigm is a natural candidate, because of its ability to combine high-productivity in programming complex concurrent applications (due to the determinism and compositionality of the underlying model, a rare feature of a concurrent semantics), direct modeling of computation/communication time, and static checking of non-functional properties (time and resource constraints). Yet generating low-level, tightly fused loops with maximal exposition of fine-grain parallelism from such languages is a difficult problem, as soon as the target processor is not the one being described by the synchronous data-flow program, but a pre-existing target on which we are folding an application program. The two tasks are totally different: although the most difficult decisions are pushed back to the programmer in the hardware synthesis case,

application programmers usually rely on the compiler to abstract away the folding of their code in a reasonably portable fashion across a variety of targets. This aspect of synchronous language compilation has largely been overlooked and constitutes the main direction of our work. Another direction lies in the description of hardware resources, at the same level as the application being mapped and scheduled onto them; this unified representation would allow the expression of the search space of program transformations, and would be a necessary step to apply incremental refinement methods (expert-driven, very popular in this domain).

Technically, we extend the classical clock calculus (a type system) of the *Lucid Synchrone* language, expliciting significantly more information about the program behavior, especially when tasks must be started and will be completed, how information flow among tasks, etc. Our main contribution is the integration of relaxed synchronous operators like jittering and bursty streams within synchronous bounds [26]. This research consists in revisiting the semantics of synchronous Kahn networks in the domain of media streaming applications and reconfigurable parallel architectures, in collaboration with Marc Duranton from Philips Research Eindhoven (now NXP Semiconductors) and with Marc Pouzet from LRI and the Proval Inria project team.

CURRENT ACTIVITIES: IST STREP ACOTES, Marie Curie ToK-IAP PSYCHES.

CURRENT PEOPLE: Albert Cohen, Christine Eisenbeis, Mohammed Fellahi.

5.2.2. Passing program semantic using software components

Beyond domain-specific and regular applications (loops and arrays), automatic compiler-based parallelization has achieved only mixed results on programs with complex control and data structures [66]. Writing, and especially debugging, large parallel programs is a notoriously difficult task [69], and one may wonder whether the vast majority of programmers will be able to cope with it. Currently, transactional memory is a popular approach [67] for reducing the programmer burden using intuitive transaction declarations instead of more complex concurrency control constructs. However, it does not depart from the classic approach of parallelizing standard C/C++/Fortran programs, where parallelism can be difficult to extract or manipulate. Parallel languages, such as HPF [73], require more ambitious evolutions of programming habits, but they also let programmers pass more semantic about the control and data characteristics of programs to the compiler for easier and more efficient parallelization. However, one can only observe that, for the moment, few such languages have become popular in practice.

A solution would have a better chance to be adopted by the community of programmers at large if it integrates well with popular practices in *software engineering*, and this aspect of the parallelization problem may have been overlooked. Interestingly, software engineering has recently evolved towards a programming model that can blend well with multi-core architectures and parallelization. Programming has consistently evolved towards more encapsulation: procedures, then objects, then *components* [86]. Essentially for two reasons, because programmers have difficulties grasping large programs and need to think locally, and because encapsulation enables *reuse* of programming efforts. Component-based programming, as proposed in Java Beans, .Net or more ad-hoc component frameworks, is the step beyond C++ or Java objects: programs are decomposed into modules which fully encapsulate code and data (no global variable) and which communicate among themselves through explicit interfaces/links.

Components have many assets for the task of developing parallel programs. (1) Components provide a pragmatic approach for bringing parallelization to the community at large thanks to component reuse. (2) Components provide an implicit and intuitive programming model: the programmer views the program as a "virtual space" (rather than a sequence of tasks) where components reside; two components residing together in the space and not linked or not communicating through an existing link implicitly operate in parallel; this virtual space can be mapped to the physical space of a multi-threaded/multi-core architecture. (3) Provided the architecture is somehow aware of the program decomposition into components, and can manipulate individual components, the compiler (and the user) would be also relieved of the issue of mapping programs to architectures.

In order to use software components for large-scale and fine-grain parallelization, the key notion is to augment them with the ability to split or replicate. For instance, a component walking a binary tree could spawn two components to scan two child nodes and the corresponding subtrees in parallel.

We are investigating a low-overhead component-based approach for fine-grain parallelism where components have the ability to replicate [71], [30]. We investigate both a hardware-supported and software-only approach to component division. We show that a low-overhead component framework, possibly paired with component hardware support, can provide both an intuitive programming model for writing fine-grain parallel programs with complex control flow and data structures, and an efficient platform for parallel components execution.

CURRENT ACTIVITIES: FET IP SARC, IST NoE HiPEAC, ANR APE.

CURRENT PEOPLE: Olivier Certner, Zheng Li, Pierre Palatin, Olivier Temam.

5.3. A biological approach to computing

5.3.1. Architecture simulation and sampling

Participants: Hugues Berry, Daniel Gracia Pérez, Olivier Temam.

Sampling and Simulation

Architecture simulation is usually based on cycle-accurate simulators that are very slow compared to execution times on the real architecture. This problem might even become a real blockage with the forthcoming complex multi-core architectures. One possible solution is to simulate only chosen pieces (i.e. samples) of the whole program and use the simulation of these samples to extrapolate to the whole program. In the past few years, several research works have demonstrated that sampling can drastically speed up architecture simulation, and several of these sampling techniques are already largely used. However, for a sampling technique to be both easily and properly used, it must fulfill a number of conditions: it should require no hardware-dependent modification of the simulator, it should simultaneously consider warm-up of the cache structures and sampling, while still delivering high speed and accuracy.

We recently proposed [17] a sampling technique focused more on transparency than on speed and accuracy, though the technique delivers almost state-of-the-art performance. Our sampling technique is based on algorithms that were originally developed for neurological image analysis. We make the following contributions: (1) a technique for splitting the execution trace into a potentially very large number of variable-size regions to capture program dynamic control flow, (2) a clustering method capable of efficiently coping with such a large number of regions, (3) a budget- based method for jointly considering warm-up and sampling costs, presenting them as a single parameter to the user, and for distributing the number of simulated instructions between warmup and sampling based on the region partitioning and clustering information. Overall, the method achieves an accuracy/time tradeoff that is close to the best reported results using clustering-based sampling (though usually with perfect or hardware-dependent warm-up), with an average CPI error of 1.68% and an average number of simulated instructions of 288 million instructions over the Spec benchmarks. The technique/tool can be readily applied to a wide range of benchmarks, architectures and simulators, and will be used as a sampling option of the UniSim modular simulation framework.

Complex systems analysis of performance

The difficulties encountered in the field of architecture simulators can in part be ascribed to the complexity of the microprocessors themselves. Indeed, performance traces obtained during simulations can be highly variable and difficult to predict. A better understanding and characterization of these complex signals could lead to development of better sampling/simulation techniques. In this context, we studied the time-evolution of performance traces during execution of several prototypical programs on prototypical modern microprocessors [10], [23]. We recorded several metrics characterizing execution performance and memory operations, and analyzed them using current techniques from complex systems sciences (nonlinear time series analysis in particular). These techniques have been used, for example, to analyze and quantify signals from complex physiological systems, such as heartbeat time series (electrocardiograms) or brain waves (electroencephalograms). Besides regular periodic behaviors, we evidenced highly variable performance evolutions for several programs. More interestingly, we showed that the evolution of performance during

the execution of several programs displays clear evidences of deterministic chaos, with sensitivities to initial conditions that are comparable to textbook chaotic systems. This is a clear demonstration that current monoprocessor architectures are, if not complex systems, at least complicated enough to yield such complex behaviors. Future work will focus on applying these analytical tools to concrete simulation issues, and extend our analysis to still more complex architectures, such as chip multiprocessors.

5.3.2. Biological neural networks as bio-inspiration sources for future architectures

Participants: Hugues Berry, Bruno Cessac, Bruno Delord, Mathias Quoy, Benoit Siri, Olivier Temam.

Current trends in the evolution of microprocessor architecture outline systems consisting of a huge number of slow components (possibly even slower than current ones) with a large rate of defects/faults as well as interconnect and placement with variable degrees of irregularity. For architectures to take advantage of such technology, they would have to rely on space much more than time/speed to achieve high performance. The major issues that architects will face at long term are thus how to design, organize and programm these systems, in such a way to warranty scalability. Searching for pieces of solution to this problem has progressively led us to biological neural networks. Indeed, biological neural networks (as opposed to artificial neural networks, ANNs) are well-known examples of systems capable of complex information processing tasks using a large number of self-organized, but slow and unreliable components. And the complexity of the tasks typically processed by biological neurons is well beyond what is classically implemented with ANNs, because ANNs lack key features of biological neural networks. Emulating the workings of biological neural networks may at first sight seem far-fetched. However, the SIA (Semiconductor Industry Association) in its 2005 roadmap addresses for the first time "biologically inspired architecture implementations" as emerging research architectures and focuses on biological neural networks as interesting scalable designs for information processing.

An abstract model for biological neural network growth

Computing machines, such as current processor architectures, are designed using a very abstract model of the physical properties of transistors and circuits. Typically, microprocessor architects do not deal with the complex physics occurring at the transistor level but rely upon a very abstract and simplified model of the undergoing physical phenomena. Similarly, if we want to start thinking about computing systems built upon biological neurons, we must come up with sufficiently abstract models of biological neural networks, that will enable the design of large systems without dealing with the detailed individual behavior of the neurons. For instance, to understand what kind of computing systems can be built upon biological neurons, we must first understand the kind of *structures* into which biological neurons can self-assemble. To this aim, we started with the biological neural network of *Caenorhabditis elegans*, which has been described in great details in the biology literature. Using this map as an oracle, we defined [13] a model of network growth in real space and provided empirical evidence that the characteristics of networks built upon this model and the above mentioned biological network closely match. Since this model describes the network *growth* using simple local rules, it can be used to represent much larger networks, as would be needed for computing systems. In other words, it allows the generation of surrogate networks with structures comparable to that of *Caenorhabditis elegans*. Further works will be devoted to testing the implementation of these bioinspired structures inside real computing systems such as the interconnect network of chip multiprocessors.

Structure and dynamics of recurrent neural networks

A second major research direction consists in the study of the relationship between function, learning and structure in recurrent neural networks (RNNs). RNNs include backward connections, which endow them with a rich variety of dynamical behaviors. Many real biological neuron networks show such high proportions of backward connections. Recurrent neural networks appear thus more interesting to the understanding of computation in large neural networks than the classical feed-forward structures used in most artificial neural networks. Unlike Hopfield-like networks, RNNs exhibit complex dynamics (limit cycles, chaos) and transitions between them. We started to study these systems using complex networks approaches. In this framework, the dynamics of the neurons (the network nodes) depends on synaptic weights (the network links) that themselves vary over time ("learning") as a function of the neuron dynamics. The system can thus be thought of as composed by two coupled layers (one for neuron dynamics and one for network structure) whose mutual coupling remains largely obscure. We proposed to use both a dynamical system and a graph

theory approach in order to understand this mutual coupling [12]. We investigated several local (unsupervised) learning rules to update synaptic strengths. These rules are simple implementations of Hebb's rule for learning in biological neurons (i.e. neurons which fire together become more tightly coupled).
Due to the aforementioned coupling, learning shapes the network dynamics, topology and function. We evidenced that the modifications of the dynamics can be related to changes in the local loop content. We further showed that, because of these local structural alterations, the global network topology changes as well. Indeed, under the influence of learning, the distribution of the strong synapses on the network is no more homogeneous, i.e. two neurons have an increasing probability to be strongly coupled if they are both connected to a third neuron by strong synapses. Besides, the mean-shortest path remains low, so that these learning rules organize the network as a small-world one. Our studies [34], [33] thus progressively uncover a global sketch allowing the understanding of the relationships between dynamics, structure and function in these networks.
Future works in this project will follow two main directions. Firstly, we will extend our fundamental studies of RNNs using tools from spectral graph theory, that should allow us to better delineate how input patterns are actually encoded into the network structure. Secondly, we will apply these fundamental studies to the case of visual object recognition. Our aim here is to develop a bio-inspired model for object recognition in which the network structure would emerge through activity-based local learning rules.

5.3.3. Optimizing the structure of large-size neural networks

Participants: Hugues Berry, Fei Jiang, Marc Schoenauer.

The connectivity structure of complex networks (i.e. their topology) is a crucial determinant of information transfer in large networks (internet, social networks, metabolic networks...). For instance, information, virus or epidemic spreading in complex networks ("small-world" or "scale-free" networks) is much more efficient/fast than in comparable random or regular networks. Other crucial properties of these systems are topology-controlled, such as tolerance to faults/defects (robustness), vaccination or the existence of critical thresholds for epidemic spreading. Several studies have applied such complex networks tools to neural networks. Here again, several functional properties of neural networks seem to depend on the network (complex) topology. For instance, a recent study (Simard *et al., Physics Letters A* 336:8-15,2005) has shown that introducing a "small-world" topology in a monolayer perceptron increases the learning rate of the network. Symmetrically, evolutionary algorithms are commonly used to modify the topology of neural networks so as to optimize their performance. But, in most cases, the studied topologies are quite simple and the number of connections/neurons is low. Furthermore, the evolutionary mechanisms used in most of these studies do not modify the topology in an intensive manner.
Hence, optimization of large neural networks through artificial evolution has hardly been studied. However optimization of complex topologies in related systems has recently begun to be inspected. For instance, Tomassini and collaborators (*Complex Systems* 15:261-284, 2005) have used evolutionary algorithms to optimize the topology of 1D-cellular automata networks and reported that their optimal topologies were systematically close to "small-world" ones. More recently, Oikonomou & Cluzel (*Nature Physics* 2:532-536, 2006) have optimized the topology of boolean networks and found that the evolution of networks with random topology is very different, even quantitatively, from networks with "scale-free" topology. Here, we wish to study the interaction between the topology of large neural networks and their learning capacities. Our approach tackles both the direct and inverse problem:

- *Direct problem*: Given a network with fixed topology, we study how the network learns to perform its target task through local (Hebb's) rules. An important part of this study consists in trying to understand the emergence of modular structures in the networks. Modularity is indeed a common feature in biological neural networks, but is usually not observed in artificial neural networks obtained with artificial evolution. Another important aspect of this part of the study is that quantification of the network efficiency can be defined on the basis of its pure performance, but it may also be based on its robustness to failures, noise, or attacks.

- *Inverse problem*: Given a set of local learning rules and a given evolutionary optimization algorithm acting on the network topology, we study what kind of topology the networks evolve to. In other words, we want to know if there exists such a thing as an optimal topology, for given local learning rules and a given task to perform. Here again, network optimization can concern the pure performance of the network, but, alternatively, it may as well concern its robustness to noise or defects.

This project is a collaboration with project-team TAO (INRIA futurs, Orsay), headed by Marc Schoenauer, and grounds the Ph.D. thesis work of F. Jiang, which is co-supervised and co-funded by both groups.

5.3.4. Individual properties of biological neurons

Participants: Hugues Berry, Bruno Delord, Stephane Genet, Emmanuel Guigon, Loic Sabarly.

The properties of biological neural networks that are of direct interest to architecture research are in part due to the intrinsic properties of the individual neurons. We are collaborating with the neuroscience research lab ANIM (INSERM U742) to develop simulation and modeling studies of specific properties of individual biological neurons such as time handling or plasticity and memory properties.
A major effort in this project is devoted to modeling of the cerebellum. The cerebellum is a non-cortical structure that contains more neurons than the rest of the brain (in mammals) and has been implied in motor control, among others. The repetition of a conserved connectivity motif (i.e. a module) throughout this structure suggests that this motif produces a unique computational process at the basis of most of the cerebellum functions. One hypothesis suggests that the cerebellum basically provides representations of the time interval separating two successive events (in the tenths second range). Purkinje cells (PCs) are the only output of the cerebellar cortex and probably provide essential mechanisms for this temporal computation. But the implied mechanisms are still poorly understood. The most probable mechanism seems to be the generation of a slow response in PCs, such as the dendrite plateau potentials observed experimentally. In this work, we studied the triggering and propagation of the plateaus in spatial representations (cables) of the complex PC dendrite [28]. Our model showed that plateau potentials form an original type of electrical signaling, that cannot be classified as action potentials nor passive electrotonus. Furthermore, our model suggests that plateau potentials could be adaptive signals, whose duration could be learned in order to code time intervals between successive events. The main perspectives of this works consist thus in including synaptic plasticity mechanisms in the model to test its adaptability and time learning properties. In particular, we will use the plasticity model that is currently under study by our group and is able to yield, in a unique dynamical process, plasticity and memory properties which agree with experimental observations at PC synapses. We shall then obtain a module able to model learning of time interval representations in the cerebellum.

6. Contracts and Grants with Industry

6.1. Collaborations involving industry

STMicroelectronics Besides the aforementioned RNTL contract COP, and the HiPEAC network of excellence, and IP SARC, we have a regular and informal collaboration on iterative compilation and novel processor architecture with the AST (Advanced Systems and Technologies) research group of STMicroelectronics based in Lugano, Switzerland and Grenoble, France.

Philips Semiconductors, now NXP We have had regular collaborations with Philips for almost 10 years now, including direct contracts. Currently, we are involved in several grants with Philips (IP SARC, Marie-Curie fellowships, ACOTES). Philips Semiconductors has recently become NXP.

ARM R&D, Cambridge In the context of the SARC FP6 FET Proactive IP project, Pierre Palatin spent a 3 months summer internship at ARM R&D, Cambridge. The goal was the application of Capsule on specific ARM architectures.

6.2. National and international collaborative grants

GGCC: EU, MEDEA+ program ITEA Call 8 project on global analysis and optimization in GCC. Our involvment lie in the compiler infrastructure, static analysis in the polyhedral model, and feature extraction for global and contiunous optimization. With CEA (dpt. of energy), UPM (Spain), SICS (Sweden), major industrial partners (Airbus, Telefonica, Bertin) and SMEs (Mandriva, MySQL, and others). 04/2006–04/2009.

ACOTES: EU, IST program FP6 STREP on language and compiler support for high-performance streaming applications. We are one of the largest contractors in the project, with major involvment in interprocedural optimization and loop transformations for concurrent distributed streaming applications; it is both a programming model and compiler project. With Philips Research (Eindhoven), IBM Research (Haifa), STMicroelectronics (AST Lugano), Nokia (Helsinki), and UPC (Barcelona). 05/2006–05/2009.

MilePost: EU, IST program FP6 STREP on machine-learning compilation. This project matches one of the core directions of the project: iterative optimization research, with an emphasis on making iterative compilation methods practical in real development environments. With IBM Research (Haifa), ARC (London), CAPS Entreprise (Rennes), IRISA (Rennes), and University of Edinburgh. 05/2006–05/2009.

PARA: French Ministry of Research ANR CIGC project on multi-level parallel programming and automatic parallelization. We are involved in automatic code generation approaches for domain-specific and target-specific optimizations; iterative and polyhedral compilation methods are explored in an application-specific context. With Bull, University of Versailles, LaBRI (University of Bordeaux), INT (Evry), CAPS Entreprise (Rennes). 01/2006–01/2009.

APE: French Ministry of Research ANR RNTL project on parallel real-time applications for embedded systems. We are developing a component-based environment called CAPSULE for distributed-memory processors. It will be applied to a novel processor of STMicroelectronics and tested on applications from Thales. With STMicroelectronics, Thales, University of Paris 6, CEA. 01/2006–01/2009.

PSYCHES: EU, IST program Marie Curie ToK IAP (Transfer of Knowledge, Industry-Academia Partnership); long-term exchange of personnel and 2 years of post-doc; with Philips Research (Eindhoven) and UPC (Barcelona). 03/2006–03/2009.

SARC: EU, IST program FP6 FET Proactive IP on advanced computer architecture. The goal is to address all the aspects of a scalable processor architecture based on multi-cores. It includes programming paradigms, compiler optimization, hardware support and simulation issues. CAPSULE is being used as component-based programming approach, and UNISIM for the simulation platform. 01/2006–01/2010.

Embedded TeraOps A SYSTEMATIC “Pôle de Competitivité” regional funding for the development of a large-scale embedded multi-core architectures, coordinated by Thales. It will initially focus on streaming applications but it will later target programs with more complex control flow. Thales, Dassault, Thomson, CEA, INRIA. 01/2006–01/2010.

MODSIM MODSIM is an INRIA grant for a joint international team between INRIA and Princeton University. The goal is the development of the UNISIM simulation platform. With Princeton University. 01/2006–12/2009.

ACI ASTICO Grant French Minister of Research grant to explore biological neuron networks as possible sources of inspiration for future computing systems, with a focus on the complex structure of these networks. Our aim is at the same time to investigate bio-inspired computing systems, and original approaches for the modeling and understanding of biological neural networks. With University of Cergy-Pontoise, University of Nice-Sophia-Antipolis and University of Paris 6. 01/2005–01/2008.

NoE HiPEAC HiPEAC is a network of excellence on High-Performance Embedded Architectures and Compilers. It involves more than 70 European researchers from 10 countries and 6 companies, including ST, Infineon and ARM. The goal of HiPEAC is to steer European research on future processor architectures and compilers to key issues, relevant to the European embedded industry. Olivier Temam is a member of the steering committee.

ACI Nanosys French Minister of Research grant to study the impact of alternative technologies, particularly nanotubes, on future computing circuits and architectures. With a large array of French laboratories in VLSI and architecture design.

RNTL COP The purpose of the project was to bring iterative optimization techniques to general-purpose and embedded processors. With STMicroelectronics, HP, CEA and University of Toulouse. 01/2003–01/2006.

7. Other Grants and Activities

7.1. Informal collaborations

University of Princeton We have an active collaboration with the Liberty group (David August) at University of Princeton in the past year. The goal is to unify our approach in modular simulation within the UNISIM framework and thus increase the likelihood that a joint environment be adopted by the wider community. This interaction is further synchronized with the Common Simulation Platform activity of the HiPEAC network. Starting January 2005, we obtained an "Joint Team" grant called MODSIM, together with the Liberty group at University of Princeton.

University of California Santa Cruz Thanks to a France-Berkeley travel grant, We are starting a collaboration with the group of Jose Renau, thanks to a 2006-2007 France-Berkeley grant. The topics are close to the infrastructure work of Alchemy: fast and accurate simulation of multi-core processors, and support for a modern parallelisation infrastructure in GCC. Jose Renau is a member of the OpenSparc consortium and contributed to major advances in architecture and compiler support for thread-level speculation.

University of Edinburgh For the past 2 years, we had a very active cooperation with University of Edinburgh on iterative optimization; Grigori Fursin, postdoc in our group, got his PhD from University Edinburgh. This collaboration has resulted in a series of joint articles [60], [25], [15].

University of Illinois We have a regular collaboration with the group of David Padua, Urbana-Champaign, Illinois, which started 6 years ago, with multiple joint publications and travel grants (CNRS-UIUC). Research focused on high-performance Java, dependence and alias analysis, processors in memory, and currently on adaptive program generation and machine learning compilers.

Texas A&M University We started a regular exchange of ideas and personnel with the Parasol laboratory, led by Lawrence Rauchwerger, a reference in parallel language compilation and architecture support. ProfṘauchwerger visited Alchemy for a total of 5 months in the last 3 years, and many of us visited TAMU for shorter periods. The collaboration led to numerous advances in the understanding of the main challenges and pitfals in scalable parallel processing, and also facilites the organization of multiple academic events (e.g., the upcoming PACT'07)

UPC We have a regular collaboration with UPC, Barcelona, which started 7 years ago, with several groups on topics ranging from program optimization to micro-architecture, resulting in several publications, joint contracts.

University of Passau We have a regular collaboration with the group of Christian Lengauer and Martin Griebl, Passau, Germany, which started 10 years ago, with multiple joint publications and travel grants (Procope, Ministry of Foreign Affairs). Our collaboration focused on polyhedral compilation techniques and recently headed towards domain-specific program generation and metaprogramming.

Lal-LPT, University of Paris Sud We have started a collaboration with physicists working on LQCD (Lattice Quantic Chromo Dynamics). We focus on the next generation of computer that would gain an order of magnitude speedup over their current APE-next processor (sustained 300 GFlops).

Paris 6 University The properties of biological neural networks that are of direct interest to architecture research are in part due to the intrinsic properties of the individual neurons. We are collaborating with the neuroscience research lab ANIM (INSERM U742) to develop simulation and modelling studies of specific properties of individual biological neurons such as time handling or plasticity and memory properties [28].

Project-Team TAO, INRIA Futurs We started a collaboration with Marc Schoenauer on evolutionary algorithms for optimization of complex systems. More precisely, we study evolutionary methods to optimize the complex structure of large size neural networks. The aim is to find wether there exists optimal organizations for the interconnect network of such large systems. This collaboration grounds F. Jiang's Ph.D. work, which is co-supervised and co-founded by the two groups.

CEA List For the past 6 years, we had a regular collaboration with the *Laboratoire Sûreté du Logiciel* (Software Safety Lab) at CEA LIST on two topics: processor simulation and program optimization. Simulation of complex processor architectures is necessary for the development of software test of complex systems investigated at CEA. Program optimization is more a way to factor in the CEA expertise in static analysis and develop new applications. CEA has funded two scholarships in our group until 2004 and 2005 respectively.

Others We also have regular contacts with several foreign research groups: the CAPSL group at University of Delaware; and the PASCAL group at University of California Irvine (NSF-INRIA grant).

Hugues Berry collaborates with Bruno Cessac (Institut Non Linéaire de Nice, UMR 6618 CNRS / Université Nice-Sophia Antipolis), Bruno Delord (ANIM, UMR 742 Inserm / Université Pierre et Marie Curie, Paris), Stéphane Genet (ANIM, UMR 742 Inserm / Université Pierre et Marie Curie, Paris), Mathias Quoy (ETIS, UMR 8051 CNRS / Université de Cergy-Pontoise / ENSEA), Olivier Michel (Ibisc, Université d'Evry), Marc Schoenauer (TAO, INRIA Futurs, Orsay), Nazim Fates (MAIA, INRIA Loraine, Nancy).

7.2. Seminar and invited scientists

ALCHEMY organizes a joint seminar with CRI (Centre de Recherches en Informatique, Ecole des Mines de Paris), LRI (Laboratoire de Recherches en Informatique, University of Paris-Sud) and PriSM (University of Versailles-Saint-Quentin). Talks of 2006 are given below.

- januray 23rd, 2006, *Hybrid Optimization*, John Cavazos, University of Edinburgh, UK.
- february 24th, 2006, *Temporal Memory Streaming*, Babak Falsafi, Carnegie Mellon University.
- april 28th, 2006, *Dynamic selective compilation and scheduling of Byte-Code*, Prof. Stefano Crespi-Reghizzi, Politecnico di Milano.
- may 5th, 2006, *Efficiently Exploring Architectural Design Spaces via Predictive Modeling*, Sally A. McKee, Cornell University Computer Systems Lab.
- june 2nd, 2006, *The Tensor Contraction Engine (TCE): A High-Level Approach to Synthesizing High-Performance Codes for Quantum Chemistry*, J. Ramanujam, Distinguished Professor, Louisiana State University.
- june 14th, 2006, *A fresh look at the design of high-end parallel computing systems*, Guang R. R. Gao, Endowed Distinguished Professor, University of Delaware.
- november 7th, 2006, *Evolution, Fractals and Complex interactions*, Evelyne Lutton, COMPLEX Team manager, INRIA.
- november 14th, 2006, *Conception, Semantics and Implementation of ReactiveML*, Louis Mandel, MOSCOVA team, INRIA.
- december 12th, 2006, *Extracting coarse-grained parallelism with the slicing framework*, Anna Beletska, Politecnico di Milano.

Erven Rohou (ST-MicroElectronics, Lugano, Switzerland), visited ALCHEMY twice for one week in 2006.

Marc Duranton (Philips NXP, Eindhoven, Netherlands) visits ALCHEMY regularly.

Lawrence Rauchwerger from Texas A&M University spent two months in our group, first as a invited professor supported by LRI in July 2006, and again as an invited professor supported by INRIA Futurs in December 2006.

8. Dissemination

8.1. Leadership within scientific community

Cédric Bastoul

- Member of the LRI department committee at the University of Paris-Sud of Paris-Sud since 2006.
- Member of the Orsay Technology Institute (IUT D'Orsay) Computer Science department committee since 2006.

Albert Cohen

- HiPEAC Summer School course on GCC (55-65 attendees). The support material for the courses and tutorials is freely available (public domain or GPL license) and has been contributed to the main GCC site (`gcc.gnu.org`, Wiki section; see also `www.hipeac.net/gcc-tutorial`).
- HiPEAC GCC Tutorial in Grenoble (France), in May 2006. A second HiPEAC GCC Tutorial will be associated with the second HiPEAC International Conference, taking place in Ghent (Belgium) in January 2007.
- Founding member of IFIP WG 2.11.
- Recruiting committee for INRIA Futurs research scientists, 2004 and 2006.
- Competitive oral examination, Écoles Normales Supérieures (TIPE d'informatique), 2004–2006.

Christine Eisenbeis

- member of IFIP WG 10.3.

Olivier Temam

- ANR Future Processor Architectures grants evaluation committee, 2006.
- HiPEAC Summer School course on UNISIM in 2005, tutorials at UPC in 2006 and scheduled at HiPEAC Conference in 2007.

PROGRAM COMMITTEES:

Albert Cohen

- Co-organizer of Dagstuhl seminar 07361, September 2007, with Sam Midkiff (Purdue), Maria-Jesus Garzaran (University of Illinois at Urbana-Champaign), and Christian Lengauer (Passau University).
- Workshop chair for IEEE Conference on Parallel Architectures and Compilation Techniques (PACT'06, 4 associated workshops).
- Editorial committee of ACM Transactions on Embedded Systems (TECS, special issue on software and compilers).

- Program committee member of the ACM symp. on Principles and Practice of Parallel Programming (PPoPP'07).
- Program committee member of the ACM symp. on Partial Evaluation and Program Manipulation (PEPM'07).

Christine Eisenbeis

- reviewer of the PhDs of Bénédicte Kenmei, Université de Strasbourg, june 27th, 2006, Mauricio Araya, Université de Nice and Inria Sophia-Antipolis, november 24th, 2006, and Rachid Seghir, Université de Strasbourg, december 7th, 2006.
- Software and Compilers for Embedded Systems, SCOPES' 2007, April 2007, Nice.
- IFIP International Conference on Network and Parallel Computing (NPC 2007), September 2007, China.

Frédéric Gruau

- Co-organizer (with Jean-Louis Giavitto, LaMI, Evry and André Dehon, University of Pennsylvania) of the Dagstuhl seminar on "Spatial Computing", September, 2006.

Olivier Temam

- ISCA, International Symposium on Computer Architecture, 2007.
- SMART, Workshop on Statistical and Machine learning approaches applied to ARchitectures and compilaTion, 2007.
- CGO, ACM/IEEE International Symposium on Code Generation and Optimization, 2007.
- HPCA, High-Performance Computer Architecture, 2007.
- MoBS, Workshop on Modeling, Benchmarking and Simulation, 2006.
- ISCA, International Symposium on Computer Architecture, 2006.
- HiPEAC'07, International Conference on High-Performance Embedded Architectures and Compilers, 2007.
- DATE, Design and Automation and Test in Europe, 2006.

8.2. Teaching at university

Hugues Berry gave two courses :

April 13th, 2006, UFR des Sciences Pharmaceutiques de Caen: "Apport de la modélisation mathématique à l'étude de la dynamique des systèmes cellulaires", Cours (3 heures) dans le cadre de l'UE "Biologie moleculaire de la cellule" du Master Sciences Biologiques et Médicales (4ème année des étude de pharmacie).

November 20th, 2006, Université de Cergy Pontoise: Cours (3 heures) dans le cadre de la seconde année du Master Recherche "Matière Organisée et Systèmes Vivants".

Olivier Temam teaches a computer architecture course at Ecole Polytechnique to 3rd-year students on computer architectures (appr. 35 hours). He also teaches a course on novel processor architectures at University of Paris Sud to Master's students.

8.3. Workshops, seminars, invitations

The project-team members have given the following talks and attended the following conferences:

Hugues Berry

- Complex dynamics of microprocessor performances during program execution: regularity, chaos, and others, Conference NKS 2006, Washington D.C., USA, 2006.
- Structure complexe et émergence dans les réseaux de neurones biologiques, Workshop AMINA, Monastir, Tunisia, 2006. Chairman of the session “Implantations Parallèles”.
- Complex biological systems and computing systems, Evry University & Genopole center, France, 2006.
- Caractérisation de la performance des microprocesseurs pendant l’exécution de programmes, 9emes Rencontres du Non Linéaire, Paris, France, 2006.
- Atelier d’épigénomique, Génopole, Evry, 27 janvier 2006.
- Séminaires de l’unité INSERM ANIM, U472, Université P. et M. Curie, february 1st, 2006.
- Séminaires de l’équipe “Bio-informatique”, L.R.I, Orsay, february 23rd, 2006.
- Journée Digiteo Labs “Architecture”, CEA, Saclay, april 27th, 2006.
- Séminaires de l’ACI “Nanosys”, ENST, Paris, october 6th, 2006.

Christine Eisenbeis

- Lal-LRI meeting, march 21st, 2006.
- Pérenniser la loi de Moore?, Cinquièmes Journées Informatique de l’IN2P3 et du DAPNIA, september, 2006.
- Dagstuhl seminar on “Spatial Computing”, September, 2006; talk on “N-synchronous Kahn networks”.

Sylvain Girbal

- First UNISIM tutorial: Learning the Basics of UNISIM. 5 day tutorial at Barcelona UPC in February 2006.
- Second UNISIM Tutorial: Prise en main de l’environnement de simulation structurelle UNISIM. Tutoriel d’une journée présenté le 3 octobre 2006 à Sympa’06, Perpignan France.
- UNISIM tutorial at Philips NXP. 3 days tutorial in Eindhoven, Netherland on December 18-20 2006.

Frédéric Gruau

- Dagstuhl seminar on “Spatial Computing”, September, 2006; talk on “Blob computing”.

Pierre Palatin

- Talk at Micro-39 (9–13 Decembre 2006, Orlando, Florida, USA); "CAPSULE: Hardware-Assisted Parallel Execution of Component-Based Programs".
- Talk at Hipeac 1st Industrial Workshop (11th May 2006, Grenoble, France), with ST-Microelectronics : "CAPSULE: Hardware-Assisted Parallel Execution of Component-Based Programs".
- Informal presentation of "Capsule" at a NANOSYS project meeting (5th November 2006). NANOSYS is an "ACI nanosciences" project called "Architectures pour l’intégration des nanocomposants moléculaires".

9. Bibliography

Major publications by the team in recent years

[1] F. AGAKOV, E. BONILLA, J. CAVAZOS, B. FRANKE, G. FURSIN, M. O'BOYLE, J. THOMSON, M. TOUSSAINT, C. WILLIAMS. *Using Machine Learning to Focus Iterative Optimization*, in "Proceedings of the 4th Annual International Symposium on Code Generation and Optimization (CGO)", 2006.

[2] C. BASTOUL, A. COHEN, S. GIRBAL, S. SHARMA, O. TEMAM. *Putting Polyhedral Loop Transformations to Work*, in "Workshop on Languages and Compilers for Parallel Computing (LCPC'03), College Station, Texas", LNCS, Springer-Verlag, October 2003, p. 23–30, http://www-rocq.inria.fr/~acohen/publications/BCGST03.ps.gz.

[3] H. BERRY, D. GRACIA PÉREZ, O. TEMAM. *Chaos in computer performance*, in "Chaos", vol. 16, 2006, 013110, http://hal.inria.fr/inria-00000109/en/.

[4] A. COHEN, M. DURANTON, C. EISENBEIS, C. PAGETTI, F. PLATEAU, M. POUZET. *N-Sychronous Kahn Networks*, in "33th ACM Symp. on Principles of Programming Languages (PoPL'06), Charleston, South Carolina", January 2006, p. 180–193, http://www-rocq.inria.fr/~acohen/publications/CDEPPP06.ps.gz.

[5] F. GRUAU, Y. LHUILLIER, P. REITZ, O. TEMAM. *Blob Computing*, in "Computing Frontiers 2004 ACM SIGMicro.", 2004, http://blob.lri.fr/publication/2004-model-blob-machine.pdf.

[6] D. PARELLO, O. TEMAM, J.-M. VERDUN. *On increasing architecture awareness in program optimizations to bridge the gap between peak and sustained processor performance : Matrix-Multiply revisited*, in "Supercomputing", IEEE, Nov 2002.

[7] S. POP, A. COHEN, G.-A. SILBER. *Induction Variable Analysis with Delayed Abstractions*, in "Intl. Conf. on High Performance Embedded Architectures and Compilers (HiPEAC'05), Barcelona, Spain", LNCS, n^{o} 3793, Springer-Verlag, November 2005, p. 218–232, http://www-rocq.inria.fr/~acohen/publications/PCS05.ps.gz.

[8] D. G. PÉREZ, G. MOUCHARD, O. TEMAM. *MicroLib: A Case for the Quantitative Comparison of Micro-Architecture Mechanisms*, in "MICRO-37: Proceedings of the 37th International Symposium on Microarchitecture", IEEE Computer Society, Dec 2004, p. 43–54.

[9] N. VASILACHE, C. BASTOUL, S. GIRBAL, A. COHEN. *Violated dependence analysis*, in "Proceedings of the ACM International Conference on Supercomputing (ICS'06), Cairns, Australia", ACM, June 2006.

Year Publications

Articles in refereed journals and book chapters

[10] H. BERRY, D. GRACIA PÉREZ, O. TEMAM. *Chaos in computer performance*, in "Chaos", vol. 16, 2006, 013110, http://hal.inria.fr/inria-00000109/en/.

[11] H. BERRY, D. PELLENC, O. GALLET. *Adsorption-induced fibronectin aggregation and fibrillogenesis*, in "Journal of Colloid and Interface Science", 2006, http://hal.inria.fr/inria-00001063/en/.

[12] H. BERRY, M. QUOY. *Structure and dynamics of random recurrent neural networks*, in "Adaptive Behavior", vol. 14, 2006, p. 129-137.

[13] H. BERRY, O. TEMAM. *Modeling Self-Developping Biological Neural Networks*, in "Neurocomputing", in press, 2006.

[14] A. COHEN, S. DONADIO, M.-J. GARZARAN, C. HERRMANN, O. KISELYOV, D. PADUA. *In Search of a Program Generator to Implement Generic Transformations for High-Performance Computing*, in "Science of Computer Programming", vol. 62, n^{o} 1, September 2006, p. 25-46.

[15] G. FURSIN, A. COHEN, M. F. P. O'BOYLE, O. TEMAM. *Quick and practical run-time evaluation of multiple program optimizations*, in "Trans. on High Performance Embedded Architectures and Compilers", vol. 1, n^{o} 1, 2006, p. 13-31.

[16] S. GIRBAL, N. VASILACHE, C. BASTOUL, A. COHEN, D. PARELLO, M. SIGLER, O. TEMAM. *Semi-Automatic Composition of Loop Transformations for Deep Parallelism and Memory Hierarchies*, in "Intl. J. of Parallel Programming", Accepted with minor revisions, 2006.

[17] D. GRACIA PÉREZ, H. BERRY, O. TEMAM. *The Practicality Dimension of Sampling*, in "IEEE Micro", vol. 26, 2006, p. 14–28.

[18] W. JALBY, C. LEMUET, S. TOUATI. *An Efficient Memory Operations Optimization Technique for Vector Loops on Itanium 2 Processors*, in "Concurrency and Computation: Practice and Experience", vol. 11, n^{o} 11, September 2006, p. 1485–1508.

[19] S. LONG, G. FURSIN. *Systematic search within an optimisation space based on Unified Transformation Framework*, in "Accepted for the special issue of the International Journal of Computational Science and Engineering (IJCSE)", 2006.

Publications in Conferences and Workshops

[20] F. AGAKOV, E. BONILLA, J. CAVAZOS, B. FRANKE, G. FURSIN, M. O'BOYLE, J. THOMSON, M. TOUSSAINT, C. WILLIAMS. *Using Machine Learning to Focus Iterative Optimization*, in "Proceedings of the 4th Annual International Symposium on Code Generation and Optimization (CGO)", 2006.

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

[22] H. BERRY. *Structure complexe et émergence dans les réseaux de neurones biologiques*, in "3ème Workshop Applications Médicales de l'Informatique: Nouvelles Approches, AMINA 2006, Monastir, Tunisia", November 2006.

[23] H. BERRY, D. GRACIA PÉREZ, O. TEMAM. *Caractérisation de la performance des microprocesseurs pendant l'exécution de programmes*, in "9emes Rencontres du Non LinÃ©aire, Paris, France", March 2006.

[24] H. BERRY, D. GRACIA PÉREZ, O. TEMAM. *Complex dynamics of microprocessor performances during program execution: Regularity, chaos, and others*, in "NKS2006 Wolfram Science Conference, Washington D.C., USA", June 2006.

[25] J. CAVAZOS, C. DUBACH, F. AGAKOV, E. BONILLA, M. O'BOYLE, G. FURSIN, O. TEMAM. *Automatic Performance Model Construction for the Fast Software Exploration of New Hardware Designs*, in "International Conference on Compilers, Architecture, And Synthesis For Embedded Systems (CASES 2006)", To appear, October 2006.

[26] A. COHEN, M. DURANTON, C. EISENBEIS, C. PAGETTI, F. PLATEAU, M. POUZET. *N-Sychronous Kahn Networks*, in "33th ACM Symp. on Principles of Programming Languages (PoPL'06), Charleston, South Carolina", January 2006, p. 180–193.

[27] B. DAUVERGNE, L. HASCOËT. *The Data-Flow Equations of Checkpointing in Reverse Automatic Differentiation*, in "Workshop on Automatic Differentiation: Tools and Applications, Reading", May 2006.

[28] S. GENET, B. DELORD, L. SABARLY, E. GUIGON, H. BERRY. *On the propagation of Ca-dependent plateau and valley potentials in cerebellar Purkinje cells and how they drive the cell output*, in "Proceedings of NeuroComp'06, Pont-à-Mousson, France", 23-24 October 2006, p. 167–170.

[29] D. GRACIA PÉREZ, H. BERRY, O. TEMAM. *Budgeted Region Sampling (BeeRS): Do Not Separate Sampling From Warm-Up, And Then Spend Wisely Your Simulation Budget*, in "5th IEEE International Symposium on Signal Processing and Information Technology 5th IEEE International Symposium on Signal Processing and Information Technology, Athens, Greece", 2006, http://hal.inria.fr/inria-00001061/en/.

[30] P. PALATIN, Y. LHUILLIER, O. TEMAM. *Capsule : Hardware-Assisted Parallel Execution of Component-Based Programs*, in "The 39th Annual IEEE/ACM International Symposium on Microarchitecture, 2006, Orlando, Florida", december 2006.

[31] S. POP, A. COHEN, C. BASTOUL, S. GIRBAL, P. JOUVELOT, G.-A. SILBER, N. VASILACHE. *GRAPHITE: Loop optimizations based on the polyhedral model for GCC*, in "Proc. of the 4th GCC Developper's Summit (to appear), Ottawa, Canada", June 2006.

[32] S. POP, A. COHEN, P. JOUVELOT, G.-A. SILBER. *The New Framework for Loop Nest Optimization in GCC: from Prototyping to Evaluation*, in "Proc. of the 12th Workshop Compilers for Parallel Computers (CPC'06), A Coruña, Spain", January 2006, http://www-rocq.inria.fr/~acohen/publications/PCJS06.ps.gz.

[33] B. SIRI, H. BERRY, B. CESSAC, B. DELORD, M. QUOY. *Topological and dynamical structures induced by Hebbian learning in random neural networks*, in "International Conference on Complex Systems, ICCS 2006, Boston, MA, USA", June 2006.

[34] B. SIRI, H. BERRY, B. CESSAC, B. DELORD, M. QUOY, O. TEMAM. *Learning-induced topological effects on dynamics in neural networks*, in "Proceedings of NeuroComp'06, Pont-à-Mousson, France", 23-24 October 2006, p. 206–209.

[35] S. TOUATI, D. BARTHOU. *On the Decidability of Phase Ordering Problem in Optimizing Compilation*, in "Proceedings of the International Conference on Computing Frontiers, Ischia, Italy", ACM, May 2006, http://www.prism.uvsq.fr/~touati/publis/CF06.pdf.

[36] N. VASILACHE, C. BASTOUL, A. COHEN. *Polyhedral Code Generation in the Real World*, in "Proceedings of the International Conference on Compiler Construction (ETAPS CC'06), Vienna, Austria", LNCS, Springer-Verlag, March 2006, p. 185–201.

[37] N. VASILACHE, C. BASTOUL, S. GIRBAL, A. COHEN. *Violated dependence analysis*, in "Proceedings of the ACM International Conference on Supercomputing (ICS'06), Cairns, Australia", ACM, June 2006.

Miscellaneous

[38] F. AGAKOV, J. CAVAZOS, G. FURSIN, M. O'BOYLE, O. TEMAM. *Predicting Good Compiler Optimizations using Performance CountersACM International Conference on Code Generation and Optimization (CGO'07), San Jose, California*, to appear, March 2007.

[39] L.-N. POUCHET, C. BASTOUL, A. COHEN, N. VASILACHE. *Iterative optimization in the polyhedral model: Part I, one-dimensional timeACM International Conference on Code Generation and Optimization (CGO'07), San Jose, California*, to appear, March 2007.

References in notes

[40] *SystemC v2.0.1 Language Reference Manual*, 2003, http://www.systemc.org/.

[41] *UNISIM: UNIted SIMulation environment*, http://unisim.org.

[42] F. AGAKOV, E. BONILLA, J. CAVAZOS, B. FRANKE, G. FURSIN, M. O'BOYLE, J. THOMSON, M. TOUSSAINT, C. WILLIAMS. *Using Machine Learning to Focus Iterative Optimization*, in "CGO-4: The Fourth Annual International Symposium on Code Generation and Optimization", 2006.

[43] C. BASTOUL. *Code Generation in the Polyhedral Model Is Easier Than You Think*, in "PACT'13 IEEE International Conference on Parallel Architecture and Compilation Techniques, Juan-les-Pins", september 2004, p. 7–16, http://hal.ccsd.cnrs.fr/ccsd-00017260.

[44] C. BASTOUL, A. COHEN, S. GIRBAL, S. SHARMA, O. TEMAM. *Putting Polyhedral Loop Transformations to Work*, in "Workshop on Languages and Compilers for Parallel Computing (LCPC'03), College Station, Texas", LNCS, Springer-Verlag, October 2003, p. 23–30.

[45] P. BERUBE, J. AMARAL. *Aestimo: a feedback-directed optimization evaluation tool*, in "Proceedings of the International Symposium on Performance Analysis of Systems and Software (ISPASS)", 2006.

[46] N. L. BINKERT, R. G. DRESLINSKI, L. R. HSU, K. T. LIM, A. G. SAIDI, S. K. REINHARDT. *The M5 Simulator: Modeling Networked Systems*, in "IEEE Micro", vol. 26, n^{o} 4, 2006, p. 52–60.

[47] P. CARRIBAULT, A. COHEN, W. JALBY. *Deep Jam: Conversion of Coarse-Grain Parallelism to Instruction-Level and Vector Parallelism for Irregular Applications*, in "Parallel Architectures and Compilation Techniques (PACT'05), St-Louis, Missouri", IEEE Computer Society, September 2005, p. 291–300.

[48] Z. CHAMSKI, M. DURANTON, A. COHEN, C. EISENBEIS, P. FEAUTRIER, D. GENIUS. *Ambient Intelligence: Impact on Embedded-System Design*, chap. Application Domain-Driven System Design for Pervasive Video Processing, Kluwer Academic Press, 2003.

[49] A. COHEN, D. GENIUS, A. KORTEBI, Z. CHAMSKI, M. DURANTON, P. FEAUTRIER. *Multi-Periodic Process Networks: Prototyping and Verifying Stream-Processing Systems*, in "Euro-Par'02, Paderborn, Germany", LNCS, vol. 2400, Springer-Verlag, August 2002.

[50] A. COHEN, S. GIRBAL, D. PARELLO, M. SIGLER, O. TEMAM, N. VASILACHE. *Facilitating the Search for Compositions of Program Transformations*, in "ACM Intl. Conf. on Supercomputing (ICS'05), Boston, Massachusetts", June 2005, p. 151–160.

[51] A. COHEN, S. GIRBAL, O. TEMAM. *A Polyhedral Approach to Ease the Composition of Program Transformations*, in "Euro-Par'04, Pisa, Italy", LNCS, n^{o} 3149, Springer-Verlag, August 2004, p. 292–303.

[52] K. D. COOPER, A. GROSUL, T. J. HARVEY, S. REEVES, D. SUBRAMANIAN, L. TORCZON, T. WATERMAN. *ACME: adaptive compilation made efficient*, in "Proceedings of the Conference on Languages, Compilers, and Tools for Embedded Systems (LCTES)", 2005, p. 69–77.

[53] K. D. COOPER, D. SUBRAMANIAN, L. TORCZON. *Adaptive Optimizing Compilers for the 21st Century*, in "J. Supercomput.", vol. 23, n^{o} 1, 2002, p. 7–22.

[54] M. DUPRÉ, N. DRACH, O. TEMAM. *Quickly building an optimizer for complex embedded architectures*, in "International Symposium on Code Generation and Optimization", ACM/IEEE, Mar 2004.

[55] P. FEAUTRIER. *Dataflow Analysis of Array and scalar references*, in "Int. J. of Parallel Programming", vol. 20, n^{o} 1, 1991, p. 23-53.

[56] P. FEAUTRIER. *Some efficient solutions to the affine scheduling problem I. One-dimensional time*, in "Int. J. of Parallel Programming", vol. 21, n^{o} 5, 1992, p. 313-347.

[57] P. FEAUTRIER. *Some efficient solutions to the affine scheduling problem II. Multi-dimensional time*, in "Int. J. of Parallel Programming", vol. 21, n^{o} 6, 1992, p. 389-420.

[58] B. FRANKE, M. O'BOYLE, J. THOMSON, G. FURSIN. *Probabilistic Source-Level Optimisation of Embedded Programs*, in "Proceedings of the Conference on Languages, Compilers, and Tools for Embedded Systems (LCTES)", 2005.

[59] B. FRANKE, M. O'BOYLE, J. THOMSON, G. FURSIN. *Probabilistic Source-Level Optimisation of Embedded Systems Software.*, in "Proceedings of the Conference on Languages, Compilers, and Tools for Embedded Systems (LCTES'05)", june 2005, p. pages 78-86.

[60] G. FURSIN, A. COHEN, M. O'BOYLE, O. TEMAM. *A Practical Method For Quickly Evaluating Program Optimizations*, in "Intl. Conf. on High Performance Embedded Architectures and Compilers (HiPEAC'05), Barcelona, Spain", LNCS, n^{o} 3793, Springer-Verlag, November 2005, p. 29–46, http://hal.inria.fr/inria-00001054/en/.

[61] G. FURSIN, M. O'BOYLE, P. KNIJNENBURG. *Evaluating Iterative Compilation*, in "Proc. Languages and Compilers for Parallel Computers (LCPC)", 2002, p. 305-315.

[62] S. GIRBAL, G. MOUCHARD, A. COHEN, O. TEMAM. *DiST: A Simple, Reliable and Scalable Method to Significantly Reduce Processor Architecture Simulation Time*, in "Intl. Conf. on Measurement and Modeling of Computer Systems, ACM SIGMETRICS'03, San Diego, California", June 2003.

[63] S. C. GOLDSTEIN, M. BUDIU. *NanoFabrics: spatial computing using molecular electronics*, in "Proceedings of the 28th annual international symposium on Computer architecture, Göteborg, Sweden", ACM Press, 2001, p. 178–191.

[64] D. GRACIA PÉREZ, H. BERRY, O. TEMAM. *IDDCA: A New Clustering Approach For Sampling*, in "MoBS: Workshop on Modeling, Benchmarking, and Simulation MoBS: Workshop on Modeling, Benchmarking, and Simulation, Madison, Wisconsin", 2005, http://hal.inria.fr/inria-00001062/en/.

[65] M. R. GUTHAUS, J. S. RINGENBERG, D. ERNST, T. M. AUSTIN, T. MUDGE, R. B. BROWN. *MiBench: A free, commercially representative embedded benchmark suite.*, in "IEEE 4th Annual Workshop on Workload Characterization, Austin, TX", December 2001..

[66] M. H. HALL, S. P. AMARASINGHE, B. R. MURPHY, S.-W. LIAO, M. S. LAM. *Detecting coarse-grain parallelism using an interprocedural parallelizing compiler*, in "Supercomputing '95: Proceedings of the 1995 ACM/IEEE conference on Supercomputing (CDROM), New York, NY, USA", ACM Press, 1995, 49.

[67] L. HAMMOND, V. WONG, M. CHEN, B. D. CARLSTROM, J. D. DAVIS, B. HERTZBERG, M. K. PRABHU, H. WIJAYA, C. KOZYRAKIS, K. OLUKOTUN. *Transactional Memory Coherence and Consistency*, in "Proceedings of the 31st Annual International Symposium on Computer Architecture", IEEE Computer Society, June 2004, 102, http://tcc.stanford.edu/publications/tcc_isca2004.pdf.

[68] M. HANEDA, P. KNIJNENBURG, H. WIJSHOFF. *On the Impact of Data Input Sets on Statistical Compiler Tuning*, in "Workshop on Performance Optimization for High-Level Languages and Libraries (POHLL)", 2006.

[69] J. HUSELIUS. *Debugging Parallel Systems: A State of the Art Report*, Technical report, n^{o} 63, Mälardalen University, Department of Computer Science and Engineering, September 2002, citeseer.ist.psu.edu/huselius02debugging.html.

[70] P. KULKARNI, S. HINES, J. HISER, D. WHALLEY, J. DAVIDSON, D. JONES. *Fast searches for effective optimization phase sequence*, in "Proc. ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI)", 2004.

[71] Y. LHUILLIER, O. TEMAM. *AP+SOMT: AgentProgramming SelfOrganized*, in "International Workshop on Complexity-Effective Design, Munich, Germany", ISCA, May 2004.

[72] S. LONG, G. FURSIN. *A heuristic search algorithm based on Unified Transformation Framework*, in "Proceedings of the 7th International Workshop on High Performance Scientific and Engineering Computing (HPSEC-05)", june 2005, p. pages 137-144.

[73] D. B. LOVEMAN. *High Performance Fortran*, in "IEEE Parallel Distrib. Technol.", vol. 1, n^{o} 1, 1993, p. 25–42.

[74] P. S. MAGNUSSON, M. CHRISTENSSON, J. ESKILSON, D. FORSGREN, G. HALLBERG, J. HOGBERG, F. LARSSON, A. MOESTEDT, B. WERNER. *Simics: A Full System Simulation Platform*, in "Computer", vol. 35, n^{o} 2, 2002, p. 50-58.

[75] D. E. MAYDAN, J. L. HENNESSY, M. S. LAM. *Efficient and Exact Data Dependency Analysis*, in "Proceedings of the SIGPLAN '91 Conference on Programming Language Design and Implementation", June 1991, p. 1-14.

[76] G. MOUCHARD, D. GRACIA PÉREZ, O. TEMAM. *FastSysC: A Fast Simulation Engine*, in "Design, Automation and Test in Europe (DATE), Paris, France", 2004, http://hal.inria.fr/inria-00001108.

[77] G. MOUCHARD. *PowerPC G3 simulator*, 2002, http://www.microlib.org.

[78] M. O'BOYLE, P. KNIJNENBURG, G. FURSIN. *Feedback Assisted Iterative Compiplation*, in "Parallel Architectures and Compilation Techniques (PACT'01)", IEEE Computer Society Press, October 2001.

[79] D. PARELLO, O. TEMAM, A. COHEN, J.-M. VERDUN. *Towards a Systematic, Pragmatic and Architecture-Aware Program Optimization Process for Complex Processors*, in "ACM Supercomputing'04, Pittsburgh, Pennsylvania", November 2004, 15.

[80] D. PARELLO, O. TEMAM, J.-M. VERDUN. *On increasing architecture awareness in program optimizations to bridge the gap between peak and sustained processor performance: matrix-multiply revisited.*, in "SC", 2002, p. 1-11, http://gala.univ-perp.fr/~dparello/publis/on_increasing_architecture_awareness.pdf.

[81] T. POGGIO, C. R. SHELTON. *Machine Learning, Machine Vision, and the Brain*, in "The AI Magazine", vol. 20, n^{o} 3, 1999, p. 37–55, citeseer.ist.psu.edu/poggio99machine.html.

[82] S. POP, A. COHEN, P. JOUVELOT, G.-A. SILBER. *The New Framework for Loop Nest Optimization in GCC: from Prototyping to Evaluation*, in "Proc. of the 12th Workshop Compilers for Parallel Computers (CPC'06), A Coruña, Spain", January 2006.

[83] W. PUGH. *The Omega test: A fast and practical integer programming algorithm for dependence analysis*, in "Comm. of the ACM", vol. 8, 1992, p. 102-114.

[84] D. G. PÉREZ, G. MOUCHARD, O. TEMAM. *MicroLib: A Case for the Quantitative Comparison of Micro-Architecture Mechanisms*, in "MICRO-37: Proceedings of the 37th International Symposium on Microarchitecture", IEEE Computer Society, Dec 2004, p. 43–54.

[85] SIA. *Semiconductor Industry Association 2005 roadmap, section on Emerging Research Devices*, 2005, http://www.sia-online.org/.

[86] C. SZYPERSKI. *Component Software: Beyond Object-Oriented Programming*, Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2002.

[87] M. WOLF, M. LAM. *A loop transformation theory and an algorithm to maximize parallelism*, in "IEEE Transactions on Parallel and Distributed Systems", vol. 2, n^{o} 4, 1991, p. 430-439.

[88] S. YEHIA, O. TEMAM. *From Sequences of Dependent Instructions to Functions: a Complexity-Effective Approach for Improving Performance without ILP or Speculation*, in "International Workshop on Complexity-Effective Design", ISCA, Jun 2003.