IN PARTNERSHIP WITH:
CNRS

Université Rennes 1

Activity Report 2011

Project-Team ALF

Amdahl's Law is Forever

IN COLLABORATION WITH: Institut de recherche en informatique et systèmes aléatoires (IRISA)

RESEARCH CENTER
Rennes - Bretagne-Atlantique

THEME
Architecture and Compiling

Table of contents

Project-Team ALF

Keywords: Processors, Compiling, Real-Time, Complexity

1. Members

Research Scientists
André Seznec [Team leader, Research Director Inria, HdR]
Pierre Michaud [Research scientist Inria]
Erven Rohou [Resarch Director Inria from 03/10/11]

Faculty Members
François Bodin [Professor on leave, External collaborator, HdR]
Isabelle Puaut [Professor, University of Rennes 1, HdR]
Damien Hardy [ATER, University of Rennes 1 till 31/09/11]

Technical Staff
Erven Rohou [till 02/10/11]
David Yuste [till 30/11/11]

PhD Students
Junjie Lai [Inria Allocation]
Ricardo Andrés Velasquéz [Inria Allocation]
Nathanaël Prémillieu [MESR Allocation, University of Rennes 1]
Benjamin Lesage [MESR Allocation, University of Rennes 1]
Luis-Germán García Morales [Inria Allocation from 03/10/11]
Bharath Narasimha Swamy [Inria Allocation from 12/09/11]

Post-Doctoral Fellow
Mridha-Mohammad Waliullah [ERCIM fellowship from 01/04/11 until 30/11/11]

Administrative Assistants
Maryse Fouché [TR Inria till 13/11/11]
Evelyne Livache [TR Inria from 14/11/11]

2. Overall Objectives

2.1. Panorama

Multicore processors have now become mainstream for both general-purpose and embedded computing. In the near future, every hardware platform will feature thread level parallelism. Therefore, the overall computer science research community, but also industry, is facing new challenges; parallel architectures will have to be exploited by every application from HPC computing, web and entreprise servers, but also PCs, smartphones and ubiquitous embedded systems.

Within a decade, it will become technologically feasible to implement 1000s of cores on a single chip. However, several challenges must be addressed to allow the end-user to benefit from these 1000's cores chips. At that time, most applications will not be fully parallelized, therefore the effective performance of most computer systems will strongly depend on their performance on sequential sections and sequential control threads: Amdahl's law is forever. Parallel applications will not become mainstream if they have to be adapted to each new platform, therefore a simple performance scalability/portability path is needed for these applications. In many application domains, particularly in real-time systems, the effective use of multicore chips will depend on the ability of the software and hardware vendors to accurately assess the performance of applications.

The ALF team regroups researchers in computer architecture, software/compiler optimization, and real-time systems. The long-term goal of the ALF project-team is to allow the end-user to benefit from the 2020's many-core platform. We address this issue through architecture, i.e. we try to influence the definition of the 2020's many-core architecture, compiler, i.e. we intend to provide new code generation techniques for efficient execution on many-core architectures and performance prediction/guarantee, i.e. we try to propose new software and architecture techniques to predict/guarantee the response time of many-core architectures.

High performance on single thread process and sequential code is a key issue for enabling overall high performance on a 1000's cores system. Therefore, we anticipate that future manycore architectures will feature heterogeneous design with many simple cores and a few complex cores. Hence the research in the ALF project focuses on refining the microarchitecture to achieve high performance on single thread process and/or sequential code sections. We focus our architecture research in two main directions 1) enhancing the microarchitecture of high-end superscalar processors, 2) exploiting/modifying heterogeneous multicore architecture on a single thread. We also tackle a technological/architecture issue, the temperature wall.

Compilers are keystone solutions for any approach that deals with high performance on 100+ core systems. But general-purpose compilers try to embrace so many domains and try to serve so many constraints that they frequently fail to achieve very high performance. They need to be deeply revisited. We identify four main compiler/software related issues that must be addressed in order to allow efficient use of multi- and many-cores: 1) programming 2) resource management 3) application deployment 4) portable performance. Addressing these challenges require to revisit parallel programming and code generation extensively.

While compiler and architecture research efforts often focus on maximizing average case performance, applications with real-time constraints do not only need high performance but also performance guarantees in all situations, including the worst-case situation. Worst-Case Execution Time estimates (WCET) need to be upper bounds of any possible execution time. The amount of safety required depends on the criticality of applications. Within the ALF team, our objective is to study performance guarantees for both (i) sequential codes running on complex cores ; (ii) parallel codes running on multicores.

Our research is partially supported by industry (IBM, STmicroelectronics), the Brittany region, the ANR (PataQCD project), and the European Union (NoE HiPEAC2 and ERC grant DAL).

2.2. Highlights

André Seznec has been awarded an ERC advanced investigator grant for 2011-2016 called DAL, Defying Amdahl's Law.

André Seznec won the 3rd Championship Branch Prediction in both the conditional branch prediction track and the indirect branch prediction track.

3. Scientific Foundations

3.1. Motivations

Multicores have become mainstream in general-purpose as well as embedded computing in the last few years. The integration technology trend allows to anticipate that a 1000-core chip will become feasible before 2020. On the other hand, while traditional parallel application domains, e.g. supercomputing and transaction servers, are benefiting from the introduction of multicores, there are very few new parallel applications that have emerged during the last few years.

In order to allow the end-user to benefit from the technological breakthrough, new architectures have to be defined for the 2020's many-cores, new compiler and code generation techniques as well as new performance prediction/guarantee techniques have to be proposed .

3.2. The context

3.2.1. Technological context: The advent of multi- and many- cores architecture

For almost 30 years since the introduction of the first microprocessor, the processor industry was driven by the Moore's law till 2002, delivering performance that doubed every 18-24 months on a uniprocessor. However since 2002 , and despite new progress in integration technology, the efforts to design very aggressive and very complex wide issue superscalar processors have essentially been stopped due to poor performance returns, as well as power consumption and temperature walls.

Since 2002-2003, the microprocessor industry has followed a new path for performance: the so-called multicore approach, i.e., integrating several processors on a single chip. This direction has been followed by the whole processor industry. At the same time, most of the computer architecture research community has taken the same path, focusing on issues such as scalability in multicores, power consumption, temperature management and new execution models, e.g. hardware transactional memory.

In terms of integration technology, the current trend will allow to continue to integrate more and more processors on a single die. Doubling the number of cores every two years will soon lead to up to a thousand processor cores on a single chip. The computer architecture community has coined these future processor chips as many-cores.

3.2.2. The application context: multicores, but few parallel applications

For the past five years, small scale parallel processor chips (hyperthreading, dual and quad-core) have become mainstream in general-purpose systems. They are also entering the high-end embedded system market. At the same time, very few (scalable) mainstream parallel applications have been developed. Such development of scalable parallel applications is still limited to niche market segments (scientific applications, transaction servers).

3.2.3. The overall picture

Till now, the end-user of multicores is experiencing improved usage comfort because he/she is able to run several applications at the same time. Eventually, in the near future with the 8-core or the 16-core generation, the end-user will realize that he/she is not experiencing any functionality improvement or performance improvement on current applications. The end-user will then realize that he/she needs more effective performance rather than more cores. The end-user will then ask either for parallel applications or for more effective performance on sequential applications.

3.3. Technology induced challenges

3.3.1. The power and temperatures walls

The power and the temperature walls largely contributed to the emergence of the small-scale multicores. For the past five years, mainstream general-purpose multicores have been built by assembling identical superscalar cores on a chip (e.g. IBM Power series). No new complex power hungry mechanisms were introduced in the core architectures, while power saving techniques such as power gating, dynamic voltage and frequency scaling were introduced. Therefore, since 2002, the designers have been able to keep the power consumption budget and the temperature of the chip within reasonable envelopes while scaling the number of cores with the technology.

Unfortunately, simple and efficient power saving techniques have already caught most of the low hanging fruits on energy consumption. Complex power and thermal management mechanisms are now becoming mainstream; e.g. the Intel Montecito (IA64) features an adjunct (simple) core which unique mission is to manage the power and temperature on two cores. Processor industry will require more and more heroic efforts on this power and temperature management policy to maintain its current performance scaling path. Hence the power and temperature walls might slow the race towards 100's and 1000's cores unless the processor industry takes a new paradigm shift from the current "replicating complex cores" (e.g. Intel Nehalem) towards many simple cores (e.g. Intel Larrabee) or heterogeneous manycores (e.g. new GPUs, IBM Cell).

3.3.2. The memory wall

For the past 20 years, the memory access time has been one of the main bottlenecks for performance in computer systems. This was already true for uniprocessors. Complex memory hierarchies have been defined and implemented in order to limit the visible memory access time as well as the memory traffic demands. Up to three cache levels are implemented for uniprocessors. For multi- and many-cores the problems are even worse. The memory hierarchy must be replicated for each core, memory bandwidth must be shared among the distinct cores, data coherency must be maintained. Maintaining cache coherency for up to 8 cores can be handled through relatively simple bus protocols. Unfortunately, these protocols do not scale for large numbers of cores, and there is no consensus on coherency mechanism for manycore systems. Moreover there is no consensus on core organization (flat ring? flat grid? hierarchical ring or grid?).

Therefore, organizing and dimensioning the memory hierarchy will be a major challenge for the computer architects. The successfull architecture will also be determined by the ability of the applications (i.e., the programmers or the compilers or the run-time) to efficiently place data in the memory hierarchy and achieve high performance.

Finally new technology opportunities may demand to revisit the memory hierarchy. As an example, 3D memory stacking enables a huge last-level cache (maybe several gigabytes) with huge bandwidth (several Kbits/ processor cycle). This dwarfs the main memory bandwidth and may lead to other architectural tradeoffs.

3.4. Need for efficient execution of parallel applications

Achieving high performance on future multicores will require the development of parallel applications, but also an efficient compiler/runtime toolchain to adapt codes to the execution platform.

3.4.1. The diversity of parallelisms

Many potential execution parallelism patterns may coexist in an application. For instance, one can express some parallelism with different tasks achieving different functionalities. Within a task, one can expose different granularities of parallelism; for instance a first layer message passing parallelism (processes executing the same functionality on different parts of the data set), then a shared memory thread level parallelism and fine grain loop parallelism (a.k.a vector parallelism).

Current multicores already feature hardware mechanisms to address these different parallelisms: physically distributed memory — e.g. the new Intel Nehalem already features 6 different memory channels — to address task parallelism, thread level parallelism — e.g. on conventional multicores, but also on GPUs or on Cell-based machines —, vector/SIMD parallelism — e.g. multimedia instructions. Moreover they also attack finer instruction level parallelism and memory latency issues. Compilers have to efficiently discover and manage all these forms to achieve effective performance.

3.4.2. Portability is the new challenge

Up to now, most parallel applications were developed for specific application domains in high end computing. They were used on a limited set of very expensive hardware platforms by a limited number of expert users. Moreover, they were executed in batch mode.

In contrast, the expectation of most end-users of the future mainstream parallel applications running on multicores will be very different. The mainstream applications will be used by thousands, maybe millions of non-expert users. These users consider functional portability of codes as granted. They will expect their codes to run faster on new platforms featuring more cores. They will not be able to tune the application environment to optimize performance. Finally, multiple parallel applications may have to be executed concurrently.

The variety of possible hardware platforms, the lack of expertise of the end-users and the varying run-time execution environments will represent major difficulties for applications in the multicore era.

First of all, the end user considers functional portability without recompilation as granted, this is a major challenge on parallel machines. Performance portability/scaling is even more challenging. It will become inconceivable to rewrite/retune each application for each new parallel hardware platform generation to exploit them. Therefore, apart from the initial development of parallel applications, the major challenge for the next decade will be to *efficiently* run parallel applications on hardware architectures radically different from their original hardware target.

3.4.3. The need for performance on sequential code sections

3.4.3.1. Most software will exhibit substantial sequential code sections

For the foreseeable future, the majority of applications will feature important sequential code sections.

First, many legacy codes were developed for uniprocessors. Most of these codes will not be completely redeveloped as parallel applications, but will evolve to applications using parallel sections for the most compute-intensive parts. Second, the overwhelming majority of the programmers have been educated to program in a sequential programming style. Parallel programming is much more difficult, time consuming and error prone than sequential programming. Debugging and maintaining a parallel code is a major issue. Investing in the development of a parallel application will not be cost-effective for the vast majority of software developments. Therefore, sequential programming style will continue to be dominant in the foreseeable future. Most developers will rely on the compiler to parallelize their application and/or use some software components from parallel libraries.

3.4.3.2. Future parallel applications will require high performance sequential processing on 1000's cores chip

With the advent of universal parallel hardware in multicores, large diffusion parallel applications will have to run on a broad spectrum of parallel hardware platforms. They will be used by non-expert users who will not be able to tune the application environment to optimize performance. They will be executed concurrently with other processes which may be interactive.

The variety of possible hardware platforms, the lack of expertise of the end-user and the varying run-time execution environments are major difficulties for parallel applications. This tends to constrain the programming style and therefore reinforces the sequential structure of the control of the application.

Therefore, *most future parallel applications will rely on a single main thread or a few main threads in charge of distinct functionalities of the application. Each main thread will have a general sequential control and can initiate and control the parallel execution of parallel tasks.*

In 1967, Amdahl [43] pointed out that, if only a portion of an application is accelerated, the execution time cannot be reduced below the execution time of the residual part of the application. Unfortunately, even highly parallelized applications exhibit some residual sequential part. For parallel applications, this indicates that the effective performance of the future 1000's cores chip will significantly depend on their ability to be efficient on the execution of the control portions of the main thread as well as on the execution of sequential portions of the application.

3.4.3.3. The success of 1000's cores architecture will depend on single thread performance

While the current emphasis of computer architecture research is on the definition of scalable multi- many- core architectures for highly parallel applications, we believe that the success of the future 1000-core architecture will depend not only on their performance on parallel applications including sequential sections, but also on their performance on single thread workloads.

3.5. Performance evaluation/guarantee

Predicting/evaluating the performance of an application on a system without explicitly executing the application on the system is required for several usages. Two of these usages are central to the research of the ALF project-team: microarchitecture research (the system to be be evaluated does not exist) and Worst Case Execution Time estimation for real-time systems (the numbers of initial states or possible data inputs is too large).

When proposing a micro-architecture mechanism, its impact on the overall processor architecture has to be evaluated in order to assess its potential performance advantages. For microarchitecture research, this evaluation is generally done through the use of cycle-accurate simulation. Developing such simulators is quite complex and microarchitecture research was helped but also biased by some popular public domain research simulators (e.g. Simplescalar [44]). Such simulations are CPU consuming and simulations cannot be run on a complete application. Sampling representative slices of the application was proposed [7] and popularized by the Simpoint [52] framework.

Real-time systems need a different use of performance prediction; on hard real-time systems, timing constraints must be respected independently from the data inputs and from the initial execution conditions. For such a usage, the Worst Case Execution Time (WCET) of an application must be evaluated and then checked against the timing constraints. While safe and tight WCET estimation techniques and tools exist for reasonably simple embedded processors (e.g. techniques based on abstract interpretation such as [46]), accurate evaluation of the WCET of an algorithm on a complex uniprocessor system is a difficult problem. Accurately modelling data cache behavior [6] and complex superscalar pipelines are still research questions as illustrated by the presence of so-called *timing anomalies* in dynamically scheduled processors, resulting from complex interactions between processor elements (among others, interactions between caching and instruction scheduling) [50].

With the advance of multicores, evaluating / guaranteeing a computer system response time is becoming much more difficult. Interactions between processes occurs at different levels. The execution time on each core depends on the behavior of the other cores. Simulations of 1000's cores micro-architecture will be needed in order to evaluate future many-core proposals. While a few multiprocessor simulators are available for the community, these simulators cannot handle realistic 1000's cores micro-architecture. New techniques have to be invented to achieve such simulations. WCET estimations on multicore platforms will also necessitate radically new techniques, in particular, there are predictability issues on a multicore where many resources are shared; those resources include the memory hierarchy, but also the processor execution units and all the hardware resources if SMT is implemented [56].

3.6. General research directions

The overall performance of a 1000's core system will depend on many parameters including architecture, operating system, runtime environment, compiler technology and application development. In the ALF project, we will essentially focus on architecture, compiler/execution environment as well as performance predictability, and in particular WCET estimation. Moreover, architecture research, and to a smaller extent, compiler and WCET estimation researches rely on processor simulation. A significant part of the effort in ALF will be devoted to define new processor simulation techniques.

3.6.1. Microarchitecture research directions

The overall performance of a multicore system depends on many parameters including architecture, operating system, runtime environment, compiler technology and application development. Even the architecture dimension of a 1000's core system cannot be explored by a single research project. Many research groups are exploring the parallel dimension of the multicores essentially targeting issues such as coherency and scalability.

We have identified that high performance on single threads and sequential codes is one of the key issues for enabling overall high performance on a 1000's core system and we anticipate that the general architecture of such 1000's core chip will feature many simple cores and a few very complex cores.

Therefore our research in the ALF project will focus on refining the microarchitecture to achieve high performance on single process and/or sequential code sections within the general framework of such an heteregeneous architecture. This leads to two main research directions 1) enhancing the microarchitecture of high-end superscalar processors, 2) exploiting/modifying heterogeneous multicore architecture on a single process. The temperature wall is also a major technological/architectural issue for the design of future processor chips.

3.6.1.1. Enhancing complex core microarchitecture

Research on wide issue superscalar processors was merely stopped around 2002 due to limited performance returns and the power consumption wall.

When considering a heterogeneous architecture featuring hundreds of simple cores and a few complex cores, these two obstacles will partially vanish: 1) the complex cores will represent only a fraction of the chip and a fraction of its power consumption. 2) any performance gain on (critical) sequential threads will result in a performance gain of the whole system

On the complex core, the performance of a sequential code is limited by several factors. At first, on current architectures, it is limited by the peak performance of the processor. To push back this first limitation, we will explore new microarchitecture mechanisms to increase the potential peak performance of a complex core enabling larger instruction issue width. The processor performance is also limited by control dependencies. To push back this limitation, we will explore new branch prediction mechanisms as well as new directions for reducing branch misprediction penalties [18], [17]. As data dependencies may strongly limit performance, we will revisit data prediction. Processor performance is also often highly dependent on the presence or absence of data in a particular level of the memory hierarchy. For the ALF multicore, we will focus on sharing the access to the memory hierarchy in order to adapt the performance of the main thread to the performance of the other cores. All these topics should be studied with the new perspective of quasi unlimited silicon budget.

3.6.1.2. Exploiting heterogeneous multicores on single process

When executing a sequential section on the complex core, the simple cores will be free. Two main research directions to exploit thread level parallelism on a sequential thread have been initiated in late 90's within the context of simultaneous multithreading and early chip multiprocessor proposals: helper threads and speculative multithreading.

Helper threads were initially proposed to improve the performance of the main threads on simultaneous multithreaded architectures [45]. The main idea of helper threads is to execute codes that will accelerate the main thread without modifying its semantic.

In many cases, the compiler cannot determine if two code sections are independent due to some unresolved memory dependency. When no dependency occurs at execution time, the code sections can be executed in parallel. Thread-Level Speculation has been proposed to exploit coarse grain speculative parallelism. Several hardware-only proposals were presented [51], but the most promising solutions integrate hardware support for software thread-level speculation [54].

In the context of future manycores, thread-level speculation and helper threads should be revisited. Many simple cores will be available for executing helper threads or speculative thread execution during the execution of sequential programs or sequential code sections. The availability of these many cores is an opportunity as well as a challenge. For example, one can try to use the simple cores to execute many different helper threads that could not be implemented within a simultaneous multithreaded processor. For thread level speculation, the new challenge is the use of less powerful cores for speculative threads. Moreover the availability of many simple cores may lead to the use of helper threads and thread level speculation at the same time.

3.6.1.3. Temperature issues

Temperature is one of the constraints that have prevented the processor clock frequency to be increased in recent years. Besides techniques to decrease the power consumption, the temperature issue can be tackled with *dynamic thermal management* [13] through techniques such as clock gating or throttling and *activity migration* [53][10].

Dynamic thermal management (DTM) is now implemented on existing processors. For high performance, processors are dimensioned according to the average situation rather than to the worst case situation. Temperature sensors are used on the chip to trigger dynamic thermal management actions, for instance thermal throttling whenever necessary. On multicores, it is possible to migrate the activity from one core to another in order to limit temperature.

A possible way to increase sequential performance is to take advantage of the smaller gate delay that comes with miniaturization, which permits in theory to increase the clock frequency. However increasing the clock frequency generally requires to increase the instantaneous power density. This is why DTM and activity migration will be key techniques to deal with Amdahl's law in future many-core processors.

3.6.2. Processor simulation research

Architecture studies, and in particular microarchitecture studies, require extensive validations through detailed simulations. Cycle accurate simulators are needed to validate the microarchitectural mechanisms.

Within the ALF project, we can distinguish two major requirements on the simulation: 1) single process and sequential code simulations 2) parallel code sections simulations.

For simulating parallel code sections, a cycle-accurate microarchitecture simulator of a 1000-core architecture will be unacceptably slow. In [12], we showed that mixing analytical modeling of the global behavior of a processor with detailed simulation of a microarchitecture mechanism allows to evaluate this mechanism. Karkhanis and Smith [47] further developed a detailed analytical simulation model of a superscalar processor. Building on top of these preliminary researches, simulation methodology mixing analytical modeling of the simple cores with a more detailed simulation of the complex cores is appealing. The analytical model of the simple cores will aim at approximately modeling the impact of the simple core execution on the shared resources (e.g. data bandwidth, memory hierarchy) that are also used by the complex cores.

Other techniques such as regression modeling [48] can also be used for decreasing the time required to explore the large space of microarchitecture parameter values. We will explore these techniques in the context of many-core simulation.

In particular, research on temperature issues will require the definition and development of new simulation tools able to simulate several minutes or even hours of processor execution, which is necessary for modeling thermal effects faithfully.

3.6.3. Compiler research directions

3.6.3.1. General directions

Compilers are keystone solutions for any approach that deals with high performance on 100+ processors systems. But general-purpose compilers try to embrace so many domains and try to serve so many constraints that they frequently fail to achieve very high performance. They need to be deeply revisited. We identify four main compiler/software related issues that must be addressed in order to allow efficient use of multi- and many-cores: 1) programming 2) resource management 3) application deployment 4) portable performance. Addressing these challenges will require to revisit parallel programming and code generation extensively.

The past of parallel programming is scattered with hundreds of parallel languages. Most of these languages were designed to program homogeneous architectures and were targeting a small and well-trained community of HPC programmers. With the new diversity of parallel hardware platforms and the new community of non-expert developers, expressing parallelism is not sufficient anymore. Resource management, application deployment and portable performance are intermingled issues that require to be addressed holistically.

As many decisions should be taken according to the available hardware, resource management cannot be separated from parallel programming. Deploying applications on various systems without having to deal with thousands of hardware configurations (different numbers of cores, accelerators, ...) will become a major concern for software distribution. The grail of parallel computing is to be able to provide portable performance on a large set of parallel machines and varying execution contexts.

Recent techniques are showing promises. Iterative compilation techniques, exploiting the huge CPU cycle count now available, can be used to explore the optimization space at compile-time. Second, machine-learning techniques can be used to automatically improve compilers and code generation strategies. Speculation can be used to deal with necessary but missing information at compile-time. Finally, dynamic techniques can select or generate at run-time the most efficient code adapted to the execution context and available hardware resources.

Future compilers will benefit from past research, but they will also need to combine static and dynamic techniques. Moreover, domain specific approaches might be needed to ensure success. The ALF research effort will focus on these static and dynamic techniques to address the multicore application development challenges.

3.6.3.2. Portability of applications and performance through virtualization

The life cycle is much longer for applications than for hardware. Unfortunately the multicore era jeopardizes the old binary compatibility recipe. Binaries cannot automatically exploit additional computing cores or new accelerators available on the silicon. Moreover maintaining backward binary compatibility on future parallel architectures will rapidly become a nightmare, applications will not run at all unless some kind of dynamic binary translation is at work.

Processor virtualization addresses the problem of portability of functionalities. Applications are not compiled to the final native code but to a target independent format. This is the purpose of languages such as Java and .NET. Bytecode formats are often *a priori* perceived as inappropriate for performance intensive applications and for embedded systems. However, it was shown that compiling a C or C++ program to a bytecode format produces a code size similar to dense instruction sets [4]. Moreover, this bytecode representation can be compiled to native code with performance similar to static compilation [3]. Therefore processor virtualization for high performance, i.e., for languages like C or C++, provides significant advantages: 1) it simplifies software engineering with fewer tools to maintain and upgrade; 2) it allows better code readability and easier code maintenancesince it avoids code specialization for specific targets using compile time macros such as #ifdef ; 3) the *execution code* deployed on the system is the execution code that has been debugged and validated, as opposed to the same *source code* has been recompiled for another platform; 4) new architectures will come with their JIT compiler. The JIT will (should) automatically take advantage of new architecture features such as SIMD/vector instructions or extra processors.

Our objective is to enrich processor virtualization to allow both functional portability and high performance using JIT at runtime, or bytecode-to-native code offline compiler. Split compilation can be used to annotate the bytecode with relevant information that can be helpful to the JIT at runtime or to the bytecode to native code offline compiler. Because the first compilation pass occurs offline, aggressive analyses can be run and their outcomes encoded in the bytecode. For example, such informations include vectorizability, memory references (in)dependencies, suggestions derived from iterative compilation, polyhedral analysis, or integer linear programming. Virtualization allows to postpone some optimizations to run time, either because they increase the code size and would increase the cost of an embedded system or because the actual hardware platform characteristics are unknown.

3.6.4. Performance predictability for real-time systems

While compiler and architecture research efforts often focus on maximizing average case performance, applications with real-time constraints do not need onlyhigh performance but also performance guarantees in all situations, including the worst-case situation. Worst-Case Execution Time estimates (WCET) need to be upper bounds of any possible execution time. The safety level required depends on the criticality of applications: missing a frame on a video in the airplane for passenger in seat 20B is less critical than a safety critical decision in the control of the airplane.

Within the ALF project, our objective is to study performance guarantees for both (i) sequential codes running on complex cores ; (ii) parallel codes running on the multicores. Considering the ALF base architecture, this results in two quite distinct problems.

For sequential code executing on a single core, one can expect that, in order to provide real-time possibility, the architecture will feature an execution mode where a given processor will be guaranteed to access a fixed portion of the shared resources (caches, memory bandwidth). Moreover, this guaranteed share could be optimized at compile time to enforce the respect of the time constraints. However, estimating the WCET of an application on a complex micro-architecture is still a research challenge. This is due to the complex interaction of micro-architectural elements (superscalar pipelines, caches, branch prediction, out-of-order execution) [50]. We will continue to explore pure analytical and static methods. However when accurate static hardware modeling methods cannot handle the hardware complexity, new probabilistic methods [49] might be needed to explore to obtain as safe as possible WCET estimates.

Providing performance guarantees for parallel applications executed on a multicore is a new and challenging issue. Entirely new WCET estimation methods have to be defined for these architectures to cope with dynamic resource sharing between cores, in particular on-chip memory (either local memory or caches) are shared, but also buses, network-on-chip and the access to the main memory. Current pure analytical methods are too pessimistic at capturing interferences between cores [58], therefore hardware-based or compiler methods such as [55] have to be defined to provide some degree of isolation between cores. Finally, similarly to simulation methods, new techniques to reduce the complexity of WCET estimation will be explored to cope with manycore architectures.

4. Application Domains

4.1. Application Domains

The ALF team is working on the fundamental technologies for computer science: processor architecture and performance-oriented compilation. The research results have impacts on any application domain that requires high performance executions (telecommunication, multimedia, biology, health, engineering, environment ...), but also on many embedded applications that exhibit other constraints such as power consumption, code size and guaranteed response time. Our research activity implies the development of software prototypes.

5. Software

5.1. Panorama

The ALF team is developing several software prototypes for research purposes: compilers, architectural simulators, programming environments, ...

Among the many prototypes developed in the project, we describe here **ATMI**, a microarchitecture temperature model for processor simulation, **STiMuL**, a temperature model for steady state studies, **ATC**, an address trace compressor, **HAVEGE**, an unpredictable random number generator and **tiptop**, a user-level Linux utility that collects data from hardware performance counters for running tasks, software developed by the team.

5.2. ATMI

Participant: Pierre Michaud.

Contact : Pierre Michaud

Status : Registered with APP Number IDDN.FR.001.250021.000.S.P.2006.000.10600, Available under GNU General Public License

Research on temperature-aware computer architecture requires a chip temperature model. General purpose models based on classical numerical methods like finite differences or finite elements are not appropriate for such research, because they are generally too slow for modeling the time-varying thermal behavior of a processing chip.

We have developed an ad hoc temperature model, ATMI (Analytical model of Temperature in MIcroprocessors), for studying thermal behaviors over a time scale ranging from microseconds to several minutes. ATMI is based on an explicit solution to the heat equation and on the principle of superposition. ATMI can model any power density map that can be described as a superposition of rectangle sources, which is appropriate for modeling the microarchitectural units of a microprocessor.

Visit http://www.irisa.fr/alf/ATMI or contact Pierre Michaud.

5.3. STiMuL

Participant: Pierre Michaud.

Status: Registered with APP Number IDDN.FR.001.220013.000.S.P.2010.000.31235, Available under GNU General Public License

Some recent research has started investigating the microarchitectural implications of 3D circuits, for which the thermal constraint is stronger than for conventional 2D circuits.

STiMuL can be used to model steady-state temperature in 3D circuits consisting of several layers of different materials. STiMuL is based on a rigorous solution to the Laplace equation [9]. The number and characteristics of layers can be defined by the user. The boundary conditions can also be defined by the user. In particular, STiMuL can be used along with thermal imaging to obtain the power density inside an integrated circuit. This power density could be used for instance in a dynamic simulation oriented temperature modeling such as ATMI.

STiMuL is written in C and uses the FFTW library for discrete Fourier transforms computations.

Visit http://www.irisa.fr/alf/stimul or contact Pierre Michaud.

5.4. ATC

Participant: Pierre Michaud.

Contact : Pierre Michaud

Status: registered with APP number IDDN.FR.001.160031.000.S.P.2009.000.10800, available under GNU LGPL License.

Trace-driven simulation is an important tool in the computer architect's toolbox. However, one drawback of trace-driven simulation is the large amount of storage that may be necessary to store traces. Trace compression techniques are useful for decreasing the storage space requirement. But general-purpose compression techniques are generally not optimal for compressing traces because they do not take advantage of certain characteristics of traces. By specializing the compression method and taking advantages of known trace characterics, it is possible to obtain a better tradeoff between the compression ratio, the memory consumption and the compression and decompression speed.

ATC is a utility and a C library for compressing/decompressing address traces. It implements a new lossless transformation, Bytesort, that exploits spatial locality in address traces. ATC leverages existing general-purpose compressors such as gzip and bzip2. ATC also provides a lossy compression mode that yields higher compression ratios while preserving certain important characteristics of the original trace.

Visit http://www.irisa.fr/alf/atc or contact Pierre Michaud.

5.5. HAVEGE

Participant: André Seznec.

Contact : André Seznec

Status : Registered with APP Number IDDN.FR.001.500017.001.S.P.2001.000.10000. Available under the LGPL license.

An unpredictable random number generator is a practical approximation of a truly random number generator. Such unpredictable random number generators are needed for cryptography. HAVEGE (HArdware Volatile Entropy Gathering and Expansion) is a user-level software unpredictable random number generator for general-purpose computers that exploits the continuous modifications of the internal volatile hardware states in the processor as a source of uncertainty [16]. HAVEGE combines on-the-fly hardware volatile entropy gathering with pseudo-random number generation.

The internal state of HAVEGE includes thousands of internal volatile hardware states and is merely unmonitorable. HAVEGE can reach an unprecedented throughput for a software unpredictable random number generator: several hundreds of megabits per second on current workstations and PCs.

The throughput of HAVEGE favorably competes with usual pseudo-random number generators such as `rand()` or `random()`. While HAVEGE was initially designed for cryptology-like applications, this high throughput makes HAVEGE usable for all application domains demanding high performance and high quality random number generators, e.g., Monte Carlo simulations.

Visit http://www.irisa.fr/alf/HAVEGE or contact André Seznec.

5.6. Tiptop

Participant: Erven Rohou.

Status: Registered with APP (Agence de Protection des Programmes). Available under GNU General Public License v2.

Tiptop is a new simple and flexible user-level tool that collects hardware counter data on Linux platforms (version 2.6.31+). The goal is to make the collection of performance and bottleneck data as simple as possible, including simple installation and usage. In particular, we stress the following points.

- Installation is only a matter of compiling the source code. No patching of the Linux kernel is needed, and no special-purpose module needs to be loaded.
- No privilege is required, any user can run *tiptop* — non-privileged users can only watch processes they own, ability to monitor anybody's process opens the door to side-channel attacks.
- The usage is similar to *top*. There is no need for the source code of the applications of interest, making it possible to monitor proprietary applications or libraries. And since there is no probe to insert in the application, understanding of the structure and implementation of complex algorithms and code bases is not required.
- Applications do not need to be restarted, and monitoring can start at any time (obviously, only events that occur after the start of *tiptop* are observed).
- Events can be counted per thread, or per process.

Tiptop is written in C. It can take advantage of libncurses when available for pseudo-graphic display.

For more information, please contact Erven Rohou.

6. New Results

6.1. Processor Architecture

Participants: Damien Hardy, Pierre Michaud, Nathanaël Prémillieu, Ricardo Andrés Velasquéz, Luis-Germán García Morales, Bharath Narasimha Swamy, André Seznec.

Our research in computer architecture covers memory hierarchy, branch prediction, superscalar implementation, as well as SMT and multicore issues.

This year, we have also initiated new research directions within the context of the ERC DAL project.

6.1.1. Null block management on the memory hierarchy

Participant: André Seznec.

It has been observed that some applications manipulate large amounts of null data. Moreover these zero data often exhibit high spatial locality. On some applications more than 20% of the data accesses concern null data blocks. To reduce the pressure on main memory, we have proposed a hardware compressed memory that only targets null data blocks, the decoupled zero-compressed memory [27]. Borrowing some ideas from the decoupled sectored cache [20], the decoupled zero-compressed memory, or DZC memory, manages the main memory as a decoupled sectored set-associative cache where null blocks are only represented by a validity bit. Our experiments show that for many applications, the DZC memory allows to artificially enlarge the main memory, i.e. it reduces the effective physical memory size needed to accommodate the working set of an application without excessive page swapping. Moreover, the DZC memory can be associated with a ZCA cache [5] to manage null blocks across the whole memory hierarchy. For some applications, such a management significantly decreases the memory traffic and therefore can significantly improve performance.

This work corresponds to the PhD of Julien Dusser defended in december 2010.

6.1.2. Emerging memory technologies

Participant: André Seznec.

Phase change memory (PCM) technology appears more scalable than DRAM technology. As PCM exhibits access time slightly longer but in the same range as DRAMs, several recent studies have proposed to use PCMs for designing main memory systems. Unfortunately PCM technology suffers from a limited write endurance; typically each memory cell can only be written a large but still limited number of times (10 millions to 1 billion writes are reported for current technology). Research proposals have essentially focused their attention on designing memory systems that will survive the average behavior of conventional applications. However PCM memory systems should be designed to survive worst-case applications, i.e., malicious attacks targeting the physical destruction of the memory through overwriting a limited number of memory cells.

In 2010, we have proposed the first design of a secure PCM-based main memory that would by construction survive overwrite attacks [19]. This secure PCM-based main memory requires a significant read and write extra memory traffic (an extra memory write per 8 demand memory writes) on all applications. Concurrent proposals require even higher extra read and write memory traffic. In collaboration with a research group from IBM, we have proposed a hardware method to detect malicious overwrite attacks on the main memory, thus limiting the memory traffic overhead on non-malicious applications [32].

6.1.3. Microarchitecture exploration of control flow reconvergence

Participants: Nathanaël Prémillieu, André Seznec.

After continuous progress over the past 15 years [18], [17], the accuracy of branch predictors seems to be reaching a plateau. Other techniques to limit control dependency impact are needed. Control flow reconvergence is an interesting property of programs. After a multi-option control-flow instruction (i.e. either a conditional branch or an indirect jump including returns), all the possible paths merge at a given program point: the reconvergence point.

Superscalar processors rely on aggressive branch prediction, out-of-order execution and instruction level parallelism for achieving high performance. Therefore, on a superscalar core , the overall speculative execution after the mispredicted branch is cancelled, leading to a substantial waste of potential performance. However, deep pipelines and out-of-order execution induce that, when a branch misprediction is resolved, instructions following the reconvergence point have already been fetched, decoded and sometimes executed. While some of this executed work has to be cancelled since data dependencies exist, cancelling the control independent work is a waste of resources and performance. We have proposed a new hardware mechanism called SYRANT, SYmmetric Resource Allocation on Not-taken and Taken paths, addressing control flow reconvergence at a reasonable cost. Moreover, as a side contribution of this research we have shown that, for a modest hardware cost, the outcomes of the branches executed on the wrong paths can be used to guide branch prediction on the correct path.

6.1.4. Confidence estimation for the TAGE predictor

Participant: André Seznec.

For the past 15 years, it has been shown that confidence estimation of branch prediction (i.e., estimating the probability of correct or incorrect prediction) can be used for various usages such as fetch gating or throttling for power saving or for controlling resource allocation policies in an SMT processor. In many proposals, using extra hardware and particularly storage tables for branch confidence estimators has been considered as a worthwhile silicon investment.

The TAGE predictor, presented in 2006 [18], is so far considered as the state-of-the-art conditional branch predictor. We have shown that very accurate confidence estimations can be done for the branch predictions realized by the TAGE predictor by simply observing the outputs of the predictor tables. Many confidence estimators proposed in the literature only discriminate between high confidence predictions and low confidence estimations. It has been recently pointed out that a more selective confidence discrimination could be useful. The observation of the outputs of the predictor tables is sufficient to grade the confidence in the branch predictions with a very good granularity. Moreover a slight modification of the predictor automaton allows to discriminate the prediction in three classes, low-confidence (with a misprediction rate in the 30 % range), medium confidence (with a misprediction rate in 8-12% range) and high confidence (with a misprediction rate lower than 1 %) [37].

6.1.5. Improving branch prediction accuracy

Participant: André Seznec.

The TAGE predictor [18] is often considered as state-of-the-art in conditional branch predictors proposed by academy. For the 3rd championship branch prediction, we have further improved its accuracy by augmenting it with small side predictors (Loop predictor, Statiscal Corrector Predictor, Immediate Update Mimicker) [34]. This predictor won the conditional branch track of the 3rd championship branch prediction. In order to further argue for real hardware implementation of the TAGE predictor, we have presented several propositions to reduce the complexity of its hardware design, to reduce its energy consumption [36] and further improve branch accuracy. On a hardware implementation of a conditional branch predictor, the predictor tables are updated at retire time. A retired branch normally induces three accesses to the branch predictor tables : read at prediction time, read at retire time and write for the update. We show that in practice, the TAGE predictor accuracy would not be significantly impaired by avoiding a systematic second read of the prediction tables at retire time for correct prediction. Combined with the elimination of silent updates, this significantly reduces the number of accesses to the predictor. Furthermore, we present a technique allowing to implement the TAGE predictor tables as bank-interleaved structures using single-port memory components. This significantly reduces the silicon footprint of the predictor as well as its energy consumption without significantly impairing its accuracy.

Correctly predicting the indirect branches has become critical with the introduction of object oriented programming, java programming as well as with the renewed importance of interpreters. The ITTAGE indirect branch predictor was introduced in [15]. Threes versions of the ITTAGE predictor were presented at the indirect branch track at the championship branch prediction by three different teams, and secured the three first places. Our proposition [35] won the championship.

6.1.6. Hardware acceleration of sequential loops

Participant: Pierre Michaud.

In a decade it will be possible to put on a single chip several hundreds of superscalar cores. A simple application of Amdahl's law shows that it will make sense to dedicate to sequential performance the silicon area and power budget corresponding to that of several tens, or perhaps several hundreds of conventional superscalar cores. This will lead to a *sequential accelerator* which will be used to accelerate sequential programs and sequential code sections in parallel programs. The question is, what will this sequential accelerator look like ? In a previous work, we have proposed a possible solution for implementing a sequential accelerator, which is to implement a superscalar core with a very "aggressive" microarchitecture and design, and to replicate this core and migrate the execution periodically on the replicas to keep the temperature resulting from the high power density under control [11]. However, future sequential accelerators will probably rely on a combination of several techniques, some already known, some yet to be invented.

We have started exploring a new solution for sequential acceleration, the hardware acceleration of dynamic loops, which are periodic sequences of dynamic instructions. A *loop accelerator* sits beside a conventional superscalar core and is specialized in executing dynamic loops [40]. Dynamic loops are detected and accelerated automatically, without help from the programmer or the compiler. The execution is migrated from the superscalar core to the loop accelerator when a dynamic loop is detected, and back to the superscalar core when a loop exit condition is encountered. Our simulations show that about one third of all the instructions executed by the SPEC CPU2006 benchmark suite belong to dynamic loops with a length of several thousands dynamic instructions, or more. The loop body size is quite diverse, ranging from a few instructions to several hundreds.

We have described a possible loop accelerator microarchitecture that exploits loop properties and avoids the main bottlenecks of conventional superscalar microarchitectures. Our preliminary study demonstrates significant global speedup on some benchmarks, with a local acceleration for loops typically around 2. Our future research on loop acceleration will explore the solution space for obtaining greater performance speedups.

6.1.7. Exploiting confidence in SMT processors

Participants: Pierre Michaud, André Seznec.

Simultaneous multithreading (SMT) [57] processors dynamically share processor resources between multiple threads. The hardware allocates resources to different threads. The resources are either managed explicitly through setting resource limits to each thread or implicitly through placing the desired instruction mix in the resources. In this case, the main resource management tool is the instruction fetch policy which must predict the behavior of each thread (branch mispredictions, long-latency loads, etc.) as it fetches instructions.

We propose the use of Speculative Instruction Window Weighting (SIWW) [25] to bridge the gap between implicit and explicit SMT fetch policies. SIWW estimates for each thread the amount of outstanding work in the processor pipeline. Fetch proceeds for the thread with the least amount of work left. SIWW policies are implicit as fetch proceeds for the thread with the least amount of work left. They are also explicit as maximum resource allocation can also be set. SIWW can use and combine virtually any of the indicators that were previously proposed for guiding the instruction fetch policy (number of in-flight instructions, number of low confidence branches, number of predicted cache misses, etc.). Therefore, SIWW is an *approach to design SMT fetch policies*, rather than a particular fetch policy.

Targeting fairness and throughput is often contradictory and a SMT scheduling policy often optimizes only one performance metric with the sacrifice of the other metric. Our simulations show that the SIWW fetch policy can achieve at the same time state-of-the-art throughput, state-of-the-art fairness and state-of-the-art harmonic performance mean.

As a side contribution of this study, we have published a study on fairness metrics for SMT processors and multicores [24].

This study was done in collaboration with Hans Vandierendonck from University of Ghent.

6.1.8. Analytical model to estimate the interaction between hardware faults on caches and predictors

Participant: Damien Hardy.

This research was undertaken during Damien Hardy's stay in the Computer Architecture group of the University of Cyprus (June-August 2011).

Technology trends suggest that in tomorrow's computing systems, failures will become a commonplace due to many factors, and the expected probability of failure will increase with scaling. Faults can result in execution errors (e.g. on caches) or simply in performance loss (e.g. predictors). Although faults can occur anywhere in the processor, the performance implications of a faulty cell vary depending on how the array is used in a processor.

A direction to determine the impact on performance due to permanent faulty cells is to predict the performance vulnerability by using analytical models. Such models, studied at the University of Cyprus, are representative for the average performance and its probability distribution. So far, analytical models have been defined to determine the impact on performance of faulty mechanisms such as caches and predictors in isolation without any interactions between them.

On the other side, in the real-time systems community, caches and predictors have been intensively studied to estimate the worst-case execution time of application by using static analysis. The ongoing research aims at defining an analytical model of performance that captures the effects of faults on both caches and predictors. This analytical model will be useful to predict future processors performance vulnerability to faults and to determine the benefits in terms of performance of reliability mechanisms.

6.1.9. Hardware support for transactional memory

Participants: Mridha-Mohammad Waliullah, André Seznec.

Parallel programming has become immensely important to harness the power of today's many core CPU. Over several years, a lot of efforts has been laid out to make parallel programming easier. Transactional memory (TM) has come out as an infrastructure that promises to simplify parallel programming. Implementation of TM in hardware is carried out to get higher performance, which is referred to as hardware transactional memory (HTM). We have focused mainly into two issues related to HTM: (1) exploring TM benchmarks to better understand the performance bottlenecks and (2) exploring innovative techniques that can streamline common case transitional execution to achieve higher performance [38]

This work was done in the framework of the Ercim postdoc stay (01/04/11 to 30/11/11) of Mridha-Mohammad Waliullah.

6.1.10. Microarchitecture research initiated in the DAL project

Participants: Pierre Michaud, Luis-Germán García Morales, Bharath Narasimha Swamy, André Seznec.

Multicore processors have now become mainstream for both general-purpose and embedded computing. Instead of working on improving the architecture of the next generation multicore, with the DAL project, we deliberately anticipate the next few generations of multicores. While multicores featuring 1000s of cores might become feasible around 2020, there are strong indications that sequential programming style will continue to be dominant. Even future mainstream parallel applications will exhibit large sequential sections. Amdahl's law indicates that high performance on these sequential sections is needed to enable overall high performance on the whole application. On many (most) applications, the effective performance of future computer systems using a 1000-core processor chip will significantly depend on their performance on both sequential code sections and single threads.

We envision that, around 2020, the processor chips will feature a few complex cores and many (may be 1000's) simpler, more silicon and power effective cores.

In the DAL research project, we will explore the microarchitecture techniques that will be needed to enable high performance on such heterogeneous processor chips. Very high performance will be required on both sequential sections, -legacy sequential codes, sequential sections of parallel applications-, and critical threads on parallel applications, -e.g. the main thread controlling the application. Our research will focus on enhancing single processes performance.

On the microarchitecture side, we will explore both a radically new approach, the sequential accelerator [11], and more conventional processor architectures. We will also study how to exploit heterogeneous multicore architectures to enhance sequential thread performance. Two PhD thesis have been initiated on these topics at fall 2011.

6.2. Compiler, vectorization, interpretation

Participants: Erven Rohou, David Yuste, André Seznec.

The usage of the bytecote-based languages such as Java has been generalized in the past few years. Applications are now very large and are deployed on many different platforms, since they are highly portable. With the new diversity of multicore platforms, functional, but also performance portability will become the major issue in the next 10 years. Therefore our research effort focuses on efficiently compiling towards bytecodes and on efficiently executing the bytecodes through JIT compilation or through direct interpretations.

6.2.1. Iterative and JIT compilation

Participants: Erven Rohou, David Yuste.

Over the last decade, iterative compilation has been an attempt to overcome the difficulty to generate extremely optimized code by letting the compilers explore many alternatives to select the best one. In this research, we extend previous work in the direction of portability. Future processors will be increasingly diverse and heterogenous, and portability is likely to be attained thanks to a bytecode format and JIT compilers. We explore how iterative compilation performed offline can generate useful information to allow the online JIT compiler to generate efficient code at very limited cost.

Part of this research is done in collaboration with STMicroelectronics, in the context of the Nano2012 Mediacom project.

6.2.2. Split vectorization

Participants: Erven Rohou, David Yuste, André Seznec.

We attempt to reconcile two apparently contradictory trends of computing systems. On the one hand, hardware heterogeneity favors the adoption of bytecode format and late, just-in-time code generation. On the other hand, exploitation of hardware features, in particular SIMD extensions through vectorization, is key to obtaining the required performance.

We showed in [33] that speculatively vectorized bytecode is: (1) robust — the approach is general enough to allow execution, both when using SIMD capabilities and also in the absence of SIMD extensions, or when using an unmodified, non-vectorizing JIT compiler; (2) risk-free — the penalty of running vectorized bytecode without SIMD support is kept at a minimum; (3) efficient — the improvement of running vectorized bytecode with SIMD support is maximized.

In [31], we focused on providing an infrastructure capable of supporting diverse SIMD targets (SSE, AltiVec, NEON), across a wide range of vectorizable kernels, with performance comparable to monolithic compiler vectorization.

This research is done within the framework of the HIPEAC2 network in collaboration with Albert Cohen (INRIA Alchemy), Ayal Zaks and Dorit Nuzman (IBM Research Labs, Haifa, Israel).

6.2.3. Vectorization Technology To Improve Interpreter Performance

Participants: Erven Rohou, David Yuste.

Recent trends in consumer electronics have created a new category of portable, lightweight software applications. Typically, these applications have fast development cycles and short life spans. They run on a wide range of systems and are deployed in a target independent bytecode format over Internet and cellular networks. Their authors are untrusted third-party vendors, and they are executed in secure managed runtimes or virtual machines. Furthermore, due to security policies, these virtual machines are often lacking just-in-time compilers and are reliant on interpreter execution.

The main performance penalty in interpreters arises from instruction dispatch. Each bytecode requires a minimum number of machine instructions to be executed. In this work we introduce a powerful and portable representation that reduces instruction dispatch thanks to vectorization technology. It takes advantage of the vast research in vectorization and its presence in modern compilers. Thanks to a split compilation strategy, our approach exhibits almost no overhead. Complex compiler analyses are performed ahead of time. Their results are encoded on top of the bytecode language, becoming new SIMD IR (i.e., intermediate representation) instructions. The bytecode language remains unmodified, thus this representation is compatible with legacy interpreters.

This approach drastically reduces the number of instructions to interpret and improves execution time. SIMD IR instructions are mapped to hardware SIMD instructions when available, with a substantial improvement.

6.2.4. Tiptop

Participant: Erven Rohou.

Hardware performance monitoring counters have recently received a lot of attention. They have been used by diverse communities to understand and improve the quality of computing systems: for example, architects use them to extract application characteristics and propose new hardware mechanisms; compiler writers study how generated code behaves on particular hardware; software developers identify critical regions of their applications and evaluate design choices to select the best performing implementation.

We propose [41] that counters be used by all categories of users, in particular non-experts, and we advocate that a few simple metrics derived from these counters are relevant and useful. For example, a low IPC (number of executed instructions per cycle) indicates that the hardware is not performing at its best; a high cache miss ratio can suggest several causes, such as conflicts between processes in a multicore environment.

We propose tiptop: a new tool, similar to the UNIX top utility, that requires no special privilege and no modification of applications. Tiptop provides more informative estimates of the actual performance than existing UNIX utilities, and better ease of use than current tools based on performance monitoring counters. With several use cases, we have illustrated possible usages of such a tool.

6.3. Understanding performance issues

Participants: Junjie Lai, Ricardo Andrés Velasquéz, Pierre Michaud, André Seznec.

6.3.1. Behavioral application-dependent superscalar core modeling

Participants: Ricardo Andrés Velasquéz, Pierre Michaud, André Seznec.

In recent years, research in microarchitecture has shifted from single-core to multi-core processors. Cycle-accurate models for many-core processors featuring hundreds or even thousands of cores are out of reach for realistic workloads. Approximate simulation methodologies which trade accuracy for simulation speed are necessary for conducting certain research, in particular for studying the impact of resource sharing between cores, where the shared resource can be caches, on-chip network, memory bus, power, temperature, etc.

Behavioral superscalar core modeling is a possible way to trade accuracy for simulation speed in situations where the focus of the study is not the core itself but what is outside the core, i.e., the *uncore*. In this modeling approach, a superscalar core is viewed as a black box emitting requests to the uncore at certain times. A behavioral core model can be connected to a cycle-accurate uncore model. Behavioral core models are built from detailed simulations. Once the time to build the model is amortized, significant simulation speedups are achieved.

We have proposed a new method for defining behavioral models for modern superscalar cores. Our method, *behavioral application-dependent superscalar core* (**BADCO**) modeling, requires two traces generated with cycle-accurate simulations. For the first trace, all the requests from the core (which includes the level-1 caches) to the uncore are forced with a null latency, i.e., we simulate a perfect uncore. For the second trace, all the requests are forced with a fixed and very long latency. Then we build a BADCO model from the timing information recorded in these two traces. A BADCO model is basically a directed graph where each node represents a group of micro-ops that may carry some requests to the uncore. Edges in this graph represent dependencies between requests. After the model is built, it can be used for simulations. During simulation, the BADCO model emulates the processor's reorder buffer, the level-1 miss status holding registers, and honors dependencies between nodes. We have compared BADCO with Lee et al.'s PDCM behavioral core model. BADCO is more accurate than PDCM on average and is more reliable [42]. BADCO predicts the execution time of a thread running on a modern superscalar core with an error typically under 5%. From our experiments, we found that BADCO is qualitatively accurate, being able to predict how performance changes when we change the uncore. The simulation speedups obtained with BADCO are typically greater than 10.

6.3.2. Architecture for Lattice QCD

Participants: Junjie Lai, André Seznec.

Simulation of Lattice QCD is a challenging computational problem that requires very high performance exceeding sustained Petaflops/s. In the framework of the ANR Cosinus PetaQCD project, we are modeling the demands of this application on the memory system and synchronization mechanisms.

In particular, GPUs have become popular to execute computing intensive scientific applications. In [39], we have introduced a GPU timing model to provide more insights into the applications' performance on GPU. A GPU CUDA program timing estimation tool (TEG) is developed based on the GPU timing model. Especially, TEG illustrates how performance scales from one warp (CUDA thread group) to multiple concurrent warps on SM (Streaming Multiprocessor). Because TEG takes the native GPU assembly code as input, it allows to estimate the execution time with only a small error. TEG can help programmers to better understand the performance results. It also allows to identify and quantify performance bottlenecks.

6.4. WCET estimation

Participants: Damien Hardy, Benjamin Lesage, Isabelle Puaut, Erven Rohou, André Seznec.

Predicting the amount of resources required by embedded software is of prime importance for verifying that the system will fulfill its real-time and resource constraints. A particularly important point in hard real-time embedded systems is to predict the Worst-Case Execution Times (WCETs) of tasks, so that it can be proven that tasks temporal constraints (typically, deadlines) will be met. Our research concerns methods for obtaining automatically upper bounds of the execution times of applications on a given hardware. Our focus this year is on (i) multicore architectures, (ii) applications compiled using just-in-time (JIT) compilation, and (iii) definition of both predictable and efficient hardware.

6.4.1. Timing analysis for multicore platforms with shared caches

Participants: Benjamin Lesage, Damien Hardy, Isabelle Puaut.

WCET estimation for multicore platforms is challenging task because of the possible interferences between cores due to shared hardware resources such as shared caches, memory bus, etc. Moreover, multi-core platforms use a hierarchy of caches, whose worst-case behavior has to be predicted safely and as tightly as possible.

6.4.1.1. Scalable fixed-point free instruction cache analysis

Estimating worst-case execution times (WCETs) for architectures with caches requires the worst-case number of cache misses to be upper bounded. Most existing static cache analysis methods, a large majority of those applying to unicores and a single cache level, use fixed-point computation and do not scale well with large code sizes.

Estimating WCETs for multicores requires the base cache analysis used to analyze every cache level in the memory hierarchy to be fast. To address this issue, we have proposed in [28] a new fast and scalable instruction cache analysis technique. In contrast to existing work, neither fixed-point computation nor heavyweight interprocedural analysis are required. Thus, code sizes that are too long to analyze with existing techniques becomes then analyzable with lower analysis time and memory consumption, and with only a slight degradation of the analysis precision. Experimental results show a reduction of the analysis execution time of a factor 5 in average (with a peak near 30 for the largest and most complex code) with a limited waste of tightness of the analysis.

6.4.1.2. Static analysis of cache hierarchies

Participants: Benjamin Lesage, Damien Hardy, Isabelle Puaut.

Determining the worst-case number of cache misses in multicore architectures requires the definition of analysis techniques applicable to cache hierarchies. In our previous work [6] we have defined the first technique that safely analyzes such hierarchies, first considering non-inclusive caches and LRU cache replacement.

We have recently generalized our previous work to support different cache hierarchy management policies between cache levels: non-inclusive, inclusive and exclusive cache hierarchies. Moreover, our analysis now supports cache hierarchies with different replacement policies: Least Recently Used (LRU), Pseudo-LRU, First-In First-Out (FIFO), Most Recently Used (MRU) and Random. Experimental results, detailed in [21] show that the method is precise in many cases (non-inclusive and exclusive cache hierarchies with the LRU replacement policy) and has a reasonable computation time. Nevertheless, considering inclusion enforcement mechanisms and non-LRU replacement policies leads to an increase of the analysis pessimism. Moreover, these two sources of pessimism are cumulative, thus resulting, in some cases, in a significant overestimation. Although inclusive cache hierarchies with non-LRU replacement policies can be analyzed statically, the cache hierarchies to be favored to obtain the tighter WCET estimates are hierarchies of non-inclusive or exclusive caches with the LRU replacement policy.

6.4.1.3. Reconciling Predictability and Just-In-Time Compilation

Participants: Isabelle Puaut, Erven Rohou.

Virtualization and just-in-time (JIT) compilation have become important toolso address application portability issues without deteriorating average-case performance. Unfortunately, JIT compilation raises predictability issues, which currently hinders its dissemination in real-time applications. Our work aims at reconciling the two domains, i.e. taking advantage of the portability and performance provided by JIT compilation, while providing predictability guarantees.

As a first step towards this ambitious goal, we have proposed two structures of code caches and have demonstrated their predictability [26]. On the one hand, the binary code caches we propose avoid too frequent function recompilations, providing good average-case performance. On the other hand, and more importantly for the system determinism, we show that the behavior of the code cache is predictable: a safe upper bound of the number of function recompilations can be computed, enabling the verification of timing constraints. Experimental results show that fixing function addresses in the binary cache ahead of time results in tighter Worst Case Execution Times (WCETs) than organizing the binary code cache in fixed-size blocks replaced using a Least Recently Used (LRU) policy.

6.4.1.4. Predictable shared caches for mixed-criticality real-time systems

Participants: Benjamin Lesage, Isabelle Puaut, André Seznec.

The general adoption of multi-core architectures has raised new opportunities as well as new issues in all application domains. In the context of real-time applications, it has created one major opportunity and one major difficulty. On the one hand, the availability of multiple high performance cores has created the opportunity to mix on the same hardware platform the execution of a complex critical real-time workload and the execution of non-critical applications. On the other hand, for real-time tasks timing deadlines must be met and enforced. Hardware resource sharing inherent to multicores hinders the timing analysis of concurrent tasks. Two different objectives are then pursued: enforcing timing deadlines for real-time tasks and achieving highest possible performance for the non-critical workload.

In this work, we suggest a hybrid hardware-based cache partitioning scheme that aims at achieving these two objectives at the same time. Plainly considering inter-task conflicts on shared cache for real-time tasks yields very pessimistic timing estimates. We remove this pessimism by reserving private cache space for real-time tasks. Upon the creation of a real-time task, our scheme reserves a fixed number of cache lines per set for the task. Therefore uniprocessor worst case execution time (WCET) estimation techniques can be used, resulting in tight WCET estimates. Upon the termination of the real-time task, this private cache space is released and made available for all the executed threads including non-critical ones. That is, apart the private spaces reserved for the real-time tasks currently running, the cache space is shared by all tasks running on the processor, i.e. non-critical tasks but also the real-time tasks for their least recently used blocks. Experiments show that the proposed cache scheme allows to both guarantee the schedulability of a set of real-time tasks with tight timing constraints and enable high performance on the non-critical tasks.

6.4.2. Preemption delay analysis for floating non-preemptive region scheduling

Participant: Isabelle Puaut.

This is joint work with Stefan M. Petters, Vincent Nélis and José Marinho, ISEP Porto, Portugal.

In real-time systems, there are two distinct trends for scheduling task sets on unicore systems: non-preemptive and preemptive scheduling. Non-preemptive scheduling is obviously not subject to any preemption delays but its schedulability may be quite poor, whereas fully preemptive scheduling is subject to preemption delays, but benefits from a higher flexibility in the scheduling decisions.

The time-delay involved by task preemptions is a major source of pessimism in the analysis of the task Worst-Case Execution Time (WCET) in real-time systems. Cache related preemption delays (CRPD) are the most important ones, and are caused by the preempting tasks that modify the cache; the preempted task then suffers an indirect delay after the preemption to reload the cache with useful information.

Preemptive scheduling policies including non-preemptive regions are a hybrid solution between non-preemptive and fully preemptive scheduling paradigms, which enables to conjugate both worlds benefits. In this work [29], we exploit the connection between the progression of a task in its operations, and the knowledge of the preemption delays as a function of its progression. Thus the pessimism in the preemption delay estimation is reduced, in comparison to state of the art methods, due to the increase in information available in the analysis.

7. Contracts and Grants with Industry

7.1. IBM Faculty award

Participant: André Seznec.

The research on Phase Change Memory and security has been partially funded by a 2010 IBM faculty award attributed to André Seznec.

7.2. Nano2012 Mediacom

Participants: Erven Rohou, David Yuste.

Mediacom is a Nano2012 project (Ministry of Industry, INRIA, STMicroelectronics). This project proposes to extend the application domain of virtualization and to combine it with split-compilation, in the context of homogeneous and heterogeneous multicore processors. The goal is to move the compilation complexity from the JIT compiler to the static compilation pass. This would enable very aggressive compilation techniques on embedded systems, such as iterative compilation, polyhedral analysis, or auto-vectorization and auto-parallelization.

8. Partnerships and Cooperations

8.1. HiPEAC2 NoEs

Participants: François Bodin, Pierre Michaud, Erven Rohou, André Seznec.

F. Bodin, P. Michaud, A. Seznec and E. Rohou are members of the European Network of Excellence HiPEAC2. HiPEAC2 addresses the design and implementation of high-performance commodity computing devices in the 10+ year horizon, covering both the processor design, the optimising compiler infrastructure, and the evaluation of upcoming applications made possible by the increased computing power of future devices.

The collaboration with University of Cyprus (Damien Hardy's internship) has been funded by the HiPEAC2 NoE.

8.2. Britanny region fellowship

Participants: Ricardo Andrés Velasquéz, Pierre Michaud, André Seznec.

The Britanny region is funding a Ph.D. fellowship for Ricardo Velasquez on the topic “Fast hybrid multicore architecture simulation”.

8.3. PetaQCD

Participants: Junjie Lai, André Seznec.

Simulation of Lattice QCD is a challenging computational problem that requires very high performance exceeding sustained Petaflops/s. The ANR PetaQCD project combines research groups from computer science, physics and two SMEs (CAPS Entreprise, Kerlabs) to address the challenges of the design of LQCD oriented supercomputer.

8.4. DAL: ERC AdG 2010- 267175, 04-2011/03-2016

Participants: Pierre Michaud, Luis-Germán García Morales, Nathanaël Prémillieu, Erven Rohou, André Seznec, Bharath Narasimha Swamy, Ricardo Andrés Velasquéz.

André Seznec has received an ERC Advanced grant.

We envision that, around 2020, the processor chips will feature a few complex cores and many (may be 1000s) simpler, more silicon and power effective cores. In the DAL research project, we will explore the microarchitecture techniques that will be needed to enable high performance on such heterogeneous processor chips. Very high performance will be required on both sequential sections -legacy sequential codes, sequential sections of parallel applications- and critical threads on parallel applications -e.g. the main thread controlling the application. Our research will focus on enhancing single process performance. On the microarchitecture side, we will explore both a radically new approach, the sequential accelerator, and more conventional processor architectures. We will also study how to exploit heterogeneous multicore architectures to enhance sequential thread performance.

For more informations, see http://www.irisa.fr/alf/dal.

9. Dissemination

9.1. Scientific community animation

- Pierre Michaud is a member of the ISPASS 2012 conference program committee.
- Isabelle Puaut is a member of the program committees of ECRTS 2012 and LCTES 2012. She was a member of the program committee of ECRTS 2011, RTAS 2011, DATE 2011, RTSS 2011, RTNS 2011 and ETFA 2011.
- Damien Hardy was a PC member of the 5th Junior Researcher Workshop on Real-Time Computing, in conjunction with RTNS 2011.
- André Seznec was a member of HPCA 2011, ISCA 2011, CASES 2011, ICCD 2011, and Micro Top Picks 2011 program committees. He is member of ISCA 2012 program committee. He is a member of the editorial board of the IEEE Micro.
- Erven Rohou was a member of the program committee of the PARMA Workshop 2011, and he is in the program committee of PARMA 2012 and of Computing Frontiers 2012.
- Erven Rohou was the Co-Chair of the 3rd International Workshop on GCC Research Opportunities (GROW 2011), colocated with CGO2011 in Chamonix, France.
- Erven Rohou co-organized the “Troisièmes rencontres de la communauté française de compilation”, Dinard, April 2011.
- F. Bodin was director of IRISA till 31/08/2011. IRISA (Institut de Recherche en Informatique et Systèmes Aléatoires) is a joint research unit (UMR 6074), including CNRS, University of Rennes 1, INSA Rennes and ENS Cachan (Brittany site). IRISA laboratory is associated with INRIA.

9.2. Teaching

- F. Bodin, A. Seznec, I. Puaut and E. Rohou are teaching computer architecture and compilation in the master of research in computer science at University of Rennes I.
- Erven Rohou teaches classes and labs of Computing Systems at école Polytechnique (INF422)
- I. Puaut teaches operating systems and real-time systems in the master degree of computer science of the University of Rennes I and at Ecole Supérieure d'ingénieurs de Rennes.
- Pierre Michaud and André Seznec are teaching computer architecture at the engineering degree in computer science at Ecole Supérieure d'ingénieurs de Rennes.
- I. Puaut is co-responsible of the Master of Research in computer science in Britanny (administered jointly by University of Rennes I, University of Bretagne Sud, University of Bretagne Occidentale, INSA de Rennes, ENS Cachan antenne de Bretagne, ENIB, Supélec, Telecom-Bretagne).

9.3. Workshops, seminars, invitations, visitors

- A. Seznec has presented a seminar on "What you will never have wanted to know on branch prediction" at AMD, Austin in Februray 2011.
- A. Seznec has presented a lesson on "Microarchitecture for the dummies" at the winter school on "Hot Topics in Distributed Computing" from March 20th to March 25th 2011 in La Plagne.
- Damien Hardy has presented a seminar on "WCET Analysis of Tasks Executed on Multicore Architectures with Shared Caches" at the University of Cyprus. February, 2011.
- Isabelle Puaut, has presented a lesson on "Estimation de pires temps d'exécution (WCET - Worst-Case Execution Times)" at the "école d'été temps-réel" Brest, August 2011 2011
- Isabelle Puaut has presented a seminar on "WCET estimation: from mono-core to multi-core architectures", at ENSTA Paris in May 2011 and at "Journées temps-réel multiprocesseur " from GDR ASR, Paris, May 2011
- Erven Rohou gave an invited talk at the Technion Computer Engineering Club, Israel Institute of Technology, Haifa, Israel.
- Erven Rohou gave an invited talk at the “Journée PEPI MACS”, organized by INRA, to foster interaction between computer scientists and biologists in need for computational power.
- Erven Rohou gave a talk at the “Quatrièmes rencontres de la communauté française de compilation” in Strasbourg, December 2011.
- Prof. Ahmed El-Mahdy, from the Egyptian-Japanese University of Science and Technology visited Inria (ALF and SYMBIOSE) to discuss collaboration options.

9.4. Miscelleanous

- I. Puaut is a member of the advisory board of the foundation Michel Métivier (http://www.fondation-metivier.org).
- I. Puaut is a member of the Technical Committee on Real-Time Systems of Euromicro, which is responsible for ECRTS, the prime European conference on real-time systems.
- Erven Rohou is a member of the working group GTInria2020 whose mission is to produce the next “Plan Stratégique”.
- A. Seznec is an elected member of the scientific committee of INRIA.
- A. Seznec has been nominated by ACM for 3 years 2011-2013 on the selection committee for the ACM-IEEE Eckert-Mauchly award.
- A. Seznec is a member of the steering committee of ISCA 2011 and ISCA 2012.

10. Bibliography

Major publications by the team in recent years

[1] F. Belletti, S. F. Schifano, R. Tripiccione, F. Bodin, P. Boucaud, J. Micheli, O. Pene, N. Cabibbo, S. de Luca, A. Lonardo, D. Rossetti, P. Vicini, M. Lukyanov, L. Morin, N. Paschedag, H. Simma, V. Morenas, D. Pleiter, F. Rapuano. *Computing for LQCD: ApeNEXT*, in "Computing in Science and Engineering", 2006, vol. 8, n^o 1, p. 18–29, http://dx.doi.org/10.1109/MCSE.2006.4.

[2] F. Bodin, A. Seznec. *Skewed associativity improves performance and enhances predictability*, in "IEEE Transactions on Computers", May 1997.

[3] M. Cornero, R. Costa, R. Fernández Pascual, A. Ornstein, E. Rohou. *An Experimental Environment Validating the Suitability of CLI as an Effective Deployment Format for Embedded Systems*, in "Conference on HiPEAC", Göteborg, Sweden, P. Stenström, M. Dubois, M. Katevenis, R. Gupta, T. Ungerer (editors), Springer, January 2008, p. 130–144.

[4] R. Costa, E. Rohou. *Comparing the size of .NET applications with native code*, in "3rd Intl Conference on Hardware/software codesign and system synthesis", Jersey City, NJ, USA, P. Eles, A. Jantsch, R. A. Bergamaschi (editors), ACM, September 2005, p. 99–104.

[5] J. Dusser, T. Piquet, A. Seznec. *Zero-content augmented caches*, in "Proceedings of the 23rd international conference on Supercomputing", New York, NY, USA, ICS '09, ACM, 2009, p. 46–55, http://doi.acm.org/10.1145/1542275.1542288.

[6] D. Hardy, I. Puaut. *WCET analysis of multi-level non-inclusive set-associative instruction caches*, in "Proc. of the 29th IEEE Real-Time Systems Symposium", Barcelona, Spain, December 2008.

[7] T. Lafage, A. Seznec. *Choosing Representative Slices of Program Execution for Microarchitecture Simulations: A Preliminary Application to the Data Stream*, in "In Workload Characterization of Emerging Applications", Kluwer Academic Publishers, 2000, p. 145–163.

[8] P. Michaud. *Exploiting the Cache Capacity of a Single-chip Multi-core Processor with Execution Migration*, in "Proceedings of the 10th International Conference on High-Performance Computer Architecture (HPCA-10 2004)", IEEE Computer Society, January 2004.

[9] P. Michaud. *STiMuL: a Software for Modeling Steady-State Temperature in Multilayers - Description and user manual*, INRIA, Apr 2010, RT-0385, http://hal.inria.fr/inria-00474286.

[10] P. Michaud, Y. Sazeides, A. Seznec, T. Constantinou, D. Fetis. *A study of thread migration in temperature-constrained multi-cores*, in "ACM Transactions on Architecture and Code Optimization", 2007, vol. 4, n^o 2, 9.

[11] P. Michaud, Y. Sazeides, A. Seznec. *Proposition for a Sequential Accelerator in Future General-Purpose Manycore Processors and the Problem of Migration-Induced Cache Misses*, in "ACM International Conference on Computing Frontiers", Italie Bertinoro, May 2010, http://hal.inria.fr/inria-00471410.

[12] P. MICHAUD, A. SEZNEC, S. JOURDAN. *An Exploration of Instruction Fetch Requirement in Out-of-Order Superscalar Processors*, in "International Journal of Parallel Programming", 2001, vol. 29, n^o 1, p. 35-58.

[13] E. ROHOU, M. SMITH. *Dynamically managing processor temperature and power*, in "Second Workshop on Feedback-Directed Optimizations", 1999.

[14] A. SEZNEC, S. FELIX, V. KRISHNAN, Y. SAZEIDES. *Design trade-offs on the EV8 branch predictor*, in "Proceedings of the 29th International Symposium on Computer Architecture (IEEE-ACM)", Anchorage, May 2002.

[15] A. SEZNEC, P. MICHAUD. *A case for (partially)-tagged geometric history length predictors*, in "Journal of Instruction Level Parallelism (http://www.jilp.org/vol8)", April 2006, http://www.jilp.org/vol8.

[16] A. SEZNEC, N. SENDRIER. *HAVEGE: a user-level software heuristic for generating empirically strong random numbers*, in "ACM Transactions on Modeling and Computer Systems", October 2003.

[17] A. SEZNEC. *Analysis of the O-GEHL branch predictor*, in "Proceedings of the 32nd Annual International Symposium on Computer Architecture", June 2005.

[18] A. SEZNEC. *The L-TAGE Branch Predictor*, in "Journal of Instruction Level Parallelism", May 2007, http://www.jilp.org/vol9.

[19] A. SEZNEC. *A Phase Change Memory as a Secure Main Memory*, in "IEEE Computer Architecture Letters", Feb 2010, http://hal.inria.fr/inria-00468866.

[20] A. SEZNEC. *Decoupled sectored caches: conciliating low tag implementation cost*, in "SIGARCH Comput. Archit. News", 1994, vol. 22, n^o 2, p. 384–393, http://doi.acm.org/10.1145/192007.192072.

Publications of the year

Articles in International Peer-Reviewed Journal

[21] D. HARDY, I. PUAUT. *WCET analysis of instruction cache hierarchies*, in "Journal of system architecture", August 2011, vol. 57, n^o 7 [*DOI :* 10.1016/J.SYSARC.2010.08.007], http://hal.inria.fr/hal-00639454/en.

[22] N. PRÉMILLIEU, A. SEZNEC. *SYRANT: SYmmetric Resource Allocation on Not-taken and Taken Paths*, in "ACM Transactions on Architecture and Code Optimization, Special Issue:Proceedings of the 2012 International Conference on High Performance and Embedded Architectures and Compilers (HiPEAC'12)", January 2012, to appear.

[23] E. ROHOU, K. WILLIAMS, D. YUSTE. *Vectorization Techonology to Improve Interpreter Performance*, in "ACM Transactions on Architecture and Code Optimization, Special Issue:Proceedings of the 2012 International Conference on High Performance and Embedded Architectures and Compilers (HiPEAC'12)", January 2012, to appear.

[24] H. VANDIERENDONCK, A. SEZNEC. *Fairness Metrics for Multithreaded Processors*, in "IEEE Computer Architecture Letters", January 2011 [*DOI :* 10.1109/L-CA.2011.1], http://hal.inria.fr/inria-00564560/en.

[25] H. VANDIERENDONCK, A. SEZNEC. *Managing SMT resource usage through speculative instruction window weighting*, in "ACM Transactions on Architecture and Code Optimization", October 2011 [*DOI :* 10.1145/2019608.2019611], http://hal.inria.fr/hal-00639171/en.

International Conferences with Proceedings

[26] A. BOUAKAZ, I. PUAUT, E. ROHOU. *Predictable Binary Code Cache: A First Step Towards Reconciling Predictability and Just-In-Time Compilation*, in "The 17th IEEE Real-Time and Embedded Technology and Applications Symposium", Chicago, United States, Marco Caccamo, April 2011, http://hal.inria.fr/inria-00589690/en.

[27] J. DUSSER, A. SEZNEC. *Decoupled Zero-Compressed Memory*, in "Proceeding HiPEAC '11 Proceedings of the 6th International Conference on High Performance and Embedded Architectures and Compilers", Heraklion, Greece, HiPEAC, January 2011 [*DOI :* 10.1145/1944862.1944876], http://hal.inria.fr/inria-00638904/en.

[28] D. HARDY, B. LESAGE, I. PUAUT. *Scalable Fixed-Point Free Instruction Cache Analysis*, in "The 32nd IEEE Real-Time Systems Symposium (RTSS 2011)", Vienne, Austria, November 2011, http://hal.inria.fr/inria-00638698/en.

[29] J. MARINHO, V. NÉLIS, S. M. PETTERS, I. PUAUT. *Preemption Delay Analysis for Floating Non-Preemptive Region Scheduling*, in "DATE 2012: Design, Automation and Test in Europe", March 2012, To appear.

[30] P. MICHAUD. *Replacement policies for shared caches on symmetric multicores : a programmer-centric point of view*, in "6th International Conference on High-Performance and Embedded Architectures and Compilers", Heraklion, Greece, January 2011 [*DOI :* 10.1145/1944862.1944890], http://hal.inria.fr/inria-00531188/en.

[31] D. NUZMAN, S. DYSHEL, E. ROHOU, I. ROSEN, K. WILLIAMS, D. YUSTE, A. COHEN, A. ZAKS. *Vapor SIMD: Auto-Vectorize Once, Run Everywhere*, in "International Symposium on Code Generation and Optimization", Chamonix, France, Olivier Temam, April 2011, http://hal.inria.fr/inria-00589692/en.

[32] M. QURESHI, A. SEZNEC, L. LUIS, M. FRANCESCHINI. *Practical and secure PCM systems by online detection of malicious write streams*, in "2011 IEEE 17th International Symposium on High Performance Computer Architecture", San Antonio, United States, IEEE, February 2011 [*DOI :* 10.1109/HPCA.2011.5749753], http://hal.inria.fr/inria-00638950/en.

[33] E. ROHOU, S. DYSHEL, D. NUZMAN, I. ROSEN, K. WILLIAMS, A. COHEN, A. ZAKS. *Speculatively Vectorized Bytecode*, in "International Conference on High-Performance and Embedded Architectures and Compilers", Heraklion, Greece, ACM, January 2011, http://hal.inria.fr/inria-00525139/en.

[34] A. SEZNEC. *A 64 Kbytes ISL-TAGE branch predictor*, in "JWAC-2: Championship Branch Prediction", San Jose, United States, JILP, June 2011, http://hal.inria.fr/hal-00639040/en.

[35] A. SEZNEC. *A 64-Kbytes ITTAGE indirect branch predictor*, in "JWAC-2: Championship Branch Prediction", San Jose, United States, JILP, June 2011, http://hal.inria.fr/hal-00639041/en.

[36] A. SEZNEC. *A New Case for the TAGE Branch Predictor*, in "MICRO 2011 : The 44th Annual IEEE/ACM International Symposium on Microarchitecture", Porto Allegre, Brazil, ACM (editor), ACM-IEEE, December 2011, http://hal.inria.fr/hal-00639193/en.

[37] A. SEZNEC. *Storage Free Confidence Estimator for the TAGE predictor*, in "17th High Performance Computer Architecture", San Antonio, United States, IEEE, February 2011 [*DOI :* 10.1109/HPCA.2011.5749750], http://hal.inria.fr/inria-00638890/en.

[38] M. M. WALIULLAH, P. STENSTROM. *Classification and Elimination of Conflicts in Hardware Transactional Memory Systems*, in "23rd International Symposium on Computer Architecture and High Performance Computing - SBAC-PAD'2011", Vitoria, Brazil, IEEE, October 2011, http://hal.inria.fr/hal-00640813/en.

Research Reports

[39] J. LAI, A. SEZNEC. *TEG: GPU Performance Estimation Using a Timing Model*, INRIA, November 2011, n^{o} RR-7804, http://hal.inria.fr/hal-00641726/en.

[40] P. MICHAUD. *Hardware acceleration of sequential loops*, INRIA, November 2011, n^{o} RR-7802, http://hal.inria.fr/hal-00641350/en.

[41] E. ROHOU. *Tiptop: Hardware Performance Counters for the Masses*, INRIA, November 2011, n^{o} RR-7789, http://hal.inria.fr/hal-00639173/en.

[42] R. A. VELASQUEZ, P. MICHAUD, A. SEZNEC. *BADCO: Behavioral Application-Dependent superscalar Core Models*, INRIA, November 2011, n^{o} RR-7795, http://hal.inria.fr/hal-00641446/en.

References in notes

[43] G. M. AMDAHL. *Validity of the Single Processor Approach to Achieving Large Scale Computing Capabilities*, in "SJCC.", 1967, p. 483–485.

[44] D. BURGER, T. M. AUSTIN. *The simplescalar tool set, version 2.0*, 1997.

[45] R. S. CHAPPELL, J. STARK, S. P. KIM, S. K. REINHARDT, Y. N. PATT. *Simultaneous subordinate microthreading (SSMT)*, in "ISCA '99: Proceedings of the 26th annual international symposium on Computer architecture", Washington, DC, USA, IEEE Computer Society, 1999, p. 186–195, http://doi.acm.org/10.1145/300979.300995.

[46] C. FERDINAND, R. WILHELM. *Efficient and Precise Cache Behavior Prediction for Real-Time Systems*, in "Real-Time Syst.", 1999, vol. 17, n^{o} 2-3, p. 131–181, http://dx.doi.org/10.1023/A:1008186323068.

[47] T. S. KARKHANIS, J. E. SMITH. *A First-Order Superscalar Processor Model*, in "Proceedings of the International Symposium on Computer Architecture", Los Alamitos, CA, USA, IEEE Computer Society, 2004, 338, http://doi.ieeecomputersociety.org/10.1109/ISCA.2004.1310786.

[48] B. LEE, J. COLLINS, H. WANG, D. BROOKS. *CPR : composable performance regression for scalable multiprocessor models*, in "Proceedings of the 41st International Symposium on Microarchitecture", 2008.

[49] Y. LIANG, T. MITRA. *Cache modeling in probabilistic execution time analysis*, in "DAC '08: Proceedings of the 45th annual conference on Design automation", New York, NY, USA, ACM, 2008, p. 319–324, http://doi.acm.org/10.1145/1391469.1391551.

[50] T. LUNDQVIST, P. STENSTRÖM. *Timing Anomalies in Dynamically Scheduled Microprocessors*, in "RTSS '99: Proceedings of the 20th IEEE Real-Time Systems Symposium", Washington, DC, USA, IEEE Computer Society, 1999.

[51] L. RAUCHWERGER, Y. ZHAN, J. TORRELLAS. *Hardware for Speculative Run-Time Parallelization in Distributed Shared-Memory Multiprocessors*, in "HPCA '98: Proceedings of the 4th International Symposium on High-Performance Computer Architecture", Washington, DC, USA, IEEE Computer Society, 1998, 162.

[52] T. SHERWOOD, E. PERELMAN, G. HAMERLY, B. CALDER. *Automatically characterizing large scale program behavior*, in "In Proceedings of the 10th International Conference on Architectural Support for Programming Languages and Operating Systems", 2002, p. 45–57.

[53] K. SKADRON, M. STAN, W. HUANG, S. VELUSAMY. *Temperature-aware microarchitecture*, in "Proceedings of the International Symposium on Computer Architecture", 2003.

[54] J. G. STEFFAN, C. COLOHAN, A. ZHAI, T. C. MOWRY. *The STAMPede approach to thread-level speculation*, in "ACM Trans. Comput. Syst.", 2005, vol. 23, n^{o} 3, p. 253–300, http://doi.acm.org/10.1145/1082469.1082471.

[55] V. SUHENDRA, T. MITRA. *Exploring locking & partitioning for predictable shared caches on multi-cores*, in "DAC '08: Proceedings of the 45th annual conference on Design automation", New York, NY, USA, ACM, 2008, p. 300–303, http://doi.acm.org/10.1145/1391469.1391545.

[56] D. M. TULLSEN, S. EGGERS, H. M. LEVY. *Simultaneous Multithreading: Maximizing On-Chip Parallelism*, in "Proceedings of the 22th Annual International Symposium on Computer Architecture", 1995.

[57] D. M. TULLSEN, S. EGGERS, H. M. LEVY. *Simultaneous Multithreading: Maximizing On-Chip Parallelism*, in "Proceedings of the 22th Annual International Symposium on Computer Architecture", June 1995.

[58] J. YAN, W. ZHAN. *WCET Analysis for Multi-Core Processors with Shared L2 Instruction Caches*, in "Proceedings of Real-Time and Embedded Technology and Applications Symposium, 2008. RTAS '08.", 2008, p. 80-89.