IN PARTNERSHIP WITH:
Université Rennes 1

Activity Report 2012

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, Compilation, Real-time, Complexity, Multicore

Creation of the Project-Team: January 01, 2009 *, Updated into Project-Team:* January 01, 2011 .

1. Members

Research Scientists

André Seznec [Team leader, Research Director Inria, HdR]
Sylvain Collange [Resarch Scientist Inria, from 01/10/12]
Pierre Michaud [Research Scientist Inria]
Erven Rohou [Resarch Director Inria]

Faculty Members

François Bodin [Professor University of Rennes I, from 01/12/12, HdR]
Damien Hardy [Associate Professor, University of Rennes 1, from 01/09/12]
Isabelle Puaut [Professor, University of Rennes 1, HdR]

Engineer

Emmanuel Riou [Inria from 01/11/12]

PhD Students

Luis-Germán García Morales [Inria Allocation]
Sajith Kalathingal [Inria Allocation from 01/12/12]
Benjamin Lesage [MESR Allocation, University of Rennes 1]
Junjie Lai [Inria Allocation]
Hanbing Li [Inria Allocation from 01/10/12]
Bharath Narasimha Swamy [Inria Allocation]
Surya Narayanan [Inria Allocation]
Arthur Pérais [Inria Allocation from 01/10/12]
Nathanaël Prémillieu [MESR Allocation, University of Rennes 1]
Arjun Suresh [Inria Allocation from 01/11/12]
Ricardo Andrés Velásquez [Inria Allocation]

Post-Doctoral Fellow

Kamil Kedzierski [Inria]

Administrative Assistants

Virginie Desroches [TR Inria from 01/06/12]
Evelyne Livache [TR Inria till 31/05/12]

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 implement heterogeneous design featuring 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 (Intel), the Brittany region, the ANR (PetaQCD and W-SEPT project)s, and the European Union (NoE HiPEAC3, ERC grant DAL, COST action TACLe).

2.2. Highlights of the Year

- André Seznec has received the **first Intel Research Impact Medal** for "His exemplary work on high-performance computer micro-architectures, branch prediction, and cache architecture, have been of tremendous benefit to Intel, the industry, and the academic community as a whole.". (See http://www.intel.es/content/www/us/en/education/university/university-research-award.html).
- André Seznec has been elevated as an IEEE Fellow "for contributions to design of branch predictors and cache memory for processor architectures".

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 doubled 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) featured 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 successful architecture will also be determined by the abilitty 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 tool chain 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 [34] 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 [36]). Such simulations are CPU consuming and simulations cannot be run on a complete application. Sampling representative slices of the application was proposed [5] and popularized by the Simpoint [45] 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 [38]), accurate evaluation of the WCET of an algorithm on a complex uniprocessor system is a difficult problem. Accurately modelling data cache behavior [4] 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) [42].

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 [49].

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 [14], [13]. 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 [37]. 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 [44], but the most promising solutions integrate hardware support for software thread-level speculation [47].

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* [10] through techniques such as clock gating or throttling and *activity migration* [46][7].

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 [9], 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 [39] 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 [40] 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 [3]. Moreover, this bytecode representation can be compiled to native code with performance similar to static compilation [2]. 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) [42]. 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 [41] 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 [50], therefore hardware-based or compiler methods such as [48] 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

Performance, processor architecture, compilers,telecommunications, multimedia, biology, health, engineering, environment, transportation

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.

Microarchitecture temperature model **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.

Microarchitecture temperature modeling

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 [6]. 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.

Address trace compression **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.

Unpredictable random number generator

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 [12]. 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.

Performance, hardware counters, analysis tool.

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.
- Any expression can be computed, using the basic arithmetic operators, constants, and counter values.
- A configuration file lets users define their prefered setup, as well as custom expressions.

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 within the ERC DAL project

Participants: Pierre Michaud, Nathanaël Prémillieu, Luis Germán Garcia Morales, Bharath Narasimha Swamy, Sylvain Collange, André Seznec, Arthur Pérais, Surya Narayanan, Sajith Kalathingal, Kamil Kedzierski.

Processor, cache, locality, memory hierarchy, branch prediction, multicore, power, temperature

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, http://www.irisa.fr/alf/dal, we 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 focuses essentially on enhancing single processes performance.

6.1.1. Microarchitecture exploration of control flow reconvergence

Participants: Nathanaël Prémillieu, André Seznec.

After continuous progress over the past 15 years [14], [13], 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 [17].

As a follower work, we are now focusing on exploiting control flow reconvergence in the special case of predication. When the target ISA has predicated instruction, it is possible to transform control dependencies into data dependencies. This process is called if-conversion. As a result, the two paths of a conditional branch is merge into one path. Hence exploiting the principles developed in SYRANT is much easier than for a standard ISA.

6.1.2. Memory controller

Participant: André Seznec.

The memory controller has become one of the performance enablers of a computer system. Its impact is even higher on multicores than it was on uniprocessor systems. We propose the sErvice Value Aware memory scheduler (EVA) to enhance memory usage. EVA builds on two concepts, the request weight and the per-thread traffic light. For a read request on memory, the request weight is an evaluation of the work allowed by the request. Per-thread traffic lights are used to track whether or not in a given situation it is worth to service requests from a thread, e.g. if a given thread is blocked by refreshing on a rank then it is not worth to serve requests from the same thread on another rank. The EVA scheduler bases its scheduling decision on a service value which is heuristically computed using the request weight and per-thread traffic lights. Our EVA scheduler implementation relies on several hardware mechanisms, a request weight estimator, per-thread traffic estimators and a next row predictor. Using these components, our EVA scheduler estimates scores to issue scheduling decisions. EVA was shown to perform efficiently and fairly compared with previous proposed memory schedulers [21]

6.1.3. Performance and power models for heterogeneous muticores

Participants: Kamil Kedzierski, André Seznec.

In the DAL project, we expect architectures to be a combination of many simple cores for parallel execution and sequential accelerators [8] built on top of complex cores for ILP intensive tasks. For evaluating these architectures, we need performance and power models. We design a parallel manycore simulator, built with pthread implementation. Such an approach allows us to maintain flexibility and scalability: our goal is to scale well both when we vary the number of cores used to perform simulation, and as we vary the number of cores being simulated. Our implementation also allows to configure each core independently for the heterogeneous architectures. Preliminary results show that the simulator uses with very small memory footprint, which is crucial for the manycore studies with number of cores constantly increasing.

A new power management approach is needed for these future manycore processors that employ both sequential accelerators and simple cores. This is due to the fact that the frequency at which a given core operates is highly correlated with the cores' size (and thus a task that the core performs). Therefore, we built Dynamic Voltage Frequency Scaling model for the on-chip voltage regulator (VR) case, as we believe that future architectures will incorporate VRs on chip.

6.1.4. Designing supercores

Participants: Pierre Michaud, Luis Germán García Morales, André Seznec.

In the framework of the DAL project, we study super-cores that could achieve very high clock frequency and a high instruction per cycle rate (IPC). The current objective is to explore the design space of possible configurations for the microarchitecture that are suitable in terms of performance, area and power for the super-core. In particular, we focus on the back-end of the microarchitecture. A way to increase the IPC is to allow the core processing more instructions simultaneously e.g. increasing the issue width. This can be done for example by replicating the functional units (FU) inside the core. However keeping the same frequency could become very challenging. Clustering of FUs is a technique that helps designers to overcome this problem, even though other problems might appear e.g. IPC loss compared to an ideal monolithic back-end due to inter-cluster delays. We have started exploring different cluster schemes and instruction steering policies with the purpose of having a wide-issue clustered microarchitecture with a high IPC, a high frequency and the problem of inter-cluster delay minimized.

6.1.5. Helper threads

Participants: Bharath Narasimha Swamy, André Seznec.

Improving sequential performance will be key to both performance on single threaded codes and scalability on parallel codes. Complex out-of-order execution processors that aggressively exploit instruction level parallelism are the obvious design direction to improve sequential performance. However, ability of these complex cores to deliver performance will be undermined by performance degrading events such as branch mis-predictions and cache misses that limit the achievable instruction throughput. As an alternative to the monolithic complex core approach, we propose to improve sequential performance on emerging heterogeneous many core architectures by harnessing (unutilized) additional cores to work as helper cores for the sequential code. Helper cores can be employed to mitigate the impact of performance degrading events and boost sequential performance, for example by prefetching data for the sequential code ahead of time.

We are currently pursuing two directions to utilize helper cores. (1) We explore the use of helper cores to emulate prefetch algorithms in software. We will adapt and extend existing prefetch mechanisms for use on the helper cores and evaluate mechanisms to utilize both compute and cache resources on the helper cores to prefetch for the main thread. We intend to target delinquent load/store instructions that cause most of the cache misses and prefetch data ahead of time, possibly even before the hardware prefetchers on the main core. (2) We explore the use of helper cores to execute pre-computation code and generate prefetch requests for the main thread. Pre-computation code is constructed from the main thread and targets to capture the data access behavior of the main thread, particularly for irregular data access patterns in control-flow dominated code. We will explore algorithms to generate pre-computation code and evaluate mechanisms for communication and synchronization between the main thread and the helper cores, specifically in the context of a heterogenous many core architecture.

6.1.6. What makes parallel code sections and sequential code sections different?

Participants: Surya Natarajan, André Seznec.

In few years from now, single die processor components will feature many cores. They can be symmetric/asymmetric or homogeneous/heterogeneous cores. The utilization of these cores depends on the application and the programming model used. We have initiated a study on understanding the difference in nature between the parallel and sequential code sections in parallel applications. Initial experiments show that instruction mix of the serial and parallel parts are different. For example, contribution of the conditional branches are dominant in serial part and data transfer instructions are dominant in the parallel part. By experimentation, we infer that the conditional branch prediction in serial part needs a bigger branch predictor compared to the parallel part. Later, we would like to define the hardware mechanisms that are needed for cost effective execution of parallel sections; cost-effective meaning silicon and energy effective since parallelism can be leveraged.

On the other hand, the shared memory model has critical sections in the parallel sections, which makes the parallel sections sequential at times. We will try to characterize the nature of these sequential code sections and particularly identify their potential bottlenecks. The objective is to address the performance bottlenecks on sequential sections through new microarchitecture and/or compiler mechanisms.

6.1.7. Revisiting Value Prediction

Participants: Arthur Pérais, André Seznec.

Value prediction was proposed in the mid 90's to enhance the performance of high-end microprocessors. The research on Value Prediction techniques almost vanished in the early 2000's as it was more effective to increase the number of cores than to dedicate silicon to Value Prediction. However high end processor chips currently feature 8-16 high-end cores and the technology will allow to implement 50-100 of such cores on a single die in a foreseeable future. Amdahl's law suggests that the performance of most workloads will not scale to that level. Therefore, dedicating more silicon area to value prediction in high-end cores might be considered as worthwhile for future multicores.

We introduce a new value predictor VTAGE harnessing the global branch history [32]. VTAGE directly inherits the structure of the indirect jump predictor ITTAGE[11]. VTAGE is able to predict with a very high accuracy many values that were not correctly predicted by previously proposed predictors, such as the FCM predictor and the stride predictor. Three sources of information can be harnessed by these predictors: the global

branch history, the differences of successive values and the local history of values. Moreover we show that the predictor components using these sources of information are all amenable to very high accuracy at the cost of some prediction coverage.

Compared with these previously proposed solutions, VTAGE can accommodate very long prediction latencies. The introduction of VTAGE opens the path to the design of new hybrid predictors. Using SPEC 2006 benchmarks, our study shows that with a large hybrid predictor, in average 55-60 % of the values can be predicted with more than 99.5 % accuracy. Evaluation of effective performance benefit is an on-going work.

6.1.8. Augmenting superscalar architecture for efficient many-thread parallel execution

Participants: Sylvain Collange, Sajith Kalathingal, André Seznec.

Heterogeneous multi-core architectures create many issues for test, design and optimizations. They also necessitate costly data transfer from the complex cores to the simple cores when switching from the parallel to sequential sections and vice-versa. We have initiated research on designing a unique core that efficiently run both sequential and massively parallel sections. It will explore how the architecture of a complex superscalar core has to be modified or enhanced to be able to support the parallel execution of many threads from the same application (10's or even 100's a la GPGPU on a single core). The overall objective is to support both sequential codes and very parallel execution, particularly data parallelism, on the same hardware core.

6.2. Other Architecture Studies

Participants: Damien Hardy, Pierre Michaud, Ricardo Andrés Velásquez, Sylvain Collange, André Seznec, Junjie Lai.

GPU, performance, simulation, vulnerability

6.2.1. Analytical model to estimate the performance vulnerability of caches and predictors to permanent faults

Participant: Damien Hardy.

This research was partially undertaken during Damien Hardy's stay in the Computer Architecture group of the University of Cyprus (January-August 2012).

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 or simply in performance loss. 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.

Virtually all previous micro-architectural work aiming to assess the performance implications of permanently faulty cells relies on simulations with random fault-maps, assumes that faulty blocks are disabled, and focuses on architectural arrays such as caches.

These studies are, therefore, limited by the fault-maps they use that may not be representative for the average and distributed performance. Moreover, they are incomplete by ignoring faults in non-architectural arrays, such as predictors, that do not affect correctness but can degrade performance.

In [20], an analytical model is proposed for understanding the implications on performance of permanently faulty cells in caches and predictors. The model for a given program execution, micro-architectural configuration, and probability of cell failure, provides rapidly the *Performance Vulnerability Factor (PVF)*. PVF is a direct measure of the performance degradation due to permanent faults. In particular, the model can determine the expected PVF as well as the PVF probability distribution bounds without using an arbitrary number of random fault-maps.

The model, once derived, can be used to explore processor behavior with different cell probability of failures. This can be helpful to forecast how processor performance may be affected by faults in the future. Additionally, this information can be useful to determine which arrays have significant PVF and make design decisions to reduce their PVF, for example through a protection mechanism, using larger cells, or even by selecting a different array organization.

6.2.2. GPU-inspired throughput architectures

Participant: Sylvain Collange.

This research was partially undertaken while Sylvain Collange was with Universidade Federal de Minas Gerais, Belo Horizonte - Brazil, (January-September 2012).

In an heterogeneous architecture where power is the primary performance constraint, parallel sections of applications need to run on throughput-optimized cores that focus on energy efficiency. The Single-Instruction Multiple Thread (SIMT) execution model introduced for Graphics Processing Units (GPUs) provides inspiration to design such future energy-efficient throughput architectures. However, the performance of SIMT architectures is vulnerable to control and data flow divergences across threads. It limits its applicability to regular data-parallel applications. We work on making SIMT architectures more efficient, and generalizing the SIMT model to general-purpose architectures.

First, hybrids between multi-thread architectures and SIMT architectures can achieve a tradeoff between energy efficiency and flexibility [35]. Second, the same concepts that benefit GPUs may be applied to vectorize dynamically single-program, multi-thread applications. Indeed, data-parallel multi-thread workloads, such as OpenMP applications, expose parallelism by running many threads executing the same program. These threads may be synchronized to run the same instructions at the same time. SPMD threads also commonly perform the same computation on the same value. We take advantage from these correlations by sharing instructions between threads. It promises to save energy and frees processing resources on multi-threaded cores [26].

Besides architecture-level improvements, the efficiency of SIMT architectures can be improved through compiler-level code optimization. By maintaining a large number of threads in flight (in the order of tens of thousands), GPUs suffer from high cache contention as the local working set of each thread increases. This raises challenges as memory accesses are costly in terms of energy. Divergence analysis is a compiler pass that identifies similarities in the control flow and data flow of concurrent threads. In particular, it detects program variables that are affine functions of the thread identifier. Register allocation can benefit from divergence analysis to unify affine variables across SIMT threads and re-materialize them when needed. It reduces the volume of register spills, relieving pressure on the memory system [28].

6.2.3. Behavioral application-dependent superscalar core modeling

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

Behavioral superscalar core modeling is a possible way to trade accuracy for processor 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 to build a model. After the model is built, it can be used for simulating uncores. 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 [29].

In a later work [33], we have shown that fast approximate microarchitecture models such as BADCO can also be very useful for selecting multiprogrammed workloads for evaluating the throughput of multicore processors. Computer architects usually study multiprogrammed workloads by considering a set of benchmarks and some combinations of these benchmarks. However, there is no standard method for selecting such sample, and different authors have used different methods. The choice of a particular sample impacts the conclusions of a study. Using BADCO, we propose and compare different sampling methods for defining multiprogrammed workloads for computer architecture [33]. We evaluate their effectiveness on a case study, the comparison of

several multicore last-level cache replacement policies. We show that random sampling, the simplest method, is robust to define a representative sample of workloads, provided the sample is big enough. We propose a method for estimating the required sample size based on fast approximate simulation. We propose a new method, workload stratification, which is very effective at reducing the sample size in situations where random sampling would require large samples.

6.2.4. Performance Upperbound Analysis of GPU applications

Participants: Junjie Lai, André Seznec.

In the framework of the ANR Cosinus PetaQCD project, we are modeling the demands of high performance scientific applications on hardware. GPUs have become popular and cost-effective hardware platforms. In this context, we have been addressing the gap between theoretical peak performance on GPU and the effective performance [22]. There has been many studies on optimizing specific applications on GPU as well as and also a lot of studies on automatic tuning tools. However, the gap between the effective performance and the maximum theoretical performance is often huge. A tighter performance upperbound of an application is needed in order to evaluate whether further optimization is worth the effort. We designed a new approach to compute the CUDA application's performance upperbound through intrinsic algorithm information coupled with low-level hardware benchmarking. Our analysis [30] allows us to understand which parameters are critical to the performance and therefore to get more insight on the performance result. As an example, we analyzed the performance upperbound of SGEMM (Single-precision General Matrix Multiply) on Fermi and Kepler GPUs. Through this study, we uncover some undocumented features on Kepler GPU architecture. Based on our analysis, our implementations of SGEMM achieve the best performance on Fermi and Kepler GPUs so far (5 % improvement on average).

6.2.5. Multicore throughput metrics

Participant: Pierre Michaud.

Several different metrics have been proposed for quantifying the throughput of multicore processors. There is no clear consensus about which metric should be used. Some studies even use several throughput metrics. We have shown several new results concerning multicore throughput metrics [16]. We have exhibited the relation between single-thread average performance metrics and throughput metrics, emphasizing that throughput metrics inherit the meaning or lack of meaning of the corresponding single-thread metric [16]. In particular, two of the three most frequently used throughput metrics in microarchitecture studies, the weighted speedup and the harmonic mean of speedups, are inconsistent: they do not give equal importance to all benchmarks. We have demonstrated that the weighted speedup favors unfairness. We have shown that the harmonic mean of IPCs, a seldom used throughput metric, is actually consistent and has a physical meaning. We have explained under which conditions the arithmetic mean or the harmonic mean of IPCs can be used as strong indicators of throughput increase.

In a subsequent work [31], we have pointed out a problem with commonly used multiprogram throughput metrics, which is that they are based on the assumption that all the jobs execute for a fixed and equal time. We argue that this assumption is not realistic. We have proposed and characterized some new throughput metrics based on the assumption that jobs execute a fixed and equal quantity of work. We have shown that using such equal-work throughput metric may change the conclusion of a microarchitecture study [31].

6.3. Compiler, vectorization, interpretation

Participants: Erven Rohou, Emmanuel Riou, Arjun Suresh, André Seznec.

The usage of the bytecode-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.

6.3.1. Vectorization Technology To Improve Interpreter Performance

Participant: Erven Rohou.

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. Finally, we finely analyze the impact of our extension on the behavior of the caches and branch predictors.

These results are published in ACM TACO [18], and will be presented at the HiPEAC 2013 conference.

6.3.2. 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 [27] 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.

Tiptop has been extended to display any user-defined arithmetic expression based on constants and counter values. A new configuration file lets users defined their default parameters as well as custom expressions.

6.3.3. Code obfuscation and JIT Compilers

Participant: Erven Rohou.

This project proposes to leverage JIT compilation to make software tamper-proof. The idea is to constantly generate different versions of an application, even while it runs, to make reverse engineering hopeless. A strong random number generator will guarantee that generated code is not reproducible, though the functionality is the same. Performance will not be sacrificed thanks to multi-core architectures: the JIT runs on separate cores, overlapping with the execution of the application.

The following directions are investigated:

1. We proposed a "change metric" that evaluates how different each new version of a function differs from the previous one, and hence contributes to the robustness of the system. The metric is based on string matching (such as in bioinformatics).
2. To increase the frequency of code switching, we consider on-stack-replacement. For performance, compilation is performed on a separate thread and pre-copying of the stack state to the new function version, thereby saving switching time.
3. We decompose a thread control-flow graph into many control-flow graphs such that the result of execution would be the same. The control-flow complexity is substantial as there are in the order of $O(n^n)$ possible combinations (where n is the number of threads and compilation units).

This is done in collaboration with the group of Prof. Ahmed El-Mahdy at E-JUST, Alexandria, Egypt.

6.3.4. Dynamic Analysis and Re-Optimization of Executables

Participants: Erven Rohou, Emmanuel Riou.

The objective of the ADT PADRONE beginning in November 2012 is to design and develop a platform for re-optimization of binary executables at run-time. We reviewed available support in hardware (such as performance monitoring unit, trap instructions), and in the Linux operating system (such as the ptrace system call). We started working on the platform, with an initial focus on analysis techniques.

6.3.5. Improving single core execution in the many-core era

Participants: Erven Rohou, André Seznec, Arjun Suresh.

In the framework of the DAL research project, we have initiated compiler research on using available unused resources in multicores to improve the performance of sequential code segments. Helper threads, driven by automated compiler infrastructure, can alleviate potential performance degradation due to resource contention. For example, loop based applications experiencing bad memory locality can be re-optimized by a just-in-time compiler to adjust to actual hardware characteristics.

6.4. WCET estimation

Participants: Damien Hardy, Benjamin Lesage, Hanbing Li, 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) multi-core architectures (ii) preemption delay analysis (iii) traceability of flow information in compilers for WCET estimation.

6.4.1. WCET estimation and multi-core systems

6.4.1.1. 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 [23], 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.

6.4.1.2. WCET-oriented cache partitioning for multi-core systems

Participant: Isabelle Puaut.

Multi-core architectures are well suited to fulfill the increasing performance requirements of embedded real-time systems. However, such systems also require the capacity to estimate the timing behavior of their critical components. Interference between tasks, as they occur on standard multi-core micro-architectures due to cache sharing are still difficult to predict accurately. An alternative is to remove these indirect interferences between tasks through partitioning of the shared cache and through the use of partitioned task scheduling.

In this work [19], we have proposed a new algorithm for joint task and cache partitioning in multi-core systems scheduled using non-preemptive Earliest Deadline First policy. The main novelty of the algorithm is to take into account the tasks' period repartition in the task partitioning problem, which is critical in a non-preemptive context. Other task properties such as task cache requirements are also considered to optimize cache partitioning. Experiments show that our algorithm outperforms the state-of-the-art algorithm for tasks and cache partitioning, named IA3 [43], in terms of schedulability, specially when the spectrum of tasks periods is wide.

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 [25], 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. The method proposed in [25] was later improved in [24], to extract more information on the code and further decrease the CRPD estimation.

6.4.3. Traceability of flow information for WCET estimation

Participants: Hanbing Li, Isabelle Puaut, Erven Rohou.

Control-flow information is mandatory for WCET estimation, to guarantee that programs terminate (e.g. provision of bounds for the number of loop iterations) but also to obtain tight estimates (e.g. identification of infeasible or mutually exclusive paths). Such flow information is expressed though annotations, that may be calculated automatically by program/model analysis, or provided manually.

The objective of this work is to address the challenging issue of the mapping and transformation of the flow information from high level down to machine code. In a first step, we will consider only the issue of conveying information through the compilation flow, without any optimization. Then, we will study the impact of optimizations on the traceability of annotations.

This research started in October 2012 and is part of the ANR W-SEPT project.

7. Bilateral Contracts and Grants with Industry

7.1. Intel Research Grant

Participant: André Seznec.

Intel is supporting the research of the ALF project-team on "Alternative ways for improving uniprocessor performance".

8. Partnerships and Cooperations

8.1. European Initiatives

8.1.1. 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 Velásquez, Arthur Pérais, Surya Narayanan, Arjun Suresh, Sajith Kalathingal, Kamil Kedzierski.

In the DAL, Defying Amdahl's Law project, 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 information, see http://www.irisa.fr/alf/dal.

8.1.2. HiPEAC3 NoE

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 HiPEAC3. HiPEAC3 addresses the design and implementation of high-performance commodity computing devices in the 10+ year horizon, covering both the processor design, the optimizing compiler infrastructure, and the evaluation of upcoming applications made possible by the increased computing power of future devices.

8.1.3. COST Action TACLe - Timing Analysis on Code-Level 10-2012/09-2015

Participants: Damien Hardy, Isabelle Puaut.

Embedded systems increasingly permeate our daily lives. Many of those systems are business- or safety-critical, with strict timing requirements. Code-level timing analysis is indispensable to ascertain whether these requirements are met. However, recent developments in hardware, especially multicore processors, and software organization make the analysis increasingly harder, thus challenging the evolution of timing analysis techniques. Principles for building "timing-composable" embedded systems are needed to make timing analysis tractable in the future. The furthering and consolidation of those principles require increased contacts within the timing analysis community as well as with the neighboring communities that deal with other forms of analysis, such as model checking and type inference, and with computer architectures and compilers. The goal of this COST Action (http://www.cost.eu/domains_actions/ict/Actions/IC1202) is to gather these forces in order to develop industrial strength code-level timing analysis techniques for future generation embedded systems.

Twelve countries are currently involved in this COST action.

8.2. Regional Initiative

8.2.1. Britanny region fellowship

Participants: Ricardo Andrés Velásquez, 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. National Initiatives

8.3.1. ANR PetaQCD 01-2009/10-2012

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.

8.3.2. ANR W-SEPT

Participants: Hanbing Li, Isabelle Puaut, Erven Rohou.

Critical embedded systems are generally composed of repetitive tasks that must meet drastic timing constraints, such as termination deadlines. Providing an upper bound of the worst-case execution time (WCET) of such tasks at design time is thus necessary to prove the correctness of the system. Static WCET estimation methods, although safe, may produce largely over-estimated values. The objective of the project is to produce tighter WCET estimates by discovering and transforming flow information at all levels of the software design process, from high level-design models (e.g. Scade, Simulink) down to binary code. The ANR W-SEPT project partners are Verimag Grenoble, IRIT Toulouse, Inria Rennes. A case study is provided by Continental Toulouse.

8.3.3. Large Scale Initiative: Large scale multicore virtualization for performance scaling and portability

Participant: Erven Rohou.

An Inria Large Scale Initiative (Action d’Envergure) has been submitted and approved. It is entitled “Large scale multicore virtualization for performance scaling and portability”. Partner project-teams include: ALF, ALGORILLE, CAMUS, REGAL, RUNTIME, as well as DALI.

This project aims to build collaborative virtualization mechanisms that achieve essential tasks related to parallel execution and data management. We want to unify the analysis and transformation processes of programs and accompanying data into one unique virtual machine.

8.3.4. ADT PADRONE 2012-2014

Participants: Erven Rohou, Emmanuel Riou.

Computer science is driven by two major trends: on the one hand, the lifetime of applications is much larger than the lifetime of the hardware for which they are initially designed; on the other hand the diversity of computing hardware keeps increasing. The net result is that many applications are not optimized for their current executing environment. The objective of PADRONE is to design and develop a platform for re-optimization of binary executables at run-time. There are many advantages: actual hardware is known, the whole application is visible (including libraries), profiling can be collected, and source code is not necessary (interesting in the case of proprietary applications).

8.4. International Initiative

8.4.1. PHC Imhotep (Egypt): Code obfuscation through JIT compilation, Jan 2012 – Dec 2013

Participant: Erven Rohou.

Collaboration with Pr Ahmed El-Mahdy, Egypt-Japan University for Science and Technology (Alexandria, Egypt)

This project proposes to leverage JIT compilation to make software tamper-proof. The idea is to constantly generate different versions of an application, even while it runs, to make reverse engineering hopeless. A strong random number generator will guarantee that generated code is not reproducible – though the functionality is the same. Performance will not be sacrificed thanks to multi-core architectures: the JIT runs on separate cores, overlapping with the execution of the application.

9. Dissemination

9.1. Scientific community animation

- Pierre Michaud was a member of the ISPASS 2012 and MuCoCos 2012 program committees
- Isabelle Puaut is a member of the program committees of ECRTS 2013, RTAS 2013 and RTCSA 2013. She was a member of the program committee of DAC 2012, ECRTS 2012, EFTA 2012, LCTES 2012, RTCSA 2012, RTNS 2012, WCET 2012.
- André Seznec was a member of Micro 2012, ISCA 2012 and ICDD2012 program committees. He is a member of the editorial board of the IEEE Micro.
- André Seznec is the Program co-chair of HiPEAC 2013, January 2013 and Program co-chair of PACT 2013 (September 2013).
- Erven Rohou was a member of the program committees of PARMA 2012, EUC 2012 and Computing Frontiers 2012.

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.
- D. Hardy teaches operating systems and compilers in the master degree of computer science of the University of Rennes I. He teaches real-time systems in the BSc degree *Embedded automotive systems*.

- 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

9.3.1. Seminars

- A. Seznec has presented a seminar on "What you will never have wanted to know on branch prediction" at University of Cyprus (April 2012), ETH Zurich (May 2012), Qualcomm (August 2012, Raleigh, North Carolina), Georgia Tech (August 2012) and ARM Sophia Antipolis (October 2012).
- A. Seznec has presented invited talks on "Should we defy the Amdahl's Law" at MATEO 2012 workshop in honor of Mateo Valero (Barcelona, June 2012), and at the "Let's Imagine the Future Workshop" in honor of Jean-Pierre Banâtre (Rennes, November 2012).
- A. Seznec has presented a seminar on "Revisiting value prediction" at Intel Hillsboro in December 2012.
- A. Seznec has presented a seminar on "HAVEGE, HArdware Volatile Entropy Gathering and Expansion: unpredictable random number generation at user level" at the LPMA laboratory (Paris, September 2012).
- I. Puaut has presented a seminar on "WCET estimation for multi-core architectures" at CNES, Toulouse, in December 2012.
- Pierre Michaud has presented a seminar on "Hardware acceleration of sequential loops" at Intel, Boston in December 2011.
- E. Rohou gave a talk at the GDR GPL/5es journées françaises de compilation: "Défis des architectures à venir - Quelle compilation pour demain ?" (Rennes, Irisa/Inria April 2012)
- E. Rohou gave an invited talk at Harvard University: "Compilation Challenges for Upcoming Architectures", (September 2012)
- E. Rohou gave a invited talk at the Workshop on Language Virtual Machines and Multicore Architectures: "Compilation Challenges for Future Architectures" (September 2012, LIP6 Paris)
- E. Rohou gave an invited talk at the Egypt-Japan University of Science and Technology in Alexandria:"Performance of Future Architectures and Compilation Challenges" (May 2012).

9.3.2. Visits

- Pr Ahmed El-Mahdy, from the Egyptian-Japanese University of Science and Technology visited the ALF project for 1 week in October 2012.
- Pr Qureshi from Georgia Tech visited the ALF project-team for 3 weeks in July 2012 in the framework of the DAL ERC project.
- Pr Baniasidi from Victoria university visited the ALF project-team for 1 week in October 2012. This visit was funded by the French embassy in Canada.
- Keisuke Kuroyanagi from University of Tokyo was a master intern in the ALF team from April 2012 to July 2012. He worked on memory scheduling [21].

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.
- I. Puaut is in the steering committee of the RTNS conference.
- I. Puaut is the french representative for the COST action TACLe (Timing Analysis on Code-Level), for which she also chairs the student mobility committee.
- Erven Rohou was 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 was a member of the steering committee of ISCA 2012.
- E. Rohou has been appointed “correspondant scientifique des relations internationales” for Inria Rennes Bretagne Atlantique.

10. Bibliography

Major publications by the team in recent years

[1] F. BODIN, A. SEZNEC. *Skewed associativity improves performance and enhances predictability*, in "IEEE Transactions on Computers", May 1997.

[2] 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.

[3] 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.

[4] 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.

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

[6] 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.

[7] 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.

[8] 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.

[9] 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.

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

[11] 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.

[12] 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.

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

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

[15] 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 Journals

[16] P. MICHAUD. *Demystifying multicore throughput metrics*, in "IEEE Computer Architecture Letters", August 2012, p. ISSN: 1556-6056 [*DOI :* 10.1109/L-CA.2012.25], http://hal.inria.fr/hal-00737044.

[17] N. PRÉMILLIEU, A. SEZNEC. *SYRANT: SYmmetric Resource Allocation on Not-taken and Taken Paths*, in "ACM Transactions on Architecture and Code Optimization (TACO) - HIPEAC Papers", January 2012, vol. 8, n^{o} 4, Article No.: 43 [*DOI :* 10.1145/2086696.2086722], http://hal.inria.fr/inria-00539647.

[18] E. ROHOU, K. WILLIAMS, D. YUSTE. *Vectorization Technology To Improve Interpreter Performance*, in "ACM Transactions on Architecture and Code Optimization", January 2013, http://hal.inria.fr/hal-00747072.

International Conferences with Proceedings

[19] B. BERNA, I. PUAUT. *PDPA: Period Driven Task and Cache Partitioning Algorithm for Mumti-core Systems*, in "20th International Conference on Real-Time and Network Systems (RTNS 2012)", Pont à Mousson, France, November 2012, http://hal.inria.fr/hal-00737591.

[20] D. HARDY, I. SIDERIS, N. LADAS, Y. SAZEIDES. *The performance vulnerability of architectural and non-architectural arrays to permanent faults*, in "MICRO 45", Vancouver, Canada, December 2012, http://hal.inria.fr/hal-00747488.

[21] K. KUROYANAGI, A. SEZNEC. *Service Value Aware Memory Scheduler by Estimating Request Weight and Using per-Thread Traffic Lights*, in "3rd JILP Workshop on Computer Architecture Competitions (JWAC-3): Memory Scheduling Championship (MSC)", Portland, États-Unis, Rajeev Balasubramonian (Univ. of Utah), Niladrish Chatterjee (Univ. of Utah), Zeshan Chishti (Intel), June 2012, http://hal.inria.fr/hal-00746951.

[22] J. LAI, A. SEZNEC. *Break Down GPU Execution Time with an Analytical Method*, in "Rapido '12", Paris, France, ACM (editor), January 2012 [*DOI :* 10.1145/2162131.2162136], http://hal.inria.fr/hal-00764874.

[23] B. LESAGE, I. PUAUT, A. SEZNEC. *PRETI: Partitioned REal-TIme shared cache for mixed-criticality real-time systems.*, in "RTNS - 20th International Conference on Real-Time and Network Systems - 2012", Pont à Mousson, France, ACM, 2012, 10, http://hal.inria.fr/hal-00661687.

[24] J. MARINHO, V. NÉLIS, S. M. PETTERS, I. PUAUT. *An Improved Preemption Delay Upper Bound for Floating Non-Preemptive Region Scheduling*, in "7th IEEE International Symposium on Industrial Embedded Systems (SIES'12)", Karlsruhe, Allemagne, June 2012, http://hal.inria.fr/hal-00737580.

[25] J. MARINHO, V. NÉLIS, S. M. PETTERS, I. PUAUT. *Preemption Delay Analysis for Floating Non-Preemptive Region Scheduling*, in "Design, Automation and Test in Europe 2012", Dresden, Allemagne, March 2012, p. 497-502, http://hal.inria.fr/hal-00737577.

[26] T. MILANEZ, S. COLLANGE, F. MAGNO QUINTÃO PEREIRA, W. MEIRA, R. FERREIRA. *Data and Instruction Uniformity in Minimal Multi-Threading*, in "24th International Symposium on Computer Architecture and High Performance Computing", New-York, NY, États-Unis, October 2012, p. 270-277 [*DOI :* 10.1109/SBAC-PAD.2012.21], http://hal.inria.fr/hal-00755273.

[27] E. ROHOU. *Tiptop: Hardware Performance Counters for the Masses*, in "41st International Conference on Parallel Processing Workshops (ICPPW)", Pittsburgh, PA, États-Unis, September 2012, p. 404-413 [*DOI :* 10.1109/ICPPW.2012.58], http://hal.inria.fr/hal-00747064.

[28] D. SAMPAIO, R. MARTINS, S. COLLANGE, F. MAGNO QUINTÃO PEREIRA. *Divergence Analysis with Affine Constraints*, in "24th International Symposium on Computer Architecture and High Performance Computing", New-York, NY, États-Unis, October 2012, p. 67-74 [*DOI :* 10.1109/SBAC-PAD.2012.22], http://hal.inria.fr/hal-00650235.

[29] R. A. VELASQUEZ, P. MICHAUD, A. SEZNEC. *BADCO: Behavioral Application-Dependent Superscalar Core Model*, in "SAMOS XII: International Conference on Embedded Computer Systems: Architectures, Modeling and Simulation", Samos, Grèce, July 2012, http://hal.inria.fr/hal-00707346.

Research Reports

[30] J. LAI, A. SEZNEC. *Bound the Peak Performance of SGEMM on GPU with software-controlled fast memory*, Inria, April 2012, n^{o} RR-7923, http://hal.inria.fr/hal-00686006.

[31] P. MICHAUD. *Constant-work multiprogram throughput metrics for microarchitecture studies*, Inria, November 2012, n^{o} RR-8150, http://hal.inria.fr/hal-00758195.

[32] A. PERAIS, A. SEZNEC. *Revisiting Value Prediction*, Inria, November 2012, n^{o} RR-8155, 22, http://hal.inria.fr/hal-00758713.

[33] R. A. VELASQUEZ, P. MICHAUD, A. SEZNEC. *Selecting Benchmarks Combinations for the Evaluation of Multicore Throughput*, October 2012, 23, http://hal.inria.fr/hal-00737446.

References in notes

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

[35] N. BRUNIE, S. COLLANGE, G. DIAMOS. *Simultaneous Branch and Warp Interweaving for Sustained GPU Performance*, in "ISCA Conference Proceedings", Portland, OR, États-Unis, June 2012, p. 49 - 60 [*DOI :* 10.1109/ISCA.2012.6237005], http://hal-ens-lyon.archives-ouvertes.fr/ensl-00649650.

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

[37] 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.

[38] 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.

[39] 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.

[40] 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.

[41] 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.

[42] 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.

[43] M. PAOLIERI, E. QUITONES, F. J. CAZORLA, R. I. DAVIS, M. VALERO. *IA3: An Interference Aware Allocation Algorithm for Multicore Hard Real-Time Systems*, in "2011 17th IEEE RealTime and Embedded Technology and Applications Symposium", 2011, p. 280–290, http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=5767118.

[44] 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.

[45] 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.

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

[47] 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.

[48] 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.

[49] 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.

[50] 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.