INRIA

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Team alf

Amdahl's Law is Forever

Rennes - Bretagne-Atlantique

Theme : Architecture and Compiling

Table of contents

1. Team

Research Scientists

André Seznec [Team leader, Research Director Inria, HdR]
Pierre Michaud [Research scientist Inria]
François Bodin [External collaborator, HdR]

Faculty Member

Isabelle Puaut [Professor, University of Rennes 1, HdR]

Technical Staff

Erven Rohou
Sylvain Leroy [till 15/10/10]
Thierry Lafage [till 01/05/10]
David Yuste [from 01/06/10]
Florent Leray

PhD Students

Damien Hardy [MESR allocation till 31/08/10]
Damien Hardy [ATER, University of Rennes 1 from 01/09/10]
Christophe Levointurier [Cifre AQL]
Junjie Lai [Inria Allocation]
Ricardo-Andres Velasquez [Inria Allocation]
Nathanaël Prémillieu [ENS Allocation]
Benjamin Lesage [MESR allocation]
Julien Dusser [ATER, University of Rennes 1 till 31/08/10]

Post-Doctoral Fellow

Kevin Williams [from 01/02/10 till 31/10/2010]

Administrative Assistant

Maryse Fouché [TR Inria]

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; the 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 1000's 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, in particular real-time systems, the effective use of multicore chips will depend on the ability of the software and hardware providers 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 intend 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 intend to propose to predict/guarantee the response time of many-core architectures.

High performance on single process and sequential codes is one of the key issues for enabling overall high performance on a 1000's cores system. Therefore we anticipate that future manycore architectures will feature heterogeneous design with many simple cores and a few complex cores. Therefore the research in the ALF project focuses on refining the microarchitecture to achieve high performance on single 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+ 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 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 times. 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 the multicores.

Our research is partially supported by industry (Intel, IBM, STMicroelectronics). We also participate in several institutionally funded projects (NoE HIPEAC2, IP Fet project SARC, ANR funded PetaQCD, "Pôle de compétitivité" funded Serenitec, Brittany Region Ecaras).

2.2. Highlights

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

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's cores chip will become feasible before 2020. On the other hand, while traditional parallel application domains, e.g. supercomputing and transaction servers, are taking benefit from the introduction of multicores, there are very few new parallel applications that have emerged during the last years.

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

3.2. The context

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

For almost 30 years after the introduction of the first microprocessor, the processor industry has been driven by the Moore's law till 2002, delivering performance doubling 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 return, 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. It will probably be technologically feasible to double the number of cores every two years for the next 12 or 15 years, thus potentially leading to more than 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

Up to 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 either a functionality improvement or a performance improvement on current applications: the end-user will then realize that he/she needs more effective performance rather than more cores. He/she 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 needed; e.g. the Intel Montecito (IA64) features an adjunct (simple) core which unique mission is to manage the power and temperature on two cores. Processor industry will require more and more heroic efforts on this power and temperature management policy to maintain its current performance scaling path. Therefore the power and temperature walls might slow the race towards 100's and 1000's cores unless 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 demand. Up to three levels of caches are implemented for uniprocessors. For multi- and many-cores the problems are even worse. The memory hierarchy must be replicated for the processors, 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 the organization of the processors (flat ring? flat grid? hierarchical ring or grid?).

Therefore, organizing and dimensioning the memory hierarchy will be a major challenge for the computer architects. The successfull architecture will also be determined by the abilitity 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 (e.g. 3D memory stacking) may demand to revisit the memory hierarchy. E.g. 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 forms may coexist in an application. For instance, one can express some parallelism with different tasks achieving different functionalities. Then, in 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 (aka 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, the parallel application will have to be executed in concurrence with other parallel applications.

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, while 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 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. Developing parallel applications is necessitating programmers educated in parallel programming. 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 overwhelming majority of software developments. Therefore, sequential programming style will continue to be dominant for 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 very performant 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 which will not be able to tune the application environment to optimize performance. They will be executed in concurrence 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 [42] pointed out that, if one accelerates only a part of an application, the execution time cannot be reduced below the execution time of the residual part of the application. Unfortunately, even very parallel 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-cores architecture for highly parallel applications, we believe that the success of the future 1000's cores 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 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 [43]). Such simulations are CPU consuming and simulations cannot be run on a complete application. Sampling representative slices of the application was proposed [6] and popularized by the Simpoint [55] 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 [45]), WCET estimation of more complex uniprocessor systems is still a difficult problem. Accurate evaluation of the WCET of an algorithm on a complex uniprocessor system is a difficult problem. Accurately modelling data cache behavior [5] 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) [53].

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 [59].

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 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 thread 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/architecture issue for the design of future processor chips.

3.6.1.1. Enhancing complex core microarchitecture

Research on wide issue superscalar processor was merely stopped around 2002 due to limited performance return and 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. Data dependencies may strongly limit performance, we will revisit data prediction [14], [13]. 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 [44]. 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 happens 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 [61], but the most promising solutions integrate hardware support for software thread-level speculation [57].

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 performing cores for speculative threads. Moreover the availability of many simple cores may lead to using at the same time helper threads and thread level speculation.

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* [56][8].

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 a 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 migrations will be key techniques to deal with Amdahl's law in future many-core processors.

3.6.2. Processor simulation research

Architecture studies and particularly 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's cores ALF base 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 [48] 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 simulations 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 [49] can also be used for decreasing the time required to explore the large space of microarchitecture parameter values. We will explore this technique 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 moved apart 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, but with 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 code generation compilers strategies. Speculation can be used to deal with necessary but missing information at compile-time. Finally, dynamic techniques can select or generate at run-time the most efficient code adapted to the execution context and available hardware resources.

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

3.6.3.2. Portability of applications and performance through virtualization

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

Processor virtualization addresses the problem of portability of functionalities. Applications are not compiled to the final native code but to a target independent format. This is the purpose of languages such as Java and .NET. Bytecode formats are often *a priori* perceived as inappropriate for performance intensive applications and for embedded systems. However, it was shown that compiling a C or C++ program to a bytecode format produces a code size similar to dense instruction sets [4]. Moreover, this bytecode representation can be compiled to native code with performance similar to static compilation [3]. Therefore processor virtualization for high performance, i.e., for languages like C or C++, provides significant advantages: 1) it simplifies software engineering with fewer tools to maintain and upgrade; 2) it allows better code readability with eliminating specific targets #ifdef and allows easier code maintenance 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 binary. 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 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 times. The amount of safety 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, architecture will feature an execution mode where a given processor will be guaranteed to access a fixed part of the shared resource (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) [53]. 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 [51] will be explored to obtain as safe as possible WCET estimates.

Providing performance guarantees for parallel applications executing 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 acces to main memory. Current pure analytical methods are too pessimistic at capturing interferences between cores [60], therefore hardware-based or compiler methods such as [58] 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 many cores architectures.

4. Application Domains

4.1. Application Domains

The ALF team is working on the foundation 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 and **HAVEGE**, an unpredictable random number generator, four softwares developed by the team.

5.2. ATMI

Participant: Pierre Michaud.

Contact : Pierre Michaud

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

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

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

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

5.3. STiMuL

Participant: Pierre Michaud.

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

Some recent research has started investigating the microarchitecture 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 [39]. The number and characteristics of layers can be defined by the user. The boundary conditions too can be defined by the user. In particular, STiMuL can be used along with thermal imaging to obtain the power density inside an integrated circuit. This power density could be used for instance in a dynamic simulation oriented temperature modeling such as ATMI.

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

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

5.4. ATC

Participant: Pierre Michaud.

Contact : Pierre Michaud

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

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

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

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

5.5. HAVEGE

Participant: André Seznec.

Contact : André Seznec

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

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

6. New Results

6.1. Processor Architecture

Participants: Julien Dusser, Pierre Michaud, Nathanaël Prémillieu, André Seznec.

Our research in computer architecture covers memory hierarchy, branch prediction, superscalar implementation, as well as SMT and multicore issues. We also address power consumption and temperature management that have become major concerns for high performance processor design.

6.1.1. Null blocks management on the memory hierarchy

Participants: Julien Dusser, André Seznec.

It has been observed that some applications manipulate large amounts of null data. Moreover these zero data often exhibit high spatial locality. On some applications more than 20% of the data accesses concern null data blocks.

We propose to leverage this property in the whole memory hierarchy [16], [26]. We have first proposed the Zero-Content Augmented cache, the ZCA cache. A ZCA cache consists of a conventional cache augmented with a specialized cache for memorizing null blocks, the Zero-Content cache or ZC cache. In the ZC cache, the data block is represented by its address tag and a validity bit. Moreover, as null blocks generally exhibit high spatial locality, several null blocks can be associated with a single address tag in the ZC cache. For instance, a ZC cache mapping 32MB of zero 64-byte lines uses less than 80KB of storage. Decompression of a null block is very simple, therefore read access time on the ZCA cache is in the same range as on a conventional cache. On applications manipulating large amount of null data blocks, such a ZC cache allows to significantly reduce the miss rate and memory traffic, and therefore to increase performance for a small hardware overhead.

To reduce the pressure on main memory, we have proposed a hardware compressed memory that only targets null data blocks, the decoupled zero-compressed memory [26]. Borrowing some ideas from the decoupled sectored cache [15], the decoupled zero-compressed memory, or DZC memory, manages the main memory as a decoupled sectored set-associative cache where null blocks are only represented by a validity bit. Our experiments show that for many applications, the DZC memory allows to artificially enlarge the main memory, i.e. it reduces the effective physical memory size needed to accommodate the working set of an application without excessive page swapping. Moreover, the DZC memory can be associated with a ZCA cache to manage null blocks across the whole memory hierarchy. On some applications, such a management significantly decreases the memory traffic and therefore can significantly improve performance.

6.1.2. Last-level cache replacement policies

Participant: Pierre Michaud.

In current computers, main memory access latency can be several hundreds of clock cycles. Hence memory accesses are very costly in terms of performance as well as in terms of energy consumption. On-chip caches are used to limit performance and energy penalty. In that respect, the last on-chip cache level is particularly important. When the application working-set does not fit entirely in the last-level cache (LLC), the cache replacement policy may have a dramatic impact on the LLC efficacy. Recently, some researchers have proposed a very cost-effective replacement policy, called DIP, that outperforms the least-recently-used (LRU) policy [54]. DIP combines two different policies and uses a single counter for dynamically finding which policy is best.

We have first proposed a simple solution for generalizing this approach to more than two policies. Our new policy-selection method is based on a single counter per policy and updating the counter values in such a way that the sum of all counter values remains always null. This latter feature makes our selection method simple and effective.

For shared LLC in multicores, we have proposed a heuristic for improving the fairness of the replacement policy. Our heuristic is based on the observation that processes experiencing the smallest number of misses per second are likely to be impaired unfairly by other processes under single-core replacement policies. We have shown that this unfair treatment can be repaired to some extent by slowing down the insertion rate of the other processes.

We have implemented these methods in two cost-effective replacement policies called 3P and 4P, which won the second and third place respectively in the single-core and multi-core tracks of the recent First JILP Workshop on Computer Architecture Competition [28].

6.1.3. Emerging memory technologies

Participant: André Seznec.

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

We have proposed the first design of a secure PCM-based main memory that would by construction survive to overwrite attacks [37], [21]. In order to prevent a malicious user from overwriting some memory cells, the physical memory address (PA) manipulated by the computer system is not the same as the PCM memory address (PCMA). PCMA is made invisible from the rest of the computer system. The PCM memory controller is in charge of the PA-to-PCMA translation. Hiding PCMA alone cannot not prevent a malicious user to overwrite a PCM memory word. Therefore in the secure PCM-based main memory, PA-to-PCMA translation is continuously modified through a random process, thus preventing a malicious user from destructing some PCM memory words. PCM address invisibility and continuous random PA-to-PCMA translation ensures security against an overwrite attack as well it ensures a practical write endurance close to the theoretical maximum. The hardware overhead needed to ensure this security in the PCM controller includes a random number generator and a medium large address translation table.

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

6.1.4. 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 [40].

6.1.5. *Confidence Estimation for the TAGE predictor*

Participant: André Seznec.

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

The TAGE predictor presented in 2006 [14] is so far considered as the state-of-the-art conditional branch predictor. We have shown that very accurate confidence estimations can be done for the branch predictions realized by the TAGE predictor by simply observing the outputs of the predictor tables. Many confidence estimators proposed in the literature only discriminate between high confidence predictions and low confidence estimations. It has been recently pointed out that a more selective confidence discrimination could be useful. The observation of the outputs of the predictor tables is sufficient to grade the confidence in the branch predictions with a very good granularity. Moreover a slight modification of the predictor automaton allows to discriminate the prediction in three classes, low-confidence (with a misprediction rate in the 30 % range), medium confidence (with a misprediction rate in 8-12% range) and high confidence (with a misprediction rate lower than 1 %) [41].

6.1.6. *Sequential accelerators in future general-purpose manycore processors*

Participants: Pierre Michaud, André Seznec.

The number of transistors that can be put on a given silicon area doubles on every technology generation. Consequently, the number of on-chip cores increases quickly, making it possible to build general-purpose processors with hundreds of cores in a near future. However, though having a large number of cores is beneficial for speeding up parallel code sections, it is also important to speed up sequential execution. We argue that it will be possible and desirable to dedicate a large fraction of the chip area and power to high sequential performance.

Current processor design styles are restrained by the implicit constraint that a processor core should be able to run continuously; therefore power hungry techniques that would allow very high clock frequencies are not used. The “sequential accelerator” [30] we propose removes the constraint of continuous functioning. The sequential accelerator consists of several cores designed for ultimate instantaneous performance. Those cores are large and power-hungry, they cannot run continuously (thermal constraint) and cannot be active simultaneously (power constraint). A single core is active at any time, inactive cores are power-gated. The execution is migrated periodically to a new core to spread the heat generation uniformly over the whole accelerator area, which solves the temperature issue. The “sequential accelerator” will be a viable solution only if the performance penalty due to migrations can be tolerated. Migration-induced cache misses may incur a significant performance loss. We propose some solutions to alleviate this problem. We also propose a

migration method, using integrated thermal sensors, such that the migration interval is variable and depends on the ambient temperature. The migration penalty can be kept negligible as long as the ambient temperature is maintained below a threshold.

This research is done in cooperation with Pr Yannakis Sazeides from University of Cyprus.

6.2. Around processor virtualization

Participants: François Bodin, Christophe Levointurier, Sylvain Leroy, Florent Leray, Erven Rohou, Thierry Lafage, Kevin Williams, David Yuste, André Seznec.

The usage of the bytecote-based languages such as Java has been generalized in the past few years. Applications are now very large and are deployed on many different platforms, since they are highly portable.

Ensuring code quality maintenance and code security on those applications is challenging. To address these issues, we are defining a refactoring platform for Java. At the same time, Java has popularized the distribution of software through bytecodes. Functional portability is the main argument for such a usage of bytecodes. With the new diversity of multicore platforms, functional, but also performance portability will become the major issue in the next 10 years. Therefore we have initiated a research effort to efficiently compile towards bytecodes.

6.2.1. Analysis and transformation of Java codes

Participants: François Bodin, Sylvain Leroy, Florent Leray, Christophe Levointurier.

All along its lifetime, an application software evolves. Development rules that were initially defined are often progressively ignored or forgotten. Development rules may even evolve. Therefore one generally observes deterioration of the quality of the code. In particular, ignoring some design rules may affect the code security or its overall performance. To avoid this deterioration, automatic refactoring has been proposed [46]. Code is automatically transformed through enforcing development rules.

Our objective is to analyze Java web applications security through an automatic process. Invalid pointers, code injections or possible data buffer overflows are commonly used in successful attacks of web applications.

Within the Serenitec project, we are developing a framework for automatic refactoring of Java codes. Our framework has been first directed towards the audit of code rules. It can analyze large applications featuring a million java code lines or more. As most of the information needed to discover issues in Web applications are not contained in the core Java code, we have designed new Web applications analysis techniques that consider the complete set of files being used to construct an application (Java, xml, ...).

6.2.2. Split vectorization

Participants: Erven Rohou, Thierry Lafage, David Yuste, Kevin Williams, André Seznec.

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

We showed in [32] that speculatively vectorized bytecode is: (1) robust — the approach is general enough to allow execution, both when using SIMD capabilities and also in the absence of SIMD extensions, or when using an unmodified, non-vectorizing JIT compiler; (2) risk-free — the penalty of running vectorized bytecode without SIMD support is kept at a minimum; (3) efficient — the improvement of running vectorized bytecode with SIMD support is maximized.

We currently investigate how vectorized bytecode performs in interpreted execution environments.

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

6.2.3. Portability of Legacy Applications

Participant: Erven Rohou.

Embedded multiprocessors have always been heterogeneous, driven by the power-efficiency and compute-density of hardware specialization. We aim to achieve portability and sustained performance of complete applications, leveraging diverse programmable cores. We combined instruction-set virtualization with just-in-time compilation, compiling C, C++ and managed languages to a target-independent intermediate language, maximizing the information flow between compilation steps in a split optimization process [24]. We showed that CLI is a convenient program representation, and we presented various techniques to compile the C language to CLI [34]. We also proposed to combine processor virtualization with component-based software engineering for the C language [35].

Part of this research is in collaboration with STMicroelectronics.

6.2.4. Application of Split Compilation to Code Specialization

Participant: Erven Rohou.

Code specialization is a typical optimization that takes advantage of runtime information to achieve good results. The optimizer, though, must decide when the extra work is worth the effort. We plan to statically pre-compute the predicates that will impact the performance, and to embed them in the bytecode as annotations, in order to simplify the decision process of the JIT compiler. The optimizer will be able to make a faster and better informed decision.

This research is done in collaboration with Prof. Stefano Crespi Reghizzi from Politecnico di Milano, and Dr. Simone Campanoni from Harvard University.

6.2.5. The Pitfalls of Benchmarking with Applications

Participants: Erven Rohou, Thierry Lafage.

Application benchmarking is a widely trusted method of performance evaluation. Compiler developers rely on them to assess the correctness and performance of their optimizations; computer vendors use them to compare their respective machines; processor architects run them to tune innovative features, and to a lesser extent to validate their correctness. Benchmarks must reflect actual workloads of interest, and return a synthetic measure of "performance". Often, benchmarks are simply a collection of real-world applications run as black boxes. We identified a number of pitfalls that derive from using applications as benchmarks, and we illustrated them with a popular, freely available, benchmark suite [33]. In particular, we advocate the fact that correctness should be defined by an expert of the application domain, and the test should be integrated in the benchmark.

6.2.6. Significance of Eliminating Writebacks from Dead Blocks in the Execution Stack

Participants: Erven Rohou, André Seznec.

This work is concerned with a technique that aims to eliminate performing writebacks of dead memory blocks in the execution stack. On a cache block replacement, to ensure correction execution, a modified block must be written back to the lower levels of the memory hierarchy .This action, however, is useless if the replaced block is dead, i.e. it will never be referenced again or it will get reallocated before being re-referenced.

In this study we explore the performance potential of eliminating writebacks from dead blocks in the stack for single and multi-threaded programs as well as multi-programmed workloads. It will also explore hardware, software and hybrid (hw+sw) mechanisms that can detect and eliminate writebacks of dead blocks in the stack. The performance of the various schemes will be evaluated to determine the performance-cost trade-offs they present.

This research is done in collaboration with Prof. Yanos Sazeides, from the University of Cyprus.

6.3. Understanding performance issues

6.3.1. Black-box modeling of superscalar cores

Participants: Ricardo-Andres Velasquez, Pierre Michaud, André Seznec.

In recent years, research in microarchitecture has shifted from single-core to multi-core processors. Cycle-accurate models for manycore processors featuring hundreds or even thousands of cores are out of reach for realistic workloads. Approximate simulation methods are needed to simulate the impact of resource sharing between cores, where the resource can be caches, on-chip network, memory bus, power, temperature, etc.

A method for black-box modeling was recently proposed by Lee et al. [50]. To enable the simulation of a large spectrum of memory hierarchies, the method assumes that the execution time of an instruction sequence generating no external memory reference is constant and that a sequence can only depend on the last external memory reference. This study exhibits simulation time reduced by a factor 30, but at the cost of a up to 20% error range over a standard superscalar simulator using unrealistic assumptions (perfect branch prediction, very simple memory hierarchy modeling, no prefetching). Our own experiments for reproducing Lee et al.'s method using the state-of-art cycle accurate simulator Zesto [52] as a baseline confirm the potentially high simulation rates of black-box models, but also points out very high error ranges on simulated performance.

This has led us to explore a new approach to better reflect the behavior of superscalar cores. We consider the problem of black-box modeling of cores as an interpolation problem. The functioning of a superscalar core is modeled through a graph whose complete structure is hidden. "Visible" nodes are the memory instructions generating an external memory access. As in [50], we assume that the execution time of an instruction sequence generating no external memory reference is constant, but we also look for the possible dependences of each sequence from previous sequences. Through executing two different simulations with extreme memory latencies, we determine dependency adges between sequences.

6.3.2. Architecture for Lattice QCD

Participants: Junjie Lai, André Seznec.

Simulation of Lattice QCD is a challenging computational problem that requires very high performance exceeding sustained Petaflops/s. In the framework of the ANR Cosinus PetaQCD project [23], we are modeling the demands of this application on the memory system and synchronization mechanisms.

The objective is to obtain a first order comparison of different design options for a LQCD machine based on off-the-shelf multi-cores or multi-cores+accelerators designs, therefore guiding the dimensioning of a dedicated machine for LQCD and/or the precise algorithm implementation on a given platform. The methodology should be able to be adapted to the study of other massively parallel applications to understand/predict their performance behavior. It should also be useful in early multi-core design phases to help to decide on internal die organization such as number of cores vs cache size, hierarchical organization, etc.

6.4. WCET estimation

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

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 task 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.

New results are related to hardware-level analysis (static analysis based on timing models) for multicore platforms with shared caches. A first study aiming at providing predictability guarantees for systems using Just-In-Time (JIT) compilation has also been initiated.

6.4.1. Timing analysis for multicore platforms with shared caches

Participants: Damien Hardy, Benjamin Lesage, Isabelle Puaut.

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

Regarding the worst-case timing analysis of cache hierarchies, we have demonstrated in previous work [47] that already published analyses of cache hierarchies are not safe in all situations, and have proposed a new safe timing analysis method for hierarchies of non-inclusive instruction cache hierarchies. In 2010, we have generalized our previous work to support different cache hierarchy management policies between cache levels: non-inclusive, inclusive and exclusive cache hierarchies. Moreover, our analysis now supports cache hierarchies with different replacement policies: Least Recently Used (LRU), Pseudo-LRU, First-In First-Out (FIFO), Most Recently Used (MRU) and Random. Experimental results, detailed in [19] and [17] show that the method is precise in many cases (non-inclusive and exclusive cache hierarchies with the LRU replacement policy) and has a reasonable computation time. Nevertheless, we have observed that considering inclusion enforcement mechanisms and non-LRU replacement policies increases the analysis pessimism. Moreover, these two sources of pessimism are cumulative, which results in some cases in a significant overestimation. Although inclusive cache hierarchies with non-LRU replacement policies can be analyzed statically, the cache hierarchies to be favored to obtain the tighter WCET estimates are hierarchies of non-inclusive or exclusive caches with the LRU replacement policy.

Another issue to be addressed when predicting the WCET of tasks executing on multicore processors is the one of shared cache levels. In [27] a safe method to estimate conflicts stemming from data cache sharing is presented, together with the integration of conflicts in data cache analyses. The other, and foremost, contribution of the paper is the introduction of bypass heuristics to reduce these conflicts, allowing for reuse to be more easily captured by shared caches analyses. Results show that plainly considering data cache sharing related conflicts, without any mechanism to reduce their number, is not a scalable approach as it tends to result in little to no reuse captured in shared cache levels. The bypass heuristics proposed in this document were shown to be an interesting solution to this issue.

6.4.2. Reconciling Predictability and Just-In-Time Compilation

Participants: Isabelle Puaut, Erven Rohou.

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

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

7. Contracts and Grants with Industry

7.1. Research grant from Intel

Participants: Nathanaël Prémillieu, Julien Dusser, André Seznec.

The researches on control independance and confidence estimation (cf. 6.1.1), and on branch prediction are partially supported by the Intel company through a research grant.

7.2. IBM Faculty award

Participant: André Seznec.

The researches on Phase Change Memory and security is partially funded by an IBM faculty award attributed to André Seznec.

7.3. Nano2012 Mediacom

Participants: Erven Rohou, Thierry Lafage, David Yuste.

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

8. Other Grants and Activities

8.1. NoEs

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

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

8.2. IP-Fet European project Sarc

Participants: Julien Dusser, Pierre Michaud, André Seznec.

SARC is an integrated IP-FET project concerned with long term research in advanced computer architecture http://www.sarc-ip.org/. It focuses on a systematic scalable approach to systems design ranging from small energy critical embedded systems right up to large scale networked data servers.

The ALF team is involved in the microarchitecture research, including temperature management and memory hierarchy management.

8.3. Britanny region fellowship

Participants: Ricardo-Andres Velasquez, Pierre Michaud, André Seznec.

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

8.4. Britanny region ECARAS

Participants: Florent Leray, Sylvain Leroy, François Bodin.

The project Ecaras has been set-up to reach a software maturity and industrial quality of softwares developed within the ALF research team under the Serenitec Project. This project benefited from funds to hire a engineer during one year for a mission of test-cases with industrial partners.

8.5. Serenitec: SEcurity analysis and Refactoring ENvironment for Internet TEChnology

Participants: François Bodin, Christophe Levointurier, Sylvain Leroy.

Serenitec aims at analyzing and improving security of Java Web applications. To achieve its goals, the project mixes a set of techniques from static program analysis, case based reasoning and refactoring techniques. Security analysis are based on the work of the Open Web Application Security Project. To validate the techniques, large web analysis will be used (500 kloc to 1 Mloc).

In this project, ALF studies basic analysis and refactoring techniques for Java codes. Serenitec is a project of the Pôle de compétitivité Images et Réseaux. It is funded by the Region Bretagne and Rennes Métropole. Partners of this project are Silicom-AQL, Caps Entreprise and Irisa/INRIA (prime).

8.6. PetaQCD

Participants: Junjie Lai, André Seznec.

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

9. Dissemination

9.1. Scientific community animation

- François Bodin was a member of the program committees of PPoPP 2010, MULTIPROG 2010, SMART'10, SBAC-PAD 2010, MuCoCoS 2010. He is a member of CGO 2011, HPCS 2011,
- Pierre Michaud was a member of the program committees of MuCoCoS 2010, e-Energy 2010, and IFMT 2010.
- Isabelle Puaut was a member of program committees of RTSS 2010, ECRTS10, RTAS 2010, JTRES 2010, RTNS 2010, real-time track of ETFA 2010, WCET 2010, RTCSA 2010. She is a member of program comittees of ECRTS 2011, RTAS 2011, DATE 2011.
- Erven Rohou was a member of the program committee of the 2PARMA Workshop 2010. He is a member of the program committee of the PARMA Workshop 2011, Como, Italy.
- André Seznec was a member of HPCA 2010, Computing Frontiers 2010, IPDPS 2010, CASES 2010, ICCD 2010, MULTIPROG'10, CMP-MSI'10, Micro Top Picks 2010 program comittees. He is a member of HPCA 2011, ISCA 2011, MULTIPROG' 11 and Micro Top Picks 2011 program committees. He is a member of the editorial board of the HiPEAC Transactions (Transactions on High-Performance Embedded Architectures and Compilers).
- Erven Rohou was the Co-Chair of the Second International Forum on Next Generation Multi-core/Manycore Technologies (IFMT'10), colocated with ISCA2010 in Saint-Malo, France. He is the Co-Chair of the 3rd International Workshop on GCC Research Opportunities (GROW 2011), colocated with CGO2011 in Chamonix, France.
- The ALF team has been organizing the 37th ISCA confence at Saint-Malo in June 2010. A. Seznec was the general chair. I. Puaut was the finance chair. P. Michaud was the local chair. E. Rohou was the web chair.

9.2. Animation of the scientific community

F. Bodin is director of IRISA. IRISA (Institut de Recherche en Informatique et Systèmes Aléatoires) is a joint research unit (UMR 6074), including CNRS, University of Rennes 1, INSA Rennes and ENS Cachan (Brittany site). IRISA laboratory is associated with INRIA.

9.3. 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.
- E. Rohou taught classes and labs of Computing Systems at École Polytechnique (INF422)
- I. Puaut teaches operating systems and real-time systems in the master degree of computer science of the University of Rennes I and at Ecole Supérieure d'ingénieurs de Rennes.
- Pierre Michaud and André Seznec are teaching computer architecture at the engineering degree in computer science at Ecole Supérieure d'ingénieurs de Rennes.
- I. Puaut is co-responsible of the Master of Research in computer science in Britanny (administered jointly by University of Rennes I, University of Bretagne Sud, University of Bretagne Occidentale, INSA de Rennes, ENS Cachan antenne de Bretagne, ENIB, Supélec, Telecom-Bretagne).

9.4. Workshops, seminars, invitations, visitors

- A. Seznec has presented seminars on "Null blocks in the memory hierarchy" and "Storage Free Confidence Estimators for the TAGE predictors" at Intel Hudson, Massachusetts and Intel Hillsboro, Oregon in October 2010.
- Pierre Michaud was invited by the HiPEAC cluster on Design and Simulation to give a presentation entitled "Understanding the numbers that you get from your temperature model".

9.5. Miscelleanous

- I. Puaut is a member of the advisory board of the fundation 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.
- A. Seznec is an elected member of the scientific committee of INRIA.
- A. Seznec has been nominated by ACM for 3 years 2011-2013 on the selection committee for the ACM-IEEE Eckert-Mauchly award.
- A. Seznec is a member of the steering committee of ISCA 2011 and HiPEAC 2010.

10. Bibliography

Major publications by the team in recent years

[1] F. BELLETTI, S. F. SCHIFANO, R. TRIPICCIONE, F. BODIN, P. BOUCAUD, J. MICHELI, O. PENE, N. CABIBBO, S. DE LUCA, A. LONARDO, D. ROSSETTI, P. VICINI, M. LUKYANOV, L. MORIN, N. PASCHEDAG, H. SIMMA, V. MORENAS, D. PLEITER, F. RAPUANO. *Computing for LQCD: ApeNEXT*, in "Computing in Science and Engineering", 2006, vol. 8, n^{o} 1, p. 18–29, http://dx.doi.org/10.1109/MCSE.2006.4.

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

[3] M. CORNERO, R. COSTA, R. FERNÁNDEZ PASCUAL, A. ORNSTEIN, E. ROHOU. *An Experimental Environment Validating the Suitability of CLI as an Effective Deployment Format for Embedded Systems*, in "Conference on HiPEAC", Göteborg, Sweden, P. STENSTRÖM, M. DUBOIS, M. KATEVENIS, R. GUPTA, T. UNGERER (editors), Springer, January 2008, p. 130–144.

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

[5] 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.

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

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

[8] 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.

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

[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

Doctoral Dissertations and Habilitation Theses

[16] J. DUSSER. *Blocs nuls dans la hiérarchie mémoire*, Université de Rennes 1, December 2010.

[17] D. HARDY. *Analyse pire cas pour processeurs multi-cœurs disposant de caches partagés*, Université de Rennes 1, December 2010.

Articles in International Peer-Reviewed Journal

[18] G. FURSIN, Y. KASHNIKOV, A. W. MEMON, Z. CHAMSKI, O. TEMAM, M. NAMOLARU, E. YOM-TOV, B. MENDELSON, A. ZAKS, E. COURTOIS, F. BODIN, P. BARNARD, E. ASHTON, E. BONILLA, J. THOMSON, C. WILLIAMS, M. O'BOYLE. *Milepost GCC: machine learning enabled self-tuning compiler*, in "International Journal of Parallel Programming", 2011, to appear.

[19] D. HARDY, I. PUAUT. *WCET analysis of instruction cache hierarchies*, in "Journal of Systems Architecture", 2010, vol. In Press, Corrected Proof [*DOI :* DOI: 10.1016/J.SYSARC.2010.08.007], http://www.sciencedirect.com/science/article/B6V1F-50X3V23-1/2/fae18562a7aef225a95fc7310b5237df.

[20] H. MUNK, E. ROHOU, ET AL.. *ACOTES Project: Advanced Compiler Technologies for Embedded Streaming*, in "International Journal of Parallel Programming", April 2010, p. 1–54.

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

[22] M. WANG, F. BODIN. *Compiler-directed memory management for heterogeneous MPSoCs*, in "Journal of Systems Architecture", 2010, vol. In Press, Corrected Proof [*DOI :* DOI: 10.1016/J.SYSARC.2010.10.008], http://www.sciencedirect.com/science/article/B6V1F-51B1WVF-1/2/8ea418a31a24acdd5b558f8754252cbd.

International Peer-Reviewed Conference/Proceedings

[23] J. ANGLES D'AURIAC, D. BARTHOU, D. BECIREVIC, R. BILHAUT, F. BODIN, P. BOUCAUD, O. BRAND-FOISSAC, J. CARBONELL, C. EISENBEIS, P. GALLARD, G. GROSDIDIER, P. GUICHON, P. HONORE, G. LE MEUR, P. PENE, L. RILLING, P. ROUDEAU, A. SEZNEC, A. STOCCHI, F. TOUZE. *Towards the petaflop for Lattice QCD simulations the PetaQCD project*, in "17th International Conference on Computing in High Energy and Nuclear Physics (CHEP'09)", Prague, 2010, vol. 219, 052021, http://hal.inria.fr/in2p3-00380246.

[24] A. COHEN, E. ROHOU. *Processor Virtualization and Split Compilation for Heterogeneous Multicore Embedded Systems*, in "47th Annual Design Automation Conference", États-Unis Anaheim, CA, 2010, http://hal.inria.fr/inria-00472274.

[25] J. DUSSER, A. SEZNEC. *Decoupled Zero-Compressed Memory*, in "Workshop on Chip Multiprocessor Memory Systems and Interconnects (CMP-MSI)", Bangalore, India, Jan 2010, 11 p., http://hal.inria.fr/inria-00468354.

[26] J. DUSSER, A. SEZNEC. *Decoupled Zero-Compressed Memory*, in "International Conference on High-Performance and Embedded Architectures and Compilers", Heraklion, Greece, ACM, 01 2011, to appear, http://hal.inria.fr/inria-00529332/en/.

[27] B. LESAGE, D. HARDY, I. PUAUT. *Shared Data Caches Conflicts Reduction for WCET Computation in Multi-Core Architectures.*, in "18th International Conference on Real-Time and Network Systems", Toulouse, France, Nov 2010, 2283, http://hal.inria.fr/inria-00531214.

[28] P. MICHAUD. *The 3P and 4P cache replacement policies*, in "JWAC 2010 - 1st JILP Worshop on Computer Architecture Competitions: cache replacement Championship", France Saint Malo, Jun 2010, http://hal.inria.fr/inria-00492968.

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

[30] 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.

[31] D. NUZMAN, S. DYSHEL, E. ROHOU, I. ROSEN, K. WILLIAMS, D. YUSTE, A. COHEN, A. ZAKS. *Auto-Vectorize Once, Run Everywhere*, in "International Symposium on Code Generation and Optimization", France Chamonix, April 2011, to appear.

[32] E. ROHOU, S. DYSHEL, D. NUZMAN, I. ROSEN, K. WILLIAMS, A. COHEN, A. ZAKS. *Speculatively Vectorized Bytecode*, in "Sixth HiPEAC Conference", Heraklion, Greece, January 2011, to appear, http://hal.inria.fr/inria-00525139.

[33] E. ROHOU, T. LAFAGE. *The Pitfalls of Benchmarking with Applications*, in "MoBS 2010 - Sixth Annual Workshop on Modeling, Benchmarking and Simulation", France Saint Malo, Jun 2010, http://hal.inria.fr/inria-00492997.

[34] E. ROHOU, A. ORNSTEIN, M. CORNERO. *CLI-Based Compilation Flows for the C Language*, in "International Conference on Embedded Computer Systems: Architectures, Modeling and Simulation", IEEE, Jul 2010, http://hal.inria.fr/inria-00505640.

[35] E. ROHOU, A. ORNSTEIN, A. E. ÖZCAN, M. CORNERO. *Combining Processor Virtualization and Component-Based Engineering in C for Many-Core Heterogeneous Embedded MPSoCs*, in "Second Workshop on Programming Models for Emerging Architectures (PMEA)", Vienna Austria, September 2010.

[36] E. ROHOU. *Portable and Efficient Auto-vectorized Bytecode: a Look at the Interaction between Static and JIT Compilers*, in "2nd International Workshop on GCC Research Opportunities", Italie Pisa, Jan 2010, http://hal.inria.fr/inria-00468015.

[37] A. SEZNEC. *Towards Phase Change Memory as a Secure Main Memory*, in "Workshop on the Use of Emerging Storage and Memory Technologies (WEST 2010)", Inde Bangalore, Jan 2010, http://hal.inria.fr/inria-00468878.

Scientific Books (or Scientific Book chapters)

[38] *Guest editorial: special issue of the Euromicro Conference on Real-Time Systems (ECRTS 2009)*, Springer Netherlands, 2010, vol. 46, p. 1-2, 10.1007/s11241-010-9099-0, http://dx.doi.org/10.1007/s11241-010-9099-0.

Research Reports

[39] 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.

[40] N. PRÉMILLIEU, A. SEZNEC. *SYRANT: SYmmetric Resource Allocation on Not-taken and Taken Paths*, INRIA, 11 2010, n^{o} RR-7463, http://hal.inria.fr/inria-00539647/en/.

[41] A. SEZNEC. *Storage Free Confidence Estimation for the TAGE branch predictor*, INRIA, Aug 2010, RR-7371, http://hal.inria.fr/inria-00512130.

References in notes

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

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

[44] 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.

[45] 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.

[46] M. FOWLER, K. BECK, J. BRANT, W. OPDYKE, D. ROBERTS. *Refactoring: improving the design of existing code*, Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1999.

[47] D. HARDY, I. PUAUT. *Predictable code and data paging for real-time systems*, in "Proc. of the 20th Euromicro Conference on Real-Time Systems", Prague, Czech Republic, July 2008, http://www.irisa.fr/caps/publications/pdfs/ECRTS08.pdf.

[48] 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.

[49] 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.

[50] K. LEE, S. EVANS, S. CHO. *Accurately approximating superscalar processor performance from traces*, in "Proceedings of the IEEE International Symposium on Performance Analysis of Systems and Software", 2009.

[51] 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.

[52] G. LOH, S. SUBRAMANIAM, Y. XIE. *Zesto: a cycle-level simulator for highly detailed microarchitecture exploration*, in "Proceedings of the IEEE International Symposium on performance analysis of systems and software", 2009.

[53] 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.

[54] M. QURESHI, A. JALEEL, Y. PATT, S.C.JR. STEELY, J. EMER. *Adaptive insertion policies for high performance caching*, in "Proceedings of the 34th International Symposium on Computer Architecture", 2007.

[55] 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.

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

[57] 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.

[58] 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.

[59] 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.

[60] 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.

[61] L. R. 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.