INRIA

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Team alf

Amdahl's Law is Forever

Rennes - Bretagne-Atlantique

Theme : Architecture and Compiling

Activity Report

2009

Table of contents

1. Team

Research Scientist

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

Faculty Member

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

Technical Staff

Erven Rohou
Sylvain Leroy
Damien Fétis [till 31/03/09]
Thomas Piquet [till 30/06/09]
Thierry Lafage [from 01/11/09]
Guillaume Papauré [till 31/01/09]

PhD Student

Christophe Levointurier [Cifre AQL]
Junjie Lai [Inria Allocation, from 01/10/09]
Ricardo Velàsquez [Inria Allocation, from 01/12/09]
Nathanaël Prémillieu [Ecole Normale Supérieure de Cachan, Antenne de Bruz,from 01/09/09]
Benjamin Lesage [MESR allocation, from 01/10/09]
Damien Hardy [MESR allocation]
Julien Dusser [Inria Allocation till 31/08/09]
Julien Dusser [ATER, University of Rennes 1 from 01/09/09]
Robert Guziolowski [Inria Allocation, till 18/09/09]

Administrative Assistant

Evelyne Livache [TR Inria,till 30/09/2009]
Maryse Fouché [TR Inria,from 1/10/2009]

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 chip. 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 core 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, STMicroelectonics). We also participate in several institutionally funded projects (NoE HIPEAC2, IP Fet project SARC, ANR funded PetaQCD and Mascotte, “Pôles de compétitivités” funded Ter@ops and Serenitec).

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 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 a 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 process 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 succesfull 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 chanels — 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. Therefore, 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 core 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 [35] 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 core 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. Developping such simulators is quite complex and microarchitecture research was helped but also biased by some popular public domain research simulators (e.g. Simplescalar [37]). 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 [45] framework.

Real-time systems need a different use of performance prediction; on hard real-time systems, timing constraints must be respected independently from the data inputs and from the initial execution conditions. For such a usage, the Worst Case Execution Time (WCET) of an application must be evaluated and then checked against the timing constraints. While safe and tight WCET estimation techniques and tools exist for reasonably simple embedded processors (e.g. techniques based on abstract interpretation such as [39]), 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) [44].

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 core 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 core micro-architecture. New techniques have to be invented to achieve such simulations. WCET estimations on multicore platforms will also necessitate radically new techniques, in particular, there are predictability issues on a multicore where many resources are shared; those resources include the memory hierarchy, but also the processor execution units and all the hardware resources if SMT is implemented [49].

3.6. General research directions

The overall performance of a 1000's core system will depend on many parameters including architecture, operating system, runtime environment, compiler technology and application development. In the ALF project, we will essentially focus on architecture, compiler/execution environment as well 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 an 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 [38]. 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 [53], but the most promising solutions integrate hardware support for software thread-level speculation [47].

In the context of future manycores, thread-level speculation and helper threads should be revisited. Many simple cores will be available for executing helper threads or speculative thread execution during the execution of sequential programs or sequential code sections. The availability of these many cores is an opportunity as well as a challenge. For example, one can try to use the simple cores to execute many different helper threads that could not be implemented within a simultaneous multithreaded processor. For thread level speculation, the new challenge is the use of less 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* [46][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 core 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 [40] 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 [42] 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 (use SIMD/vector instructions, adapt to the available number of 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) [44]. 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 [43] 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 [51], therefore hardware-based or compiler methods such as [48] have to be defined to provide some degree of isolation between cores. Finally, similarly to simulation methods, new techniques to reduce the complexity of WCET estimation will be explored to cope with 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, **ATC** an address trace compressor and **HAVEGE**, an unpredictable random number generator, three 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. 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 [24].

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

5.4. 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, Robert Guziolowski, 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. We have first proposed the Zero-Content Augmented cache, the ZCA cache [19]. 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 [30]. 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. 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 be 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 design of a secure PCM-based main memory that would by construction survive to overwrite attacks [33]. In order to prevent a malicious user to overwrite 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 does 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, such preventing a malicious user to overwrite 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.

6.1.3. 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 are studying a new hardware mechanism called SRANT, Symmetric Resource Allocation on Not-taken and Taken paths, addressing control flow reconvergence.

6.1.4. 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"[31] 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 so as 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.

6.1.5. Exploiting confidence in SMT processors

Participants: Pierre Michaud, André Seznec.

Simultaneous multithreading (SMT) [50] processors dynamically share processor resources between multiple threads. The hardware allocates resources to different threads. The resources are either managed explicitly through setting resource limits to each thread or implicitly through placing the desired instruction mix in the resources. In this case, the main resource management tool is the instruction fetch policy which must predict the behavior of each thread (branch mispredictions, long-latency loads, etc.) as it fetches instructions.

We propose the use of Speculative Instruction Window Weighting (SIWW) [34] to bridge the gap between implicit and explicit SMT fetch policies. SIWW estimates for each thread the amount of outstanding work in the processor pipeline. Fetch proceeds for the thread with the least amount of work left. SIWW policies are implicit as fetch proceeds for the thread with the least amount of work left. They are also explicit as maximum resource allocation can also be set. SIWW can use and combine virtually any of the indicators that were previously proposed for guiding the instruction fetch policy (number of in-flight instructions, number of low confidence branches, number of predicted cache misses, etc.). Therefore, SIWW is an *approach to designing SMT fetch policies*, rather than a particular fetch policy.

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

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

6.2. Software tools for architecture

Participants: Ricardo Velàsquez, Pierre Michaud, André Seznec.

6.2.1. Fast hybrid multicore architecture simulation

Participants: Ricardo Velàsquez, Pierre Michaud, André Seznec.

So far, detailed cycle-accurate simulation has been the preferred methodology for research in microarchitecture. However, with the advent of multicore processors and the rapid growth of the number of on-chip cores, cycle accurate simulation is becoming more and more impractical, as it requires a lot of development and leads to slow and heavy simulators. To cope with this problem, researchers have begun exploring approximate multicore simulation methodologies that trade accuracy for simulation speed.

We are initiating a research effort to define a fast and accurate hybrid simulation methodology for simulating the execution of parallel applications on future multicore processors.

6.2.2. Online compression of cache-filtered address traces

Participant: Pierre Michaud.

Trace-driven simulation is potentially much faster than cycle-accurate simulation. However, one drawback 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 the compression ratio of existing trace compressors is limited because they implement lossless compression. We propose two new methods for compressing cache-filtered address traces. The first method, bytesort, is a lossless compression method that achieves high compression ratios on cache-filtered address traces. The second method is a lossy one, based on the concept of phase. We have combined these two methods in a trace compressor called ATC [24]. Our experimental results show that ATC gives high compression ratio while keeping the memory-locality characteristics of the original trace.

6.3. Enabling high performance applications on emerging architectures

Participants: François Bodin, Damien Fétis, Junjie Lai, André Seznec.

To achieve very high performance in some application domains, architectures must be specialized. We are involved in an ANR project aiming at defining an architecture for Lattice QCD (Quantum ChromoDynamics) and a “Pôle de compétitivité” project aiming at defining a powerful platform for embedded systems.

6.3.1. Architecture for Lattice QCD

Participants: François Bodin, Junjie Lai, André Seznec.

Simulation of Lattice QCD is a challenging computational problem that requires very high performance exceeding sustained Petaflops/s. In the framework of the ANR Cosinus PetaQCD project, we will model 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. The methodology should be able to be adapted to the study of other massively parallel applications to understand 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.

6.3.2. Data Locality Analysis of Parallel C Programs

Participants: François Bodin, Guillaume Papauré.

In the context of the POPS project, we have studied strategies to achieve efficient parallel execution based on OpenMP and assuming asymmetric core behaviors [36]. This work aims at understanding how to limit performance degradation of OpenMP programs when running on computation nodes that are shared by multiple applications. This competition for hardware resources is a particularly important efficiency factor in the context of hybrid parallel programming (e.g. mix of OpenMP and MPI) or in application servers used in a multi-user context.

6.3.3. Backend optimizations: Sofan for the Ter@ops project

Participants: François Bodin, Damien Fétis.

In the context of the ApeNEXT project [1], we have developed over the last few years SOFAN (Software Optimizer for ApeNEXT). This software optimizer attempts to explore different back end optimization strategies for ApeNEXT applications. In 2007, a production version of SOFAN was delivered to the users of ApeNEXT. In the context of the Ter@ops project, SOFAN has been retargeted towards one of the accelerators of the Ter@ops machine, the Thomson FIRE EVO coprocessor. FIRE EVO is SIMD coprocessor with an array of 16 VLIW processing units. A *gcc* code generator was developed by the Alchemy EPI for this accelerator. SOFAN was adapted to realise the control flow and data flow analysis of SIMD assembly code. It also optimizes the VLIW Processing Units resources utilisation.

6.4. Around processor virtualization

Participants: François Bodin, Christophe Levointurier, Sylvain Leroy, Erven Rohou, Thierry Lafage, André Seznec.

The usage of the Java language 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. However ensuring code quality maintenance and code security on those applications is challenging. To address these issues, we are defining a refactoring platform for Java. 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. We have initiated a research effort to efficiently compile towards bytecodes.

6.4.1. Analysis and transformation of Java codes

Participants: François Bodin, Sylvain Leroy, Christophe Levointurier.

The growing size of web applications makes it more and more difficult to keep the code secure with a high quality level. This is a challenge to developers. Code refactoring is one of the techniques to address this issue. However it must be automated to come at low cost and high productivity. The Serenitec (Security analysis and Refactoring ENvironment for Internet TEChnology) platform provides a flexible set of tools to implement Java source code analysis and refactoring.

Refactoring functions are expressed using sets of scripts written in the Java language. The complex analysis or refactoring operations are broken down into several simple scripts that are pasted into a task graph similar to Kahn's networks. For achieving fast processing of large applications (millions of lines of code) the scripts/tasks can then be executed in parallel if they deal with different parts of the code. Analysis and refactoring operations are gathered in reference sets. Each reference set aims to deal with a standard of coding rules. Serenitec offers static program analysis to ensure the correctness of source code transformations. Current implementation achieves high execution speed. For instance, interprocedural symbol renaming (variable, classes and methods) across large applications can be performed in a few minutes using a standard workstation. Serenitec can also be used in continuous integration engines such as Hudson [41].

6.4.2. Split vectorization

Participants: Erven Rohou, Thierry Lafage, 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, is key to obtaining the required performance.

We want to show that vectorized bytecode is a viable approach, that can yield the expected speedups in the presence of SIMD instructions, and a minor penalty in its absence. We also analyze the interaction between the static and the JIT compilers and we devise suggestions for a good performance of vectorized bytecode.

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

6.4.3. 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.

6.5. WCET estimation

Participants: Damien Hardy, Benjamin Lesage, Thomas Piquet, Isabelle Puaut.

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. A particular focus is put on hardware-level analysis (static analysis based on timing models) for multicore platforms.

In 2009, our new results concern the static WCET analysis of tasks running on multicore platforms with shared instruction and/or data caches.

6.5.1. Timing analysis of multicore platforms with shared instruction caches

Participants: Damien Hardy, Thomas Piquet, 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.

We have proposed in [21] a safe WCET estimation method for multi-core architectures with shared non-inclusive instruction cache(s). The method computes the WCET of one task running on one core, in competition with an arbitrary number of other tasks running on the other cores, and thus competing for the shared instruction cache. The proposed method, similarly to [52], estimates, using static analysis, the worst-case conflicts for the shared caches. However, it is more general than [52] in the sense that it supports multiple levels of shared caches, set-associative caches and an arbitrary number of real-time tasks and cores competing for the shared caches.

While safe, the proposed WCET estimation method might be too pessimistic in case of high pressure for the shared cache level(s). This potential pessimism is reduced in [21] by the proposal of the compiler-directed bypass scheme. Single-usage (not reused) program blocks in shared instruction caches are identified thanks to static analysis of the program code. This enables a compiler-directed bypass scheme, which, from the static knowledge of single-usage blocks, allows a drastic reduction of inter-task and intra-task interferences, and thus a tighter WCET estimate. The experimental results given in [21], demonstrate the practicality of our approach.

Our ongoing work is to support a larger panel of cache architectures (inclusive caches, exclusive caches).

6.5.2. Evaluation of cache-related migration delays

Participants: Damien Hardy, Isabelle Puaut.

Two approaches may be used for task scheduling in multicore systems: *partitioned* and *global* scheduling. Under partitioned scheduling, tasks are assigned to cores and are not allowed to migrate. While this class of approaches imposes no migration overhead, it has the following limitations: task partitioning is a NP-hard problem; dynamic task admittance is hard to support because it would require online re-partitioning. To address these limitations, global scheduling techniques have been proposed. The fundamental premises of these techniques is that task may migrate between cores.

However task migration results in a direct cost, required to save and restore the task context, but also in an indirect cost to reload the cache contents after the migration. After a migration, the reused cache blocks are reloaded in all levels of the cache hierarchy.

Task migration thus results extra cache misses compared with a migration-free execution. Extra cache misses occur in the private cache to load reused blocks as well as on the shared L2 cache when using non inclusive cache hierarchies.

We have proposed in [22] methods to compute safe estimations of the *Instruction Cache Related Migration Delay (CRMD)* suffered by a task after each migration. Our method relies on static code analysis. Our experimental results demonstrate estimated CRMDs much lower than those that would be obtained with a naive approach.

6.5.3. Timing analysis on multicore platforms with shared data and unified caches

Participants: Damien Hardy, Benjamin Lesage, Isabelle Puaut.

WCET estimation for multicore platforms requires to analyse data cache content as well as instruction cache content. As a first step towards considering the whole memory hierarchy of multicore platforms, we have proposed in [23] a static analysis of data cache hierarchies, based on our previous work published in [5] for instruction cache hierarchies. Our ongoing work concerns the use of (i) compiler-directed bypass to data as well as (ii) cache partitioning methods to decrease or avoid conflicts in shared data or unified caches.

6.6. Unpredictable random number generation on multicores

Participant: André Seznec.

The HAVEGE algorithm [12] generates unpredictable random numbers by gathering entropy from internal processor states that are inheritably volatile and impossible to tamper with in a controlled fashion by any application running on the target system. The method used to gather the entropy implies that its main loop will almost monopolize the CPU; the output depends on the operating system and other running applications, as well as some internal mechanisms that stir the processor states to generate an enormous amount of entropy. The algorithm was designed with the idea of single-core CPUs in mind, and no parallelization; however the recent market explosion of multi-core CPUs and the lack of results in increasing the CPU frequency justifies the need to research a multithreaded parallel version of HAVEGE, capable of running the same algorithm loop on each core independently and transparently combine the results in one single output bitstream. In [26], we demonstrate how such a parallelization is possible and benchmark the output speed of its implementation.

This research was done in collaboration with Alin Suciu, Tudor Carean, and Kinga Marton from Technical University of Cluj-Napoca.

7. Contracts and Grants with Industry

7.1. Research grant from Intel

Participants: Nathanaël Prémillieu, Julien Dusser, André Seznec.

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

7.2. 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.

7.3. Britanny region fellowship

Participants: Ricardo Velàsquez, Pierre Michaud, André Seznec.

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

7.4. Ter@ops project

Participants: François Bodin, Damien Fétis.

Ter@ops aims at defining and developing a large-scale embedded multi-core architectures. In this project, ALF adapts the backend optimizer SOFAN to a SIMD architecture part of the Ter@ops project.

This project is funded by the "Pôle de compétitivité SYSTEMATIC".

7.5. Mascotte

Participants: Isabelle Puaut, Thomas Piquet.

MasCotTE (http://www.projet-mascotte.org/) is an acronym for "MAîtriSe et COnTrôle des Temps d'Exécution" (Estimation and control of execution times). MasCotTE is funded by the Predit program of the ANR.

The aim of MasCotTE is to design the methods, techniques and tools required for controlling the execution times of automotive embedded real-time software (through static analysis and/or testing). Emphasis is put on the study of the impact of performance enhancing features on the predictability of embedded software. The project defines some guidelines on how to use such performance enhancing features in a predictable manner.

7.6. Nano2012 Mediacom

Participants: Erven Rohou, Thierry Lafage.

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 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, polyedral analysis, or auto-vectorization and auto-parallelization.

7.7. POPS

Participants: François Bodin, Guillaume Papauré.

POPS “Pour une nouvelle génération de serveurs et d’applications intensives á l’échelle du PetaFlops” is a project of the Pôle de compétitivité Systematics. The partners of the project are BULL, Caps enterprise, CS Systèmes d’information, EDF, ESI, Eurodecision, Medit, NewPhenix, Resonate, CEA DAM, CEA LIST, Ecole centrale de Paris, IFP, INRIA, INT, Université d’Evry, Université de Paris Sud, Université de Versailles St Quentin. The project aims at building supercomputers achieving Petaflop effective performance range.

In the project, we study programming environments to exploit the hybrid multicore architecture.

7.8. 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. 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, Robert Guziolowski, 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.

9. Dissemination

9.1. Scientific community animation

- Isabelle Puaut was a member of program committee of RTSS09 (Real-Time Systems Sumposium), RTNS 2009 (17th International Conference on Real-Time and Network Systems), RTCSA 2009 (15th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications), 2009 edition of the Real-Time and Networked Embedded Systems track of ETFA (14th IEEE international conference on Emerging Technologies and Factory Automation), WCET09 (9th Workshop on WCET analysis, held in conjunction with ECRTS09), Isabelle Puaut is a member of program committee of PC member of ECRTS 2010 (22nd Euromicro Conference on Real-Time Systems), RTAS 2010 (16th IEEE Real-Time and Embedded Technology and Applications Symposium), JTRES 2010 (The 8th International Workshop on Java Technologies for Real-time and Embedded Systems)
- Isabelle Puaut was the program committee chair of ECRTS 2009 (21th Euromicro Conference on Real-Time Systems), held in Dublin, July 2009.
- Pierre Michaud was a member of MuCoCoS 2009 program comittee. He is a member of MuCoCoS 2010 and e-Energy 2010 program committees.
- Erven Rohou is a member of the program committee of the 2PARMA Workshop to be held in the context of ARCS 2010 in Hannover.
- André Seznec was a member of ISPASS'09, MULTIPROG'09, CMP-MSI'09, Micro09 program comittees. He is a member of HPCA 2010, Computing Frontiers 2010, IPDPS 2010, MULTIPROG'10, CMP-MSI'10, Micro Top Picks 2010 program comittees. He is a member of the editorial board of the HiPEAC Transactions (Transactions on High-Performance Embedded Architectures and Compilers).
- André Seznec was the general co-chair of HiPEAC 2009 conference (Paphos, Cyprus, January 2009)
- The ALF team is organizing the 37th ISCA confence at Saint-Malo in June 2010. A. Seznec is the general chair. I. Puaut is the finance chair. P. Michaud is the local chair. E. Rohou is the web chair.

9.2. University teaching and responsabilities

- F. Bodin, A. Seznec, I. Puaut and E. Rohou are teaching computer architecture and compilation in the master of research in computer sciences at University of Rennes I.
- E. Rohou taught computer architecture labs at the engineering school of the University of Rennes 1 (DIIC2, IFSIC)
- E. Rohou taught labs of Computing Systems at École Polytechnique (INF422)
- I. Puaut teaches operating systems, real-time systems and real-time programming in the master degree of computer science of the University of Rennes I.
- Damien Hardy teaches compilation in the master degree of computer science of the University of Rennes 1, as well as real-time operating systems in the BSc degree *Embedded automotive systems* of the university of Rennes 1.
- Pierre Michaud, André Seznec and Erven Rohou are teaching computer architecture at the engineering degree in computer science at Ecole Supérieure d'ingénieurs de Rennes.
- Since september 2009, I. Puaut is co-responsible of the Master of Research in computer science in Britanny (University of Rennes I, University of Bretagne Sud, University of Bretagne Ouest, INSA, ENS Cachan antenne de Bruz, ENST Bretagne)

9.3. Workshops, seminars, invitations, visitors

- A. Seznec has presented a seminar on branch prediction at th IBM company in Yorktown Heights in september 2009 entitled “All you will never have wanted to know on branch prediction”.
- Pierre Michaud presented an invited seminar entitled "ATMI: Un modèle analytique de la température des microprocesseurs" at the Journée Thématique organized by the GDR SoC/Sip in May 2009.

- Pierre Michaud presented an invited seminar entitled "Introduction aux problèmes de consommation énergétique et d'échauffement des microprocesseurs" at the Journées du CUIC organized by the CEA in October 2009.

9.4. 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 a member of the steering comittee of the HIPEAC conference
- A. Seznec is an elected member of the scientific comittee of INRIA.

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", vol. 8, n^{o} 1, 2006, 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", vol. 4, n^{o} 2, 2007, 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", vol. 29, n^o 1, 2001, 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", vol. 22, n^o 2, 1994, p. 384–393, http://doi.acm.org/10.1145/192007.192072.

Year Publications

Doctoral Dissertations and Habilitation Theses

[16] E. PETIT. *Partitionnement Automatique d'Applications en Codelets Spéculatifs pour les Systèmes Hétérogènes à Mémoires Distribuées*, Université de Rennes I, May 2009, Ph. D. Thesis.

Articles in International Peer-Reviewed Journal

[17] H. VANDIERENDONCK, A. SEZNEC. *Fetch Gating Control through Speculative Instruction Window Weighting*, in "Transaction on HiPEAC", vol. 2, 2009, p. 128-148.

International Peer-Reviewed Conference/Proceedings

[18] F. BODIN, S. MATZ, M. WANG. *Improving Data Locality on the Cell BE Architecture*, in "Proceedings of the 22nd International Workshop on Languages and Compilers for Parallel Computing", 2009.

[19] J. DUSSER, T. PIQUET, A. SEZNEC. *Zero-content augmented caches*, in "Proceedings of International Conference on Supercomputing '09", June 2009, p. 46-55.

[20] G. GROSDIDIER, C. EISENBEIS, F. BODIN, A. SEZNEC, R. BILHAUT, G. LE MEUR, P. ROUDEAU, F. TOUZE, J. ANGLES D'AURIAC, J. CARBONELL, D. BECIREVIC, P. BOUCAUD, O. BRAND-FOISSAC, P. PENE, D. BARTHOU, P. GUICHON, P. HONORE, P. GALLARD, L. RILLING. *The PetaQCD project*, in "Proceedings of the 17th International Conference on Computing in High Energy and Nuclear Physics (CHEP09), Prague", 03 2009, http://hal.in2p3.fr/in2p3-00380246/en/.

[21] D. HARDY, T. PIQUET, I. PUAUT. *Using Bypass to Tighten WCET Estimates for Multi-Core Processors with Shared Instruction Caches*, in "Proc. of the 30th IEEE Real-Time Systems Symposium, Washington, D.C., USA", December 2009.

[22] D. HARDY, I. PUAUT. *Estimation of cache-related migration delays for multi-core processors with shared instruction caches*, in "Proc. of the 17th International Conference on Real-Time and Network Systems (RTNS), Paris, France", October 2009.

[23] B. LESAGE, D. HARDY, I. PUAUT. *WCET analysis of multi-level set-associative data caches*, in "Proc. of the 9th International Workshop on worst-case execution time analysis, in conjunction with the 14th Euromicro Conference on Real-Time Systems, Dublin, Ireland", July 2009.

[24] P. MICHAUD. *Online compression of cache-filtered address traces*, in "Proceedings of the IEEE International Symposium on Performance Analysis of Systems and Software", April 2009.

[25] E. ROHOU. *Combining Processor Virtualization and Split Compilation for Heterogeneous Multicore Embedded Systems*, in "Emerging Uses and Paradigms for Dynamic Binary Translation, Dagstuhl, Germany", B. R. CHILDERS, J. DAVIDSON, K. D. BOSSCHERE, M. L. SOFFA (editors), Dagstuhl Seminar Proceedings, n^{o} 08441, Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, Germany, 2009, http://drops.dagstuhl.de/opus/volltexte/2009/1887.

[26] A. SUCIU, T. CAREAN, A. SEZNEC, K. MARTON. *Parallel HAVEGE*, in "Proceedings of the 8th International Conference on Parallel Processing and Applied Mathematic", Sept 2009.

[27] G. SVELTO, A. ORNSTEIN, E. ROHOU. *A Stack-Based Internal Representation for GCC*, in "First International Workshop on GCC Research Opportunities (GROW09), in conjunction with HiPEAC 2009", January 2009, p. 37–48.

Books or Proceedings Editing

[28] I. PUAUT (editor). *Real-Time Systems, 2009. ECRTS '09. 21st Euromicro Conference on*, vol. 5409, July 2009, http://dx.doi.org/10.1109/ECRTS.2009.1.

[29] A. SEZNEC, J. S. EMER, M. F. P. O'BOYLE, M. MARTONOSI, T. UNGERER (editors). *High Performance Embedded Architectures and Compilers, Fourth International Conference, HiPEAC 2009, Paphos, Cyprus, January 25-28, 2009. Proceedings*, Lecture Notes in Computer Science, vol. 5409, Springer, 2009.

Research Reports

[30] J. DUSSER, A. SEZNEC. *Decoupled Zero-Compressed Memory*, n^{o} RR-7073, INRIA, 2009, http://hal.inria.fr/inria-00426765/en/, Research Report.

[31] P. MICHAUD, Y. SAZEIDES, A. SEZNEC. *Proposition for a sequential accelerator in future general-purpose manycore processors*, n^{o} RR-7106, INRIA, 2009, http://hal.inria.fr/inria-00433234/en/, Research Report.

[32] E. ROHOU, A. C. ORNSTEIN, A. E. ÖZCAN, M. CORNERO. *Combining Processor Virtualization and Component-Based Engineering in C for Heterogeneous Many-Core Platforms*, n^{o} RR-6933, INRIA, 2009, http://hal.inria.fr/inria-00397823/en/, Research Report.

[33] A. SEZNEC. *Towards Phase Change Memory as a Secure Main Memory*, n° RR-7088, INRIA, 2009, http://hal.inria.fr/inria-00430010/en/, Research Report.

[34] H. VANDIERENDONCK, A. SEZNEC. *Managing SMT Resource Usage through Speculative Instruction Window Weighting*, n° RR-7103, INRIA, 2009, http://hal.inria.fr/inria-00433081/en/, Research Report.

References in notes

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

[36] S. BALAKRISHNAN, R. RAJWAR, M. UPTON, K. LAI. *The Impact of Performance Asymmetry in Emerging Multicore Architectures*, in "SIGARCH Comput. Archit. News", vol. 33, n° 2, 2005, p. 506–517, http://doi.acm.org/10.1145/1080695.1070012.

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

[38] 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.

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

[40] 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.

[41] K. KAWAGUCHI. *Hudson monitors executions of repeated jobs, such as building a software project or jobs run by cron.*, http://hudson.dev.java.net/.

[42] 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.

[43] 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.

[44] 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.

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

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

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

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

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

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

[51] 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.

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

[53] 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.