IN PARTNERSHIP WITH:
Université Rennes 1

Activity Report 2014

Project-Team ALF

Amdahl's Law is Forever

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

RESEARCH CENTER
Rennes - Bretagne-Atlantique

THEME
Architecture, Languages and Compilation

Table of contents

Project-Team ALF

Keywords: Processors, Compilation, Real-time, Complexity, Multicore

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

1. Members

Research Scientists

André Seznec [Team leader, Inria, Senior Researcher, HdR]
Sylvain Collange [Inria, Researcher]
Pierre Michaud [Inria, Researcher]
Erven Rohou [Inria, Senior Researcher]

Faculty Members

Damien Hardy [Univ. Rennes I, Associate Professor]
Alain Ketterlin [on Inria secondment, Univ. Strasbourg, Associate Professor, until March 2014]
Isabelle Puaut [Univ. Rennes I, Professor, HdR]

Engineers

Thibault Person [Inria]
Emmanuel Riou [Inria]

PhD Students

Nabil Hallou [Inria]
Sajith Kalathingal [Inria]
Surya Khizakanchery Natarajan [Inria]
Hanbing Li [Inria]
Andrea Mondelli [Inria]
Bharath Narasimha Swamy [Inria]
Arthur Perais [Inria]
Aswinkumar Sridharan [Inria]
Arjun Suresh [Inria]

Post-Doctoral Fellow

Tao Sun [Inria, granted by FP7 ERC DAL project]

Visiting Scientists

Stijn Eyerman [PostDoc U. Ghent, from Apr 2014 until May 2014]
Erik Hagersten [Pr University of Uppsala, from Sep 2014 until Dec 2014]

Administrative Assistant

Virginie Desroches [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; parallel architectures will have to be exploited by every application from HPC computing, web and entreprise servers, but also PCs, smartphones and ubiquitous embedded systems.

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

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

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

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

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

Our research is partially supported by industry (Intel), the ANR W-SEPT project, the "projet d'investissement d'avenir" Usine Nouvelle (the project Capacités) and the European Union (NoE HiPEAC3, ERC grant DAL, COST action TACLe).

3. Research Program

3.1. Motivations

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

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

3.2. The context

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

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

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

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

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

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

3.2.3. The overall picture

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

3.3. Technology induced challenges

3.3.1. The power and temperatures walls

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

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

3.3.2. The memory wall

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

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

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

3.4. Need for efficient execution of parallel applications

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

3.4.1. The diversity of parallelisms

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

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

3.4.2. Portability is the new challenge

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

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

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

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

3.4.3. The need for performance on sequential code sections

3.4.3.1. Most software will exhibit substantial sequential code sections

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

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

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

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

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

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

In 1967, Amdahl [37] pointed out that, if only a portion of an application is accelerated, the execution time cannot be reduced below the execution time of the residual part of the application. Unfortunately, even highly parallelized applications exhibit some residual sequential part. For parallel applications, this indicates that the effective performance of the future 1000's cores chip will significantly depend on their ability to be efficient on the execution of the control portions of the main thread as well as on the execution of sequential portions of the application.

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

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

3.5. Performance evaluation/guarantee

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

When proposing a micro-architecture mechanism, its impact on the overall processor architecture has to be evaluated in order to assess its potential performance advantages. For microarchitecture research, this evaluation is generally done through the use of cycle-accurate simulation. Developing such simulators is quite complex and microarchitecture research was helped but also biased by some popular public domain research simulators (e.g. Simplescalar [38]). Such simulations are CPU consuming and simulations cannot be run on a complete application. Sampling representative slices of the application was proposed [4] and popularized by the Simpoint [48] 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 [40]), accurate evaluation of the WCET of an algorithm on a complex uniprocessor system is a difficult problem. Accurately modelling data cache behavior [3] 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) [45].

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 [52].

3.6. General research directions

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

3.6.1. Microarchitecture research directions

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

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

3.6.1.1. Enhancing complex core microarchitecture

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

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

On the complex core, the performance of a sequential code is limited by several factors. At first, on current architectures, it is limited by the peak performance of the processor. To push back this first limitation, we will explore new microarchitecture mechanisms to increase the potential peak performance of a complex core enabling larger instruction issue width. The processor performance is also limited by control dependencies. To push back this limitation, we will explore new branch prediction mechanisms as well as new directions for reducing branch misprediction penalties [10], [12]. As data dependencies may strongly limit performance, we will revisit data prediction. Processor performance is also often highly dependent on the presence or absence of data in a particular level of the memory hierarchy. For the ALF multicore, we will focus on sharing the access to the memory hierarchy in order to adapt the performance of the main thread to the performance of the other cores. All these topics should be studied with the new perspective of quasi unlimited silicon budget.

3.6.1.2. Exploiting heterogeneous multicores on single process

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

Helper threads were initially proposed to improve the performance of the main threads on simultaneous multithreaded architectures [39]. The main idea of helper threads is to execute codes that will accelerate the main thread without modifying its semantic.

In many cases, the compiler cannot determine if two code sections are independent due to some unresolved memory dependency. When no dependency occurs at execution time, the code sections can be executed in parallel. Thread-Level Speculation has been proposed to exploit coarse grain speculative parallelism. Several hardware-only proposals were presented [46], but the most promising solutions integrate hardware support for software thread-level speculation [50].

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

3.6.1.3. Temperature issues

Temperature is one of the constraints that have prevented the processor clock frequency to be increased in recent years. Besides techniques to decrease the power consumption, the temperature issue can be tackled with *dynamic thermal management* [9] through techniques such as clock gating or throttling and *activity migration* [49][5].

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

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

3.6.2. Processor simulation research

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

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

For simulating parallel code sections, a cycle-accurate microarchitecture simulator of a 1000-core architecture will be unacceptably slow. In [6], 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 [42] further developed a detailed analytical simulation model of a superscalar processor. Building on top of these preliminary researches, simulation methodology mixing analytical modeling of the simple cores with a more detailed simulation of the complex cores is appealing. The analytical model of the simple cores will aim at approximately modeling the impact of the simple core execution on the shared resources (e.g. data bandwidth, memory hierarchy) that are also used by the complex cores.

Other techniques such as regression modeling [43] can also be used for decreasing the time required to explore the large space of microarchitecture parameter values. We will explore these techniques in the context of many-core simulation.

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

3.6.3. Compiler research directions

3.6.3.1. General directions

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

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

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

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

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

3.6.3.2. Portability of applications and performance through virtualization

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

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

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

3.6.4. Performance predictability for real-time systems

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

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

For sequential code executing on a single core, one can expect that, in order to provide real-time possibility, the architecture will feature an execution mode where a given processor will be guaranteed to access a fixed portion of the shared resources (caches, memory bandwidth). Moreover, this guaranteed share could be optimized at compile time to enforce the respect of the time constraints. However, estimating the WCET of an application on a complex micro-architecture is still a research challenge. This is due to the complex interaction of micro-architectural elements (superscalar pipelines, caches, branch prediction, out-of-order execution) [45]. 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 [44] might be needed to explore to obtain as safe as possible WCET estimates.

Providing performance guarantees for parallel applications executed on a multicore is a new and challenging issue. Entirely new WCET estimation methods have to be defined for these architectures to cope with dynamic resource sharing between cores, in particular on-chip memory (either local memory or caches) are shared, but

also buses, network-on-chip and the access to the main memory. Current pure analytical methods are too pessimistic at capturing interferences between cores [53], therefore hardware-based or compiler methods such as [51] have to be defined to provide some degree of isolation between cores. Finally, similarly to simulation methods, new techniques to reduce the complexity of WCET estimation will be explored to cope with manycore architectures.

4. Application Domains

4.1. Any computer usage

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

5. New Software and Platforms

5.1. Panorama

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

Among the prototypes developed in the project, this section reports only the softwares that had significant revisions in 2014. Among the softwares available from the project website and not reported here, **ATMI** **http://www.irisa.fr/alf/atmi** , a microarchitecture temperature model for processor simulation, **STiMuL** **http://www.irisa.fr/alf/stimul**, a temperature model for steady state studies, **ATC** **http://www.irisa.fr/alf/atc**, an address trace compressor, and **HAVEGE** **http://www.irisa.fr/alf/havege** an unpredictable random number generator.

5.2. TPCalc

Participant: Pierre Michaud.

microarchitecture simulation

TPCalc is a throughput calculator for microarchitecture studies concerned with multi-program workloads consisting of sequential programs. Because microarchitecture simulators are slow, it is difficult to simulate throughput experiments where a multicore executes many jobs that enter and leave the system. The usual practice of measuring instantaneous throughput on independent coschedules chosen more or less randomly is not a rigorous practice because it assumes that all the coschedules are equally important, which is not always true. TPCalc can compute the average throughput of a throughput experiment without actually doing the throughput experiment. The user first defines the workload heterogeneity (number of different job types), the multicore configuration (number of cores and symmetries). TPCalc provides a list of base coschedules. The user then simulates these coschedules, using some benchmarks of his/her choice, and feeds back to TPCalc the measured execution rates (e.g., instructions per cycle or instructions per second).TPCalc eventually outputs the average throughput. Several throughput metrics are available, corresponding to different workload assumptions. These metrics are described in our ACM TACO paper, a collaboration with Ghent University [15].

TPCalc is an open-source software written in C++. It runs on Unix-based systems (Linux, OS X ...). It is available for download at http://www.irisa.fr/alf/downloads/michaud/tpcalc.html.

5.3. Heptane

Participants: Isabelle Puaut, Damien Hardy.

WCET estimation

Status: Registered with APP (Agence de Protection des Programmes). Available under GNU General Public License v3, with number IDDN.FR.001.510039.000.S.P.2003.000.10600.

The aim of Heptane is to produce upper bounds of the execution times of applications. It is targeted at applications with hard real-time requirements (automotive, railway, aerospace domains). Heptane computes WCETs using static analysis at the binary code level. It includes static analyses of microarchitectural elements such as caches and cache hierarchies.

For more information, please contact Damien Hardy or Isabelle Puaut.

5.4. Tiptop

Participant: Erven Rohou.

Performance, hardware counters, analysis tool.

Status: Registered with APP (Agence de Protection des Programmes). Available under GNU General Public License v2, with number IDDN.FR.001.450006.000.S.P.2011.000.10800. Current version is 2.2, released March 2013.

Tiptop has been integrated in major Linux distributions, such as Fedora, Debian, Ubuntu.

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

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

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

For more information, please contact Erven Rohou or visit http://tiptop.gforge.inria.fr.

5.5. Padrone

Participants: Erven Rohou, Emmanuel Riou.

Performance, profiling, dynamic optimization

Status: Ongoing development, early prototype. Registered with APP (Agence de Protection des Programmes).

Padrone is new platform for dynamic binary analysis and optimization. It provides an API to help clients design and develop analysis and optimization tools for binary executables. Padrone attaches to running applications, only needing the executable binary in memory. No source code or debug information is needed. No application restart is needed either. This is specially interesting for legacy or commercial applications, but also in the context of cloud deployment, where actual hardware is unknown, and other applications competing for hardware resources can vary. The profiling overhead is minimum.

Padrone is instrumental to the PhD developments of Nabil Hallou.

Padrone is written in C.

For more information, please contact Erven Rohou.

5.6. Barra

Participant: Sylvain Collange.

GPU simulator

Other Contributors : David Defour (Université de Perpignan), Alexandre Kouyoumdjian (Inria), Elie Gedeon (ENS Lyon), Fabrice Mouhartem (Inria)

Status : APP registration in progress. Available under the new BSD License

Research on throughput-oriented architectures demands accurate and representative models of GPU architectures in order to be able to evaluate new architectural ideas, explore design spaces and characterize applications. The Barra project [1] is a simulator of the NVIDIA Tesla GPU architecture.

Barra builds upon knowledge acquired through micro-benchmarking, in order to provide a baseline model representative of industry practice. The simulator provides detailed statistics to identify optimization opportunities and is fully customizable to experiment ideas of architectural modifications. Barra incorporates both a functional model and a cycle-level performance model.

Visit http://barra.gforge.inria.fr/ or contact Sylvain Collange.

6. New Results

6.1. Highlights of the Year

André Seznec and Pierre Michaud won the 4th Championship Branch Prediction in all the 3 categories, 4KB, 32 KB and unlimited storage predictors [23], [33], thus confirming the past championships in 2011, 2006 and 2004.

6.2. Processor Architecture

Participants: Pierre Michaud, Bharath Narasimha Swamy, Sylvain Collange, Erven Rohou, André Seznec, Arthur Perais, Surya Khizakanchery Natarajan, Sajith Kalathingal, Tao Sun, Andrea Mondelli, Aswinkumar Sridharan, Alain Ketterlin.

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

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

[1] http://gforge.inria.fr/plugins/mediawiki/wiki/barra/index.php/Main_Page

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

In the DAL research project, http://www.irisa.fr/alf/dal, we explore the microarchitecture techniques that will be needed to enable high performance on such heterogeneous processor chips. Very high performance will be required on both sequential sections, -legacy sequential codes, sequential sections of parallel applications-, and critical threads on parallel applications, -e.g. the main thread controlling the application. Our research focuses essentially on enhancing single process performance.

6.2.1. Microarchitecture

6.2.1.1. Branch prediction

Participants: André Seznec, Pierre Michaud.

We submitted 3 predictors to the 4th Championship Branch Prediction that took place along with the ISCA 2014 conference [33], [22], [23]. Our predictors combine some branch prediction techniques that we introduced in our previous works, in particular TAGE [10] and GEHL [12]. The predictor we submitted to the 4KB and 32KB tracks was ranked first [33] in both tracks. The 3 predictors we submitted to the unlimited-size track took the first three ranks. We have established a new reference point for branch predictability limits [23].

The 12 competing predictors were mostly using already published branch prediction techniques. The main learning from this year's contest is that choosing the right combination of techniques for the given constraints is at least as important as trying to specialize branch predictors for certain branch behaviors.

6.2.1.2. Revisiting Value Prediction

Participants: Arthur Perais, André Seznec.

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

First, we introduce a new value predictor VTAGE harnessing the global branch history [29]. VTAGE directly inherits the structure of the indirect jump predictor ITTAGE [10]. VTAGE is able to predict with a very high accuracy many values that were not correctly predicted by previously proposed predictors, such as the FCM predictor and the stride predictor. Three sources of information can be harnessed by these predictors: the global branch history, the differences of successive values and the local history of values. Moreover, VTAGE does not suffer from short critical prediction loops and can seamlessly handle back-to-back predictions, contrarily to previously proposed, hard to implement FCM predictors.

Second, we show that all predictors are amenable to very high accuracy at the cost of some loss on prediction coverage [29]. This greatly diminishes the number of value mispredictions and allows to delay validation until commit-time. As such, no complexity is added in the out-of-order engine because of VP (save for ports on the register file) and pipeline squashing at commit-time can be used to recover. This is crucial as adding *selective replay* in the OoO core would tremendously increase complexity.

Third, we leverage the possibility of validating predictions at commit to introduce a new microarchitecture, EOLE [28]. EOLE features *Early Execution* to execute simple instructions whose operands are ready in parallel with Rename and *Late Execution* to execute simple predicted instructions and high confidence branches just before Commit. EOLE depends on Value Prediction to provide operands for *Early Execution* and predicted instructions for *Late Execution*. However, Value Prediction requires EOLE to become truly practical. That is, EOLE allows to reduce the out-of-order issue-width by 33% without impeding performance. As such, the number of ports on the register file diminishes. Furthermore, optimizations of the register file such as *banking* further reduce the number of required ports. Overall EOLE possesses a register file whose complexity is on-par with that of a regular wider-issue superscalar while the out-of-order components (scheduler, bypass)

are greatly simplified. Moreover, thanks to Value Prediction, speedup is obtained on many benchmarks of the SPEC'00/'06 suite.

6.2.1.3. Skewed Compressed Caches

Participant: André Seznec.

Cache compression seeks the benefits of a larger cache with the area and power of a smaller cache. Ideally, a compressed cache increases effective capacity by tightly compacting compressed blocks, has low tag and metadata overheads, and allows fast lookups. Previous compressed cache designs, however, fail to achieve all these goals. In this study, we propose the Skewed Compressed Cache (SCC), a new hardware compressed cache that lowers overheads and increases performance. SCC tracks super- blocks to reduce tag overhead, compacts blocks into a variable number of sub-blocks to reduce internal fragmentation, but retains a direct tag-data mapping to find blocks quickly and eliminate extra metadata (i.e., no backward pointers). SCC does this using novel sparse super-block tags and a skewed associative mapping that takes compressed size into account. In our experiments, SCC provides on average 8% (up to 22%) higher performance, and on average 6% (up to 20%) lower total energy, achieving the benefits of the recent Decoupled Compressed Cache [47] with a factor of 4 lower area overhead and lower design complexity.

This study was done in collaboration with Somayeh Sardashti and David Wood from University of Wisconsin.

6.2.1.4. Efficient Execution on Guarded Instruction Sets

Participant: André Seznec.

ARM ISA based processors are no longer low complexity processors. Nowadays, ARM ISA based processor manufacturers are struggling to implement medium-end to high-end processor cores which implies implementing a state-of-the-art out-of-order execution engine. Unfortunately providing efficient out-of-order execution on legacy ARM codes may be quite challenging due to guarded instructions.

Predicting the guarded instructions addresses the main serialization impact associated with guarded instructions execution and the multiple definition problem. Moreover, guard prediction allows to use a global branch-and-guard history predictor to predict both branches and guards, often improving branch prediction accuracy. Unfortunately such a global branch-and-guard history predictor requires the systematic use of guard predictions. In that case, poor guard prediction accuracy would lead to poor overall performance on some applications.

Building on top of recent advances in branch prediction and confidence estimation, we propose a hybrid branch and guard predictor, combining a global branch history component and global branch-and-guard history component. The potential gain or loss due to the systematic use of guard prediction is dynamically evaluated at run-time. Two computing modes are enabled: systematic guard prediction use and high confidence only guard prediction use. Our experiments show that on most applications, an overwhelming majority of guarded instructions are predicted. Therefore a relatively inefficient but simple hardware solution can be used to execute the few unpredicted guarded instructions. Significant performance benefits are observed on most applications while applications with poorly predictable guards do not suffer from performance loss [7].

This study is accepted to ACM Transactions on Architecture and Compiler Optimizations (to appear January 2015) and will be presented at the HIPEAC conference in January 2015.

6.2.1.5. Clustered microarchitecture

Participants: Andrea Mondelli, Pierre Michaud, André Seznec.

In the last 10 years, the clock frequency of high-end superscalar processors did not increase significantly. Performance keeps being increased mainly by integrating more cores on the same chip and by introducing new instruction set extensions. However, this benefits only to some applications and requires rewriting and/or recompiling these applications. A more general way to increase performance is to increase the IPC, the number of instructions executed per cycle.

We argue that some of the benefits of technology scaling should be used to increase the IPC of future superscalar cores. Starting from microarchitecture parameters similar to recent commercial high-end cores, we show that an effective way to increase the IPC is to increase the issue width. But this must be done without impacting the clock cycle. We propose to combine two known techniques: clustering and register write specialization. The objective of past work on clustered microarchitecture was to allow a higher clock frequency while minimizing the IPC loss. This led researchers to consider narrow-issue clusters. Our objective, instead, is to increase the IPC without impacting the clock cycle, which means wide-issue clusters. We show that, on a wide-issue dual cluster, a very simple steering policy that sends 64 consecutive instructions to the same cluster, the next 64 instructions to the other cluster, and so on, permits tolerating an inter-cluster delay of several cycles. We also propose a method for decreasing the energy cost of sending results of one cluster to the other cluster.

This work is currently under submission.

6.2.1.6. Adaptive Intelligent Memory Systems

Participants: André Seznec, Aswinkumar Sridharan.

On multicores, the processors are sharing the memory hierarchy, buses, caches, and memory. The performance of any single application is impacted by its environment and the behavior of the other applications co-running on the multicore. Different strategies have been proposed to isolate the behavior of the different co-running applications, for example performance isolation cache partitioning, while several studies have addressed the global issue of optimizing throughput through the cache management.

However these studies are limited to a few cores (2-4-8) and generally feature mechanisms that cannot scale to 50-100 cores. Moreover so far the academic propositions have generally taken into account a single parameter, the cache replacement policy or the cache partitioning. Other parameters such as cache prefetching and its aggressiveness already impact the behavior of a single thread application on a uniprocessor. Cache prefetching policy of each thread will also impact the behavior of all the co-running threads.

Our objective is to define an Adaptive and Intelligent Memory System management hardware, AIMS. The goal of AIMS will be to dynamically adapt the different parameters of the memory hierarchy access for each individual co-running process in order to achieve a global objective such as optimized throughput, thread fairness or respecting quality of services for some privileged threads.

6.2.2. Microarchitecture Performance Modeling

6.2.2.1. Multiprogram throughput of multicore/SMT processors

Participant: Pierre Michaud.

This research was done in collaboration with Stijn Eyerman and Wouter Rogiest from Ghent University.

There are several aspects to the performance of a multicore processor. One of them is multiprogram throughput, that is, how fast a multicore can execute several independent jobs. However, defining throughput metrics that are both meaningful and practical for computer architecture studies is not straightforward. We present a method to construct throughput metrics in a systematic way: we start by expressing assumptions on job sizes, job types distribution, scheduling, etc., that together define a theoretical throughput experiment. The throughput metric is then the average throughput of this experiment. Different assumptions lead to different metrics, so one should be aware of these assumptions when making conclusions based on results using a specific metric. Throughput metrics should always be defined from explicit assumptions, because this leads to a better understanding of the implications and limits of the results obtained with that metric. We elaborate multiple metrics based on different assumptions. In particular, we show that commonly used throughput metrics such as instructions per cycle and weighted speedup implicitly assume a variable workload, that is, a workload which depends on the machine being evaluated. However, in many situations, it is more realistic to assume a fixed workload. Hence we propose some new fixed-workload throughput metrics. Evaluating these new metrics requires to solve a continuous-time Markov chain. We released a software, TPCalc, that takes as input the performance results of individual coschedules simulations and computes fixed-workload throughput, taking advantage of multicore symmetries [15].

In a subsequent work, we applied our framework to symbiotic scheduling on a symmetric multicore or SMT processor. Symbiotic scheduling tries to exploit the fact, because of resource sharing (execution units, caches, memory bandwidth, etc.) and because different jobs have different characteristics, the performance may be increased by carefully choosing the coschedules. We show that, when assuming a fixed workload, an optimal schedule maximizing throughput can be found by solving a linear programming problem. However, the throughput gains we observed in our experiments, 3% on average, are significantly smaller than what we expected based on published studies on symbiotic scheduling. We analyzed the reasons for this and we found the two main reasons for this discrepancy: previous studies either did not consider a fixed workload but a variable one, or did not report throughput gains but response time reductions. Response time reductions can be artificially magnified by setting the job arrival rate close to the maximum throughput.

This work will be presented at the ISPASS 2015 conference.

6.2.2.2. Modeling multi-threaded programs execution time in the many-core era

Participants: Surya Khizakanchery Natarajan, Bharath Narasimha Swamy, André Seznec.

Estimating the potential performance of parallel applications on the yet-to-be-designed future many cores is very speculative. The traditional laws used to predict performance of an application do not reflect on the various scaling behaviour of a multi-threaded (MT) application leading to optimistic estimation of performance in manycore era. In this paper, we study the scaling behavior of MT applications as a function of input workload size and the number of cores. For some MT applications in the benchmark suites we analysed, our study shows that the serial fraction in the program increases with input workload size and can be a scalability-limiting factor. Similar to previous studies [41] , we find that using a powerful core (heterogeneous architecture) to execute this serial part of the program can mitigate the impact of serial scaling and improve the overall performance of an application in many-core era [25].

6.2.3. Hardware/Software Approaches

6.2.3.1. Helper threads

Participants: Bharath Narasimha Swamy, Alain Ketterlin, André Seznec.

Heterogeneous Many Cores (HMC) architectures that mix many simple/small cores with a few complex/large cores are emerging as a design alternative that can provide both fast sequential performance for single threaded workloads and power-efficient execution for throughput oriented parallel workloads. The availability of many small cores in a HMC presents an opportunity to utilize them as low-power helper cores to accelerate memory-intensive sequential programs mapped to a large core. However, the latency overhead of accessing small cores in a loosely coupled system limits their utility as helper cores. Also, it is not clear if small cores can execute helper threads sufficiently in advance to benefit applications running on a larger, much powerful, core. In [24], we present a hardware/software framework called core-tethering to support efficient helper threading on heterogeneous many-cores. Core-tethering provides a co-processor like interface to the small cores that (a) enables a large core to directly initiate and control helper execution on the helper core and (b) allows efficient transfer of execution context between the cores, thereby reducing the performance overhead of accessing small cores for helper execution. Our evaluation on a set of memory intensive programs chosen from the standard benchmark suites show that, helper threads using moderately sized small cores can significantly accelerate a larger core compared to using a hardware prefetcher alone. We find that a small core provides a good trade-off against using an equivalent large core to run helper threads in a HMC. Additionally, helper prefetching on small cores when used along with hardware prefetching, can provide an alternate design point to growing instruction window size for achieving higher sequential performance on memory intensive applications.

6.2.3.2. Branch Prediction and Performance of Interpreter

Participants: Erven Rohou, André Seznec, Bharath Narasimha Swamy.

Interpreters have been used in many contexts. They provide portability and ease of development at the expense of performance. The literature of the past decade covers analysis of why interpreters are slow, and many software techniques to improve them. A large proportion of these works focuses on the dispatch loop, and in particular on the implementation of the switch statement: typically an indirect branch instruction. Folklore attributes a significant penalty to this branch, due to its high misprediction rate. We revisit this assumption, considering state-of-the-art branch predictors and the three most recent Intel processor generations on current interpreters. Using both hardware counters on Haswell, the latest Intel processor generation, and simulation of the ITTAGE, we show that the accuracy of indirect branch prediction is no longer critical for interpreters. We further compare the characteristics of these interpreters and analyze why the indirect branch is less important than before.

This study [8] has been accepted for publication at CGO 2015 (International Symposium on Code Generation and Optimization).

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

Participants: Sylvain Collange, André Seznec, Sajith Kalathingal.

We aim at exploring the design of a unique core that efficiently runs both sequential and massively parallel sections. We explore how the architecture of a complex superscalar core has to be modified or enhanced to efficiently run several threads from the same application.

Rather than vectorize at compile-time, our approach is to dynamically vectorize SPMD programs at the micro-architectural level. The SMT-SIMD hybrid core we propose extracts data parallelism from thread parallelism by scheduling groups of threads in lockstep, in a way inspired by the execution model of GPUs. As in GPUs, conditional branches whose outcomes differ between threads are handled with conditionally masked execution. However, while GPUs rely on explicit re-convergence instructions to restore lockstep execution, we target existing general-purpose instruction sets, in order to run legacy binary programs. Thus, the main challenge consists in detecting re-convergence points dynamically.

To handle this difficulty, we can build on [17]. In this work done in collaboration with Fernando Pereira and his team at UFMG, Brasil, we proposed instruction fetch policies that apply heuristics to maximize the cycles spent in lockstep execution, and evaluated them under a micro-architecture independent model [17]. Results highlight the necessity of a tradeoff between maximizing throughput and extracting data-level parallelism with lockstep execution.

6.3. Compiler, vectorization, interpretation

Participants: Erven Rohou, Emmanuel Riou, Bharath Narasimha Swamy, Arjun Suresh, André Seznec, Nabil Hallou, Alain Ketterlin, Sylvain Collange.

6.3.1. Compilers for emerging throughput architectures

Participant: Sylvain Collange.

This work is done in collaboration with Fernando Pereira and his team at UFMG, Brasil.

GPU architectures present new challenges for compilers. Their performance characteristics demand SPMD programs with a high control-flow and memory regularity. Such architecture takes advantage of the regularity in programs to exploit data-level parallelism. In addition to the traditional challenges of code parallelization, new compilers for GPU and future throughput architectures face the task of improving the regularity of parallel programs. In particular, compiler analyses that identify control-flow divergence and memory divergence are a stepping stone for many optimizations. These optimizations include traditional code transformation such as loop interchange and tiling, which use divergence as an additional decision criterion, but also new optimizations specific to GPU architectures such as iteration delaying or branch fusion. In addition, the regularity parameter is an important aspect for workload characterization, as it provides a criterion for task scheduling in heterogeneous environments, such as multi-core processors with GPU. Our objectives include both accurate static and dynamic analyses for thread divergence, and the applications that it enables. We propose to combine static analyses with runtime checks, in order to get the best from both complementary approaches.

6.3.2. Improving sequential performance through memoization

Participants: Erven Rohou, André Seznec, Arjun Suresh.

Many applications perform repetitive computations, even when properly programmed and optimized. Performance can be improved by caching results of pure functions, and retrieving them instead of recomputing a result (a technique called memoization).

We proposed a simple technique for enabling software memoization of any dynamically linked pure function and we illustrate our framework using a set of computationally expensive pure functions – the transcendental functions.

Our technique does not need the availability of source code and thus can be applied even to commercial applications as well as applications with legacy codes. As far as users are concerned, enabling memoization is as simple as setting an environment variable.

Our framework does not make any specific assumptions about the underlying architecture or compiler tool-chains, and can work with a variety of current architectures.

We present experimental results for x86-64 platform using both gcc and icc compiler tool-chains, and for ARM cortex-A9 platform using gcc. Our experiments include a mix of real world programs and standard benchmark suites: SPEC and Splash2x. On standard benchmark applications that extensively call the transcendental functions we report memoization benefits of upto 16 %, while much higher gains were realized for programs that call the expensive Bessel functions. Memoization was also able to regain a performance loss of 76 % in *bwaves* due to a known performance bug in the gcc libm implementation of *pow* function.

6.3.3. Code Obfuscation

Participant: Erven Rohou.

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

A new obfuscation technique [27] based of decomposition of CFGs into threads has been proposed. We exploit the mainstream multi-core processing in these systems to substantially increase the complexity of programs, making reverse engineering more complicated. The novel method automatically partitions any serial thread into an arbitrary number of parallel threads, at the basic-block level. The method generates new control-flow graphs, preserving the blocks' serial successor relations and guaranteeing that one basic-block is active at a time through using guards. The method generates m^n different combinations for m threads and n basic-blocks, significantly complicating the execution state. We also provide proof of correctness for the method.

We propose to leverage JIT compilation to make software tamper-proof. The idea is to constantly generate different versions of an application, even while it runs, to make reverse engineering hopeless. More precisely a JIT engine is used to generate new versions of a function each time it is invoked, applying different optimizations, heuristics and parameters to generate diverse binary code. A strong random number generator will guarantee that generated code is not reproducible, though the functionality is the same.

This work has been accepted for publication in January 2015 at the International Workshop on Dynamic Compilation Everywhere (DCE-2015).

6.3.4. Padrone

Participants: Erven Rohou, Alain Ketterlin, Emmanuel Riou.

The objective of the ADT PADRONE is to design and develop a platform for re-optimization of binary executables at run-time. Development is ongoing, and an early prototype is functional. In [30], we described the infrastructure of Padrone, and showed that its profiling overhead is minimum. We illustrated its use through two examples. The first example shows how a user can easily write a tool to identify hotspots in their application, and how well they perform (for example, by computing the number of executed instructions per cycle). In the second example, we illustrate the replacement of a given function (typically a hotspot) by an optimized version, while the program runs.

We believe PADRONE fills an empty design point in the ecosystem of dynamic binary tools.

6.3.5. Dynamic Binary Re-vectorization

Participants: Erven Rohou, Nabil Hallou, Alain Ketterlin, Emmanuel Riou.

This work is done in collaboration with Philippe Clauss (Inria CAMUS).

Applications are often under-optimized for the hardware on which they run. Several reasons contribute to this unsatisfying situation, including the use of legacy code, commercial code distributed in binary form, or deployment on compute farms. In fact, backward compatibility of instruction sets guarantees only the functionality, not the best exploitation of the hardware. In particular SIMD instruction sets are always evolving.

We proposed a runtime re-vectorization platform that dynamically adapts applications to execution hardware. Programs distributed in binary forms are re-vectorized at runtime for the underlying execution hardware. Focusing on the x86 SIMD extensions, we are able to automatically convert loops vectorized for SSE into the more recent and powerful AVX. A lightweight mechanism leverages the sophisticated technology put in a static vectorizer and adjusts, at minimal cost, the width of vectorized loops. We achieve speedups in line with a native compiler targeting AVX. Our re-vectorizer is implemented inside a dynamic optimization platform; its usage is completely transparent to the user and requires neither access to source code nor rewriting binaries.

6.4. WCET estimation

Participants: Damien Hardy, Hanbing Li, 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 tasks temporal constraints (typically, deadlines) will be met. Our research concerns methods for obtaining automatically upper bounds of the execution times of applications on a given hardware. Our new results this year are on (i) multi-core architectures (ii) WCET estimation for faulty architectures (iii) traceability of flow information in compilers for WCET estimation.

6.4.1. WCET estimation and its interactions with compilation

6.4.1.1. On the comparison of deterministic and probabilistic WCET estimation techniques

Participants: Damien Hardy, Isabelle Puaut.

This is joint work with Jaume Abella, Eduardo Quinones and Francisco J. Cazorla from Barcelona Supercomputing Center

Several timing analysis techniques have been proposed to obtain Worst-Case Execution Time (WCET) estimates of applications running on a particular hardware. They can be classified into two classes of approaches: deterministic timing analysis techniques (DTA), that produce a unique WCET estimate, and probabilistic timing analysis techniques (PTA) that produce multiple WCET estimates with associated probabilities. Both approaches have their static (SDTA, SPTA) and measurement-based (MBDTA, MBPTA) variants. The lack of comparison figures among those techniques makes complex the selection of the most appropriate one.

This work [19] makes a first attempt towards comparing comprehensively SDTA, SPTA and MBPTA qualitatively and quantitatively, under different cache configurations implementing LRU and random replacement. We identify strengths and limitations of each technique depending on the characteristics of the program under analysis and the hardware platform, thus providing users with guidance on which approach to choose depending on their target application and hardware platform.

6.4.2. WCET estimation for architectures with faulty caches

Participants: Damien Hardy, Isabelle Puaut.

Technology scaling, used to increase performance, has the negative consequence of providing less reliable silicon primitives, resulting in an increase of the probability of failure of circuits, in particular for SRAM cells. While space redundancy techniques exist to recover from failures and provide fault-free chips, they will not be affordable anymore in the future due to their growing cost. Consequently, other approaches like fine grain disabling and reconfiguration of hardware elements (e.g. individual functional units or cache blocks) will become economically necessary. This fine-grain disabling will lead to degraded performance compared to a fault-free execution.

A common implicit assumption in all static worst-case execution time (WCET) estimation methods is that the target processor is not subject to faults. Their result is not safe anymore when using fine grain disabling of hardware components, which degrades performance.

In [16] a method that statically calculates a probabilistic WCET bound in the presence of permanent faults in instruction caches is provided. The method, from a given program, cache configuration and probability of cell failure, derives a probabilistic WCET bound. An essential benefit of our approach is that its probabilistic nature stems only from the probability associated with the presence of faults. By construction, the worst-case execution path cannot be missed, since it is determined using static analysis, extended to cope with permanent faults. This allows our method to be used in safety-critical real-time systems. The method is computationally tractable, since it avoids the exhaustive enumeration of all possible fault locations. Experimental results show that the proposed method accurately estimates WCETs in the presence of permanent faults compared to a method that explores all possible locations for faults. On the one hand, the proposed method allows to quantify the impact of permanent faults on WCET estimates for chips with a known probability of cell failure for the whole chip lifetime. On the other hand, and most importantly, our work can also be used in architectural exploration frameworks to select the most appropriate fault management mechanisms, for current and future chip designs.

6.4.3. *Traceability of flow information for WCET estimation*

Participants: Hanbing Li, Isabelle Puaut, Erven Rohou.

This research is part of the ANR W-SEPT project.

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

The objective of this work is to address the challenging issue of the mapping and transformation of the flow information from high level down to machine code. In our recent work [21], we have proposed a framework to systematically transform flow information from source code to machine code.

The framework defines a set of formulas to transform flow information for standard compiler optimizations. Transforming the flow information is done within the compiler, in parallel with transforming the code. There thus is no guessing what flow information have become, it is transformed along with the code. The framework is general enough to cover all linear flow constraints and all typical optimizations implemented in modern compilers. Our implementation in the LLVM compiler shows that we can improve the WCET of Malardalen benchmarks by 60% in average (up to 86%) by turning on optimizations. We also provide new insight on the impact of existing optimizations on the WCET.

6.4.4. *Verified WCET estimation*

Participant: Isabelle Puaut.

This is joint work with Andre Oliveira Maroneze, David Pichardie and Sandrine Blazy from the Celtique group at Inria/IRISA Rennes.

Current WCET estimation tools, even when based on sound static analysis techniques, are not verified. This may lead to bugs being accidentally introduced in the implementation. The main contribution of this work [13], [26] is a formally verified WCET estimation tool operating over C code.

Our tool is integrated to the formally verified CompCert C compiler. It is composed of two main parts: a loop bound estimation and an Implicit Path Enumeration Technique (IPET)-based WCET calculation method. We evaluated the precision of the WCET estimates on a reference benchmark and obtained results which are competitive with state-of-the-art WCET estimation techniques. The code of our tool is automatically generated from its formal specification. Furthermore, machine-checked proofs ensure the estimated WCET is at least as large as the actual WCET.

6.5. Computer arithmetic

Participant: Sylvain Collange.

6.5.1. Application-specific number systems

Collaboration with Mark G. Arnold, XLNS Research, USA.

Reconfigurable FPGA platforms let designers build efficient application-specific circuits, when the performance or energy efficiency of general-purpose CPUs is insufficient, and the production volume is not enough to offset the very high cost of building a dedicated integrated circuit (ASIC). One way to take advantage of the flexibility offered by FPGAs is to tailor arithmetic operators for the application. In particular, the Logarithmic Number System (LNS) is suitable for embedded applications dealing with low-precision, high-dynamic range numbers.

Like floating-point, LNS can represent numbers from a wide dynamic range with constant relative accuracy. However, while standard floating-point offer so-called subnormal numbers to represent numbers close to zero with constant absolute accuracy, LNS numbers abruptly overflow to zero, resulting in a gap in representable numbers close to zero that can impact the accuracy of numerical algorithms.

In collaboration with Mark G. Arnold, Sylvain Collange proposed a generalization of LNS that incorporates features analogous to subnormal floating-point [14]. The Denormal LNS (DLNS) system we introduce defines a class of hybrid number systems that offer quasi-constant absolute accuracy close to zero and quasi-constant relative accuracy on larger numbers. These systems can be configured to range from pure LNS (constant relative accuracy) to fixed-point (constant absolute accuracy across the whole range).

6.5.2. Deterministic floating-point primitives for high-performance computing

Parallel algorithms such as reduction are ubiquitous in parallel programming, and especially high-performance computing. Although these algorithms rely on associativity, they are use on floating-point data, on which operations are not associative. As a result, computations become non-deterministic, and the result may change according to static and dynamic parameters such as machine configuration or task scheduling.

In collaboration with David Defour (UPVD), Stef Graillat and Roman Iakymchuk (LIP6), we introduce a solution to compute deterministic sums of floating-point numbers efficiently and with the best possible accuracy. A multi-level algorithm incorporating a filtering stage that uses fast vectorized floating-point expansions and an accumulation stage based on super-accumulators in a high-radix carry-save representation guarantees accuracy to the last bit even on degenerate cases while maintaining high performance in the common cases [35].

7. Bilateral Contracts and Grants with Industry

7.1. Bilateral Contracts with Industry

7.1.1. Intel research grant ALF-INTEL2014-8957

Participant: André Seznec.

Intel is supporting the research of the ALF project-team on "Mixing branch and value prediction to enable high sequential performance".

8. Partnerships and Cooperations

8.1. National Initiatives

8.1.1. Capacités: Projet "Investissement d'Avenir", 1/11/14 to 31/01/2018

Participants: Damien Hardy, Isabelle Puaut.

The project objective is to develop a hardware and software platform based on manycore architectures, and to demonstrate the relevance of these manycore architectures (and more specifically the Kalray manycore) for several industrial applications. The Kalray MPPA manycore architecture is currently the only one able to meet the needs of embedded systems simultaneously requiring high performance, lower power consumption, and the ability to meet the requirements of critical systems (low latency I/O, deterministic processing times, and dependability). The project partners are Kalray (lead), Airbus, Open-Wide, Safran Sagem, IS2T, Real Time ar Work, Dassault Aviation, Eurocopter, MBDA, Supersonic Imagine, ProbaYes, IRIT, Onera, Verimag, Inria, Irisa, Tima and Armines.

8.1.2. Inria Project Lab: Multicore 2013-2016

Participants: Erven Rohou, Alain Ketterlin, Nabil Hallou.

The Inria Project Lab (formerly *Action d'Envergure*) started in 2013. It is entitled "Large scale multicore virtualization for performance scaling and portability". Partner project-teams include: ALF, ALGORILLE, CAMUS, REGAL, RUNTIME, as well as DALI. This project aims to build collaborative virtualization mechanisms that achieve essential tasks related to parallel execution and data management. We want to unify the analysis and transformation processes of programs and accompanying data into one unique virtual machine.

8.1.3. ADT IPBS 2013-2015

Participants: Sylvain Collange, Erven Rohou, André Seznec, Thibault Person.

As multi-core CPUs and parallel accelerators become pervasive, all execution platforms are now parallel. Research on architecture, compilers and systems now focuses on parallel platforms. New contributions need to be validated against parallel applications that are expected to be representative of current or future workloads. The research community relies today on a few benchmarks sets (SPLASH, PARSEC ...) Existing parallel benchmarks are scarce, and some of them have issues such as aging workloads or non-representative input sets. The IPBS initiative aims at leveraging the diversity of parallel applications developed within Inria to provide a set of benchmarks, named the Inria Parallel Benchmark Suite, to the research community.

8.1.4. ADT Padrone 2012–2014

Participants: Erven Rohou, Alain Ketterlin, Emmanuel Riou.

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

8.1.5. ANR W-SEPT 2012-2015

Participants: Hanbing Li, Isabelle Puaut, Erven Rohou.

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

8.2. European Initiatives

8.2.1. FP7 & H2020 Projects

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

Type: IDEAS

Instrument: ERC Advanced Grant

Duration: April 2011 - March 2016

Coordinator: André Seznec

Inria contact: André Seznec

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

8.2.1.2. HiPEAC3 NoE

Participants: Pierre Michaud, Erven Rohou, André Seznec.

P. Michaud, A. Seznec and E. Rohou are members of the European Network of Excellence HiPEAC3. HiPEAC3 addresses the design and implementation of high-performance commodity computing devices in the 10+ year horizon, covering both the processor design, the optimizing compiler infrastructure, and the evaluation of upcoming applications made possible by the increased computing power of future devices.

8.2.2. Collaborations in European Programs, except FP7 & H2020

8.2.2.1. COST Action TACLe - Timing Analysis on Code-Level (http://www.tacle.eu) 10-2012/09-2015

Participants: Damien Hardy, Isabelle Puaut.

Embedded systems increasingly permeate our daily lives. Many of those systems are business- or safety-critical, with strict timing requirements. Code-level timing analysis (used to analyze software running on some given hardware w.r.t. its timing properties) is an indispensable technique for ascertaining whether or not these requirements are met. However, recent developments in hardware, especially multi-core processors, and in software organization render analysis increasingly more difficult, thus challenging the evolution of timing analysis techniques.

New principles for building "timing-composable" embedded systems are needed in order to make timing analysis tractable in the future. This requires improved contacts within the timing analysis community, as well as with related communities dealing with other forms of analysis such as model-checking and type-inference, and with computer architectures and compilers. The goal of this COST Action is to gather these forces in order to develop industrial-strength code-level timing analysis techniques for future-generation embedded systems, through several working groups:

- WG1 Timing models for multi-cores and timing composability
- WG2 Tooling aspects
- WG3 Early-stage timing analysis
- WG4 Resources other than time

Isabelle Puaut is in the management committee of the COST Action TACLe - Timing Analysis on Code-Level (http://www.tacle.eu). She is responsible of Short Term Scientific Missions (STSM) within TACLe.

8.3. International Initiatives

8.3.1. Participation In International Programs

8.3.1.1. UFGM Chair (Brasil)

Program: Cátedras Francesas UFMG

Title: Compiler Support for emerging parallel architectures

Inria principal investigator: Sylvain Collange

International Partner (Institution - Laboratory - Researcher):

Universidade Federal de Minas Gerais (UFMG) - Computer Science Department - Fernando Pereira

Duration: Sep 2014 - Dec 2014

We propose . The project develop compilation techniques for code optimization to speedup applications that run in Graphics Processing Units (GPUs). The objective is to enable developers code high-performance programs in high-level languages, while taking maximum benefit from the hardware. In particular, we seek to alleviate control and memory divergence, which are important performance limiters specific to GPU architectures. For instance, the call fusion optimization factors out a common function call invoked from multiple independent conditional branches to enable the hardware to execute the function in SIMD mode regardless of branch divergence.

8.3.2. Informal collaborations

The ALF project-team has informal collaborations (visits, common publications) with University of Wisconsin at Madison (Pr Wood), University of Toronto (Pr Moshovos), University of Ghent (Dr Eyerman), University of Upsalla (Pr Hagersten), University of Cyprus (Pr Sazeides), the Egyptian-Japanese University of Science and Technology (Pr Ahmed El-Mahdy).

8.4. International Research Visitors

8.4.1. Visits of International Scientists

- Dr Stijn Eyerman from University of Ghent has been visiting the ALF project-team in April-May 2014.
- Pr Erik Hagerstern from Uppsala University has been visiting the ALF project-team in September-December 2014
- Pr Fernando Magno Quintão Pereira, from the Federal University of Minas Gerais visited the ALF project for 1 week in January 2014.

8.4.2. Visits to International Teams

Sylvain Collange has been invited on a professor chair at Universidade Federal de Minas Gerais, Brasil (September-December 2014). The subject of the collaboration is "Compiler Support for emerging parallel architectures".

9. Dissemination

9.1. Promoting Scientific Activities

9.1.1. Scientific events organisation

9.1.1.1. Member of the organizing committee

- Isabelle Puaut is a member of the Executive committee of IEEE Technical Committee on Real-Time Systems
- Isabelle Puaut is member of the Steering committees of ECRTS and WCET conferences.
- Isabelle Puaut is the chair of the RTNS steering committee.
- Isabelle Puaut and Damien Hardy are organizing the Ecole d'été Temps Réel in Rennes in August 2015.
- Sylvain Collange is a member of the organization committee of Rencontres Arithmétiques de l'Informatique Mathématique (RAIM) 2015 in Rennes.

9.1.2. Scientific events selection

9.1.2.1. Member of the conference program committee

- Isabelle Puaut was a member of RTSS 2014, RTAS 2014, RTCSA 2014 and SIES 2014 program committees. She is a member of the RTAS 2015 and WCET 2015 program committees.
- Damien Hardy was a member of RTNS 2014 and WCET 2014 program committees. He is a member of the WCET 2015 program committee.
- André Seznec was a member of the MICRO 2014 top picks committee, a member of SAMOS 2014 program committee and HPCA 2015 committee.
- Erven Rohou was a member of the program committee of DITAM-PARMA 2015
- Pierre Michaud was a member of the program committees of the HPCC 2014 conference and the OMHI 2014 and CBP4 workshops.
- Sylvain Collange was a member of the program committee of the "Conférence en Parallélisme, Architecture et Système"(ComPas) 2014, Neuchâtel, Suisse.

9.1.3. Journal

9.1.3.1. Member of the editorial board

- André Seznec is a member of the editorial board of the IEEE Micro.

9.2. Teaching - Supervision - Juries

9.2.1. Teaching

Master: A. Seznec, P. Michaud, A. Perais, Architecture avancée, 36 hours, M2, Ecole Supérieure d'Ingénieurs de Rennes, France

Master: I. Puaut, D. Hardy, Operating systems - process management, 130 hours, M1, Université de Rennes I, France

Master: I. Puaut, Système d'exploitation gestion mémoire, 39 hours, M1, Université de Rennes I, France

Master: I. Puaut, D. Hardy, Systèmes temps-réel, 69 hours, M1, Université de Rennes I, France

Master: D. Hardy, A. Perais, Systèmes d'exploitation, 44 hours, M1, Université de Rennes I, France

Licence: D. Hardy, Informatique temps-réel, 40 hours, L3, Université de Rennes I, France

Master: S. Collange, Programmation parallèle, 22 hours, M1, Université de Rennes I, France

Master/PhD: S. Collange, GPU programming, 30 hours, MSc/PhD programs, Universidade Federal de Minas Gerais, Brasil

9.2.2. Supervision

PhD in progress: Nabil Hallou, Université Rennes 1, Feb 2013, co-advisors E. Rohou and P. Clauss (EPI Camus Inria Strasbourg)

PhD in progress: Sajith Kalathingal, Université Rennes 1, Dec 2012, co-advisors S. Collange and A. Seznec

PhD in progress: Surya Khizakanchery Natarajan, Université Rennes 1, Jan 2012, advisor A. Seznec

PhD in progress: Hanbing Li, Université Rennes 1, Oct 2012, co-advisors E. Rohou and I. Puaut

PhD in progress: Andrea Mondelli, Université Rennes 1, Oct 2013, co-advisors P. Michaud and A. Seznec

PhD in progress: Bharath Narasimha Swamy, Université Rennes 1, Sept 2011, advisor A. Seznec

PhD in progress: Arthur Perais, Université Rennes 1, Sept 2012, advisor A. Seznec

PhD in progress: Aswinkumar Sridharan, Université Rennes 1, Oct 2013, advisor A. Seznec

PhD in progress: Arjun Suresh , Université Rennes 1, Dec 2012, co-advisors E. Rohou and A. Seznec

9.2.3. Dissemination

- Erven Rohou and Damien Hardy taught labs on "Compilation, Optimization, Debug methods" at "Ecole Jeune Chercheurs en programmation" in Rennes, June 2014
- Nabil Hallou presented "Dynamic re-vectorization of binary code", at Huitièmes rencontres de la communauté française de compilation, Jul 2014.
- Emmanuel Riou presented "Padrone: a dynamic binary modification tool" at the Intel Compiler, ARchitecture and Tools conference, at Haifa, Israel, Dec 2014
- Damien Hardy presented "Worst Case Execution Time Estimation and Permanent Faults" in the workshop on Challenges in Mixed Criticality and Real-time and Reliability in Networked Complex Embedded Systems at the Hipeac Computer System Week, May 2014.
- André Seznec and Arthur Perais presented their work on Value Prediction at a seminar at Qualcomm, Raleigh, in February 2014.
- André Seznec presented the work on Value Prediction at Intel, Hillsboro in November 2014.
- Arthur Perais was invited to present its work on Value Prediction at the COMPASS conference, Neufchatel, April 2014, http://compas2014.unine.ch/.

9.3. Miscelleanous

- Erven Rohou co-advised a MSc. student at the Egypt-Japan University of Science and Technology.
- Erven Rohou is a member of the Inria CDT (Commission du Développement Technologique)
- As “correspondant scientifique des relations internationales” for Inria Rennes Bretagne Atlantique, Erven Rohou is a member of the Inria COST GTRI (Groupe de Travail "Relations Internationales" du Comité d’Orientation Scientifique et Technologique).
- A. Seznec was an elected member of the scientific committee of Inria till November 2014.
- A. Seznec is an elected member of the administration board of Inria since November 2014.

10. Bibliography

Major publications by the team in recent years

[1] 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, pp. 130–144

[2] 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, pp. 99–104

[3] 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

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

[5] 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 p.

[6] 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, pp. 35-58

[7] N. PRÉMILLIEU, A. SEZNEC. *Efficient Out-of-Order Execution of Guarded ISAs*, Inria, November 2013, n^{o} RR-8406, 24 p. , http://hal.inria.fr/hal-00910335

[8] E. ROHOU, B. NARASIMHA SWAMY, A. SEZNEC. *Branch Prediction and the Performance of Interpreters - Don't Trust Folklore*, Inria, November 2013, n^{o} RR-8405, 23 p. , http://hal.inria.fr/hal-00911146

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

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

[11] 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

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

Publications of the year

Doctoral Dissertations and Habilitation Theses

[13] A. OLIVEIRA MARONEZE. *Certified Compilation and Worst-Case Execution Time Estimation*, Université Rennes 1, June 2014, https://tel.archives-ouvertes.fr/tel-01064869

Articles in International Peer-Reviewed Journals

[14] G. ARNOLD, S. COLLANGE. *Options for Denormal Representation in Logarithmic Arithmetic*, in "Journal of VLSI Signal Processing-Systems for Signal, Image, and Video Technology", March 2014, vol. 77, n^{o} 1-2, pp. 207-220 [*DOI :* 10.1007/s11265-014-0874-3], https://hal.inria.fr/hal-01087059

[15] S. EYERMAN, P. MICHAUD, W. ROGIEST. *Multiprogram Throughput Metrics: A Systematic Approach*, in "ACM Transactions on Architecture and Code Optimization", October 2014, vol. 11, n^{o} 3, 26 p. [*DOI :* 10.1145/2663346], https://hal.archives-ouvertes.fr/hal-01087743

[16] D. HARDY, I. PUAUT. *Static Probabilistic Worst Case Execution Time Estimation for Architectures with Faulty Instruction Caches*, in "Real-Time Systems", 2014, 25 p. , https://hal.inria.fr/hal-01086884

[17] T. MILANEZ, S. COLLANGE, F. MAGNO QUINTÃO PEREIRA, W. MEIRA, A. FERREIRA. *Thread scheduling and memory coalescing for dynamic vectorization of SPMD workloads*, in "Parallel Computing", October 2014, vol. 40, n^{o} 9, pp. 548–558 [*DOI :* 10.1016/J.PARCO.2014.03.006], https://hal.inria.fr/hal-01087054

[18] N. PRÉMILLIEU, A. SEZNEC. *Efficient Out-of-Order Execution of Guarded ISAs*, in "ACM Transactions on Architecture and Code Optimization (TACO) ", December 2014, 21 p. [*DOI :* 10.1145/2677037], https://hal.inria.fr/hal-01103230

International Conferences with Proceedings

[19] J. ABELLA, D. HARDY, I. PUAUT, E. QUINONES, F. J. CAZORLA. *On the Comparison of Deterministic and Probabilistic WCET Estimation Techniques*, in "26th Euromicro Conference on Real-Time Systems", Madrid, Spain, July 2014, https://hal.inria.fr/hal-01086875

[20] S. ELSHOBAKY, A. EL-MAHDY, E. ROHOU, L. EL-SAYED, N. ELDERINI. *A lightweight incremental analysis and profiling framework for embedded devices*, in "Proceedings of the 17th International Workshop on Software and Compilers for Embedded Systems", Sankt-Goar, Germany, June 2014, pp. 60-68 [*DOI :* 10.1145/2609248.2609263], https://hal.inria.fr/hal-01086903

[21] H. LI, I. PUAUT, E. ROHOU. *Traceability of Flow Information: Reconciling Compiler Optimizations and WCET Estimation*, in "RTNS - 22nd International Conference on Real-Time Networks and Systems", Versailles, France, October 2014 [*DOI :* 10.1145/2659787.2659805], https://hal.inria.fr/hal-01072138

[22] P. MICHAUD. *Five poTAGEs and a COLT for an unrealistic predictor*, in "4th JILP Workshop on Computer Architecture Competitions (JWAC-4): Championship Branch Prediction (CBP-4)", Minneapolis, United States, June 2014, https://hal.archives-ouvertes.fr/hal-01087692

[23] P. MICHAUD, A. SEZNEC. *Pushing the branch predictability limits with the multi-poTAGE+SC predictor* , in "4th JILP Workshop on Computer Architecture Competitions (JWAC-4): Championship Branch Prediction (CBP-4)", Minneapolis, United States, June 2014, https://hal.archives-ouvertes.fr/hal-01087719

[24] B. NARASIMHA SWAMY, A. KETTERLIN, A. SEZNEC. *Hardware/Software Helper Thread Prefetching On Heterogeneous Many Cores*, in "2014 IEEE 26th International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD)", Paris, France, October 2014 [*DOI :* 10.1109/SBAC-PAD.2014.39], https://hal.inria.fr/hal-01087752

[25] S. NARAYANAN, B. NARASIMHA SWAMY, A. SEZNEC. *Impact of serial scaling of multi-threaded programs in many-core era*, in "WAMCA - 5th Workshop on Applications for Multi-Core Architectures", Paris, France, October 2014 [*DOI :* 10.1109/SBAC-PADW.2014.9], https://hal.archives-ouvertes.fr/hal-01089446

[26] A. OLIVEIRA MARONEZE, S. BLAZY, D. PICHARDIE, I. PUAUT. *A Formally Verified WCET Estimation Tool*, in "14th International Workshop on Worst-Case Execution Time Analysis", Madrid, Spain, July 2014 [*DOI :* 10.4230/OASIcs.WCET.2014.11], https://hal.inria.fr/hal-01087194

[27] R. OMAR, A. EL-MAHDY, E. ROHOU. *Arbitrary control-flow embedding into multiple threads for obfuscation: a preliminary complexity and performance analysis*, in "Proceedings of the 2nd international workshop on Security in cloud computing", Kyoto, Japan, June 2014 [*DOI :* 10.1145/2600075.2600080], https://hal.inria.fr/hal-01086958

[28] A. PERAIS, A. SEZNEC. *EOLE: Paving the Way for an Effective Implementation of Value Prediction*, in "International Symposium on Computer Architecture", Minneapolis, MN, United States, ACM/IEEE, June 2014, vol. 42, pp. 481 - 492 [*DOI :* 10.1109/ISCA.2014.6853205], https://hal.inria.fr/hal-01088130

[29] A. PERAIS, A. SEZNEC. *Practical data value speculation for future high-end processors*, in "International Symposium on High Performance Computer Architecture", Orlando, FL, United States, IEEE, February 2014, pp. 428 - 439 [*DOI :* 10.1109/HPCA.2014.6835952], https://hal.inria.fr/hal-01088116

[30] E. RIOU, E. ROHOU, P. CLAUSS, N. HALLOU, A. KETTERLIN. *PADRONE: a Platform for Online Profiling, Analysis, and Optimization*, in "DCE 2014 - International workshop on Dynamic Compilation Everywhere", Vienne, Austria, January 2014, https://hal.inria.fr/hal-00917950

[31] E. ROHOU, B. NARASIMHA SWAMY, A. SEZNEC. *Branch Prediction and the Performance of Interpreters - Don't Trust Folklore*, in "International Symposium on Code Generation and Optimization", Burlingame, United States, February 2015, https://hal.inria.fr/hal-01100647

[32] S. SARDASHTI, A. SEZNEC, D. A. WOOD. *Skewed Compressed Cache*, in "47th Annual IEEE/ACM International Symposium on Microarchitecture", Cambridge, United Kingdom, December 2014, https://hal.inria.fr/hal-01088050

[33] A. SEZNEC. *TAGE-SC-L Branch Predictors*, in "JILP - Championship Branch Prediction", Minneapolis, United States, June 2014, https://hal.inria.fr/hal-01086920

Research Reports

[34] G. ARNOLD, S. COLLANGE. *Options for Denormal Representation in Logarithmic Arithmetic*, January 2014, n^{o} RR-8412, 27 p. , https://hal.inria.fr/hal-00909096

Other Publications

[35] S. COLLANGE, D. DEFOUR, S. GRAILLAT, R. IAKYMCHUK. *Full-Speed Deterministic Bit-Accurate Parallel Floating-Point Summation on Multi- and Many-Core Architectures*, February 2014, https://hal.archives-ouvertes.fr/hal-00949355

[36] R. IAKYMCHUK, D. DEFOUR, S. COLLANGE, S. GRAILLAT. *Reproducible and Accurate Matrix Multiplication for GPU Accelerators*, January 2015, https://hal.archives-ouvertes.fr/hal-01102877

References in notes

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

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

[39] 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, pp. 186–195, http://doi.acm.org/10.1145/300979.300995

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

[41] M. D. HILL, M. R. MARTY. *Amdahl's Law in the Multicore Era*, in "IEEE Computer", 2008

[42] 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 p. , http://doi.ieeecomputersociety.org/10.1109/ISCA.2004.1310786

[43] 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

[44] 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, pp. 319–324, http://doi.acm.org/10.1145/1391469.1391551

[45] 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

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

[47] S. SARDASHTI, D. A. WOOD. *Decoupled compressed cache: exploiting spatial locality for energy-optimized compressed caching*, in "The 46th Annual IEEE/ACM International Symposium on Microarchitecture, MICRO-46, Davis, CA, USA, December 7-11, 2013", 2013, pp. 62–73, http://doi.acm.org/10.1145/2540708.2540715

[48] 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, pp. 45–57

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

[50] J. G. STEFFAN, C. COLOHAN, A. ZHAI, T. C. MOWRY. *The STAMPede approach to thread-level speculation*, in "ACM Transactions on Computer Systems", 2005, vol. 23, n° 3, pp. 253–300, http://doi.acm.org/10.1145/1082469.1082471

[51] 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, pp. 300–303, http://doi.acm.org/10.1145/1391469.1391545

[52] 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

[53] 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, pp. 80-89