

An Abstraction Methodology for the Evaluation of Multi-core Multi-threaded Architectures

Ruken Zilan
Computer Science
(CS) Dept.
Barcelona
Supercomputing
Center (BSC)
Barcelona, Spain

Javier Verdú,
Jorge García
Computer Architecture
(CA) Dept.
Universitat Politècnica
de Catalunya (UPC)
Barcelona, Spain

Mario Nemirovsky
ICREA Professor at
CS Dept., BSC
Barcelona, Spain

Rodolfo A. Milito
CISCO Systems
San Francisco, CA,
USA

Mateo Valero
CS Dept., BSC
&
CA Dept., UPC
Barcelona, Spain

Abstract— As the evolution of multi-core multi-threaded processors continues, the complexity demanded to perform an extensive trade-off analysis, increases proportionally. Cycle-accurate or trace-driven simulators are too slow to execute the large amount of experiments required to obtain indicative results. To achieve a thorough analysis of the system, software benchmarks or traces are required. In many cases when an analysis is needed most, during the earlier stages of the processor design, benchmarks or traces are not available. Analytical models overcome these limitations but do not provide the fine grain details needed for a deep analysis of these architectures.

In this work we present a new methodology to abstract processor architectures, at a level between cycle-accurate and analytical simulators. To apply our methodology we use queueing modeling techniques. Thus, we introduce Q-MAS, a queueing based tool targeting a real chip (the UltraSPARC T2 processor) and aimed at facilitating the quantification of trade-offs during the design phase of multi-core multi-threaded processor architectures. The results demonstrate that Q-MAS, the tool that we developed, provides accurate results very close to the actual hardware, with a minimal cost of running what-if scenarios.

Keywords-component; Fine grain modeling; a methodology to build simulators; a simulation tool for multi-levels of shared resource architecture modeling; UltraSPARC T2, queueing modeling; “what-if”s for CPU architecture.

I. INTRODUCTION

The lack of having enough instruction-level parallelism has motivated the appearance of thread level parallelism (TLP) as a common strategy to improve processor performance on a single chip. Current multi-threaded architectures exploit different paradigms of TLP. On the one hand, threads running on a single core (i.e. FGMT, CGMT, SMT architectures) share most of the hardware resources. On the other hand, threads running on a CMP processor share only some levels of the cache memory hierarchy. Each of these designs offers different benefits as they take advantage of TLP in different ways. This leads processors’ vendors to combine several TLP paradigms in a single chip, such as pure-SMT and pure-CMP. Thus, most of massive multi-threaded processors have multiple threads per core and multiple cores per chip.

There are different computing areas that demand a massive exploitation of TLP, as the processors that execute network applications. As these applications evolve towards more

complex packet processing (e.g. deep packet inspection, intrusion detection), the processors have to provide higher throughput to sustain the processing of high bandwidth network traffic. Massive multi-threaded architectures exploit the TLP required by this computing area, but introduce new complexities to be analyzed at both hardware (e.g. efficient use of processor resources) and software levels (e.g. optimization of multi-threaded code).

Researchers normally use execution or trace driven simulators to analyze processors performance. However, such tools present a large number of drawbacks (e.g. costly development of simulators and large amount of time requirement to run the experiments, especially in multi-threaded architectures [1]). Several proposals reduce run times by executing particular sections of the application. For instance, the interval simulator [2] simulates few sections of the actual input execution trace. Similar approaches are taken in statistical [3, 4, 5], sampled [6, 7], and co-phase matrix methods [8].

Nonetheless, in many cases the workload (i.e. benchmarks and/or useful network traffic traces) is not available, especially in the early phases of new systems development, in which case simulations cannot be done. To overcome such limitations, researchers use other approaches like analytical models. Few analytical solutions have been published for single-threaded [9, 10, 11] and multi-threaded architectures [12, 13, 14, 15]. Other studies perform coarse grain analyses for multi-threaded systems [16, 17, 18, 19]. Nevertheless, none of these models allow researchers doing fine grain analysis of particular units/issues of the processor architecture. Besides, they do not integrate changes in the behavior of code execution (e.g. changes in the instruction mix during the processing).

We present our work-in-progress that introduces a methodology to abstract significant resources of multi-core multi-threaded architectures to build a queueing model simulator. The abstraction layer is focused on the fine-grain characterization of the architecture, as well as combining time-variant characteristics of application execution and incoming network traffic. To our knowledge, this is the first work that defines queueing based techniques to model current and future massive multi-threaded architectures at fine-grain, especially integrating three different elements of the environment (i.e. processor architecture, application, and network traffic).

This work has been supported by the Ministry of Science and Technology of Spain under contract TIN-2007-60625, the HiPEAC European Network of Excellence and a Collaboration Agreement between Cisco Systems and BSC.

Moreover, we use queueing modeling to apply the proposed methodology in a build-in-progress simulator, called Q-MAS (Queueing Modeling for Analyses & Simulations). Q-MAS aims at doing research on architectural tradeoffs in the early processor design/analysis phases. Analytical models either make too many assumptions or they are pretty complex. Then the complexity of an analytical model can be very high due to the number of parameters and features for modeling a massive multithreaded architecture with multiple levels of hardware resource sharing levels. The key idea of the tool is to provide a deep understanding of the multi-core multi-threaded limitations and architectural trade-offs easily, while reducing the cost and complexity of cycle-accurate simulations, as well as reducing the exponential complexity and removing the rough assumptions of analytical solutions.

Unlike common architectural simulators, our tool does not require programming or executing any benchmarks. Instead, researchers set up input parameters of the architecture together with the benchmark and network traffic scenarios to be analyzed. In fact, one of the main advantages of the tool is the simplicity of doing what-ifs and overcoming the lack of real code and/or network traffic availability. Furthermore, it provides detailed characterization of architectures (e.g. pipelined/non-pipelined execution units, performance per thread/core, multiple levels of hardware resource sharing), unlike other queueing systems and analytical models that are based on a simpler representation of the processor.

Finally, we validate our design methodologies comparing the results obtained by Q-MAS to the ones taken from real hardware (the UltraSPARC T2 processor [20]). Our analyses demonstrate that the results are close to the real performance executing particular stressing benchmarks (i.e. workloads that stress a particular hardware unit) and with reduced error rates (less than 2.3%) executing benchmarks with representative instruction set mix.

II. THE PROPOSED METHODOLOGY

In our methodology, we follow a bottom-up approach in designing the system. We identify and classify the main hardware shared resources that we want to model from the lowest sharing level to the highest sharing level. Later, we model each resource (or component) independently, according to their characteristics (i.e. pipelined or non-pipelined), including thread selection/scheduling policies, and issue limitations constraints. We show some of the dependencies, such as shared hardware resource contention effects, with the interconnections among the components. It is important to emphasize that with the suggested representations we give the total execution time for each instruction without the need of running all pipeline execution stages of processors.

The procedure below summarizes the steps used to apply our methodology to abstract the characteristics of a multi-core multi-threaded architecture:

1. Classify the available sharing levels of hardware resources in the processor to obtain a hierarchical classification.

2. Identify the main hardware resources to be abstracted and modeled, of the current hierarchical level. Determine whether each unit of the current hierarchical level is pipelined or non-pipelined, the cycles taken by the execution, and the issue limitations (e.g. processing two requests in parallel). Cache units are exceptions, in which we define a given average hit rate. Finally, select the proper mathematical approach for modeling each resource.
3. For each resource of the current hierarchical level, define the customer selection/scheduling policy (e.g. priority based). Also, identify policies that may require additional modeling.
4. Model the interconnections of structures among units that belong to the current and to the next hierarchical level.
5. Go to the next hierarchical level and start again from the step 2, till the last hierarchical level is reached.

We also designate the applications in the model (e.g. instruction mix, distribution of instructions, number of software threads) defining by customer classes that will be processed in the queueing system. That is, each customer that will be processed in a queue symbolizes an instruction of a given type.

The application is abstracted following the next steps:

1. Define the amount of software threads available in the code and select the proper modeling for a given thread.
2. Establish the instruction mix (or types) of the application, as well as the inter-arrival distribution of each instruction type. All threads can present the same or different instruction mixes. Such characterizations determine how many different types of customers we need in our model.

Finally, the incoming network traffic is abstracted as follows:

1. Determine the inter-arrival time distribution and burstiness of incoming packets.
2. Define the packet features that lead different executions of the code (e.g. packet size, burstiness of TCP flows).

III. THE TOOL AND VALIDATIONS

We use queueing techniques to develop a model that simulates the behavior of a multi-core multi-threaded architecture: the Q-MAS tool. To build Q-MAS, we use the JMT framework [21], which is a queueing network model simulator.

We apply the proposed abstraction methodology to the UltraSPARC T2 (the SUN T2) processor [20]. It is the first commercial multi-core multi-threaded processor that comprises three sharing levels of hardware resources. The model is designed as a combination of blocks representing the abstracted architectural structures. Although we use diverse queueing techniques to develop the different architectural elements, in this paper we show one of them as an example.

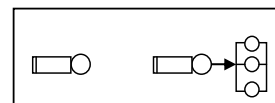
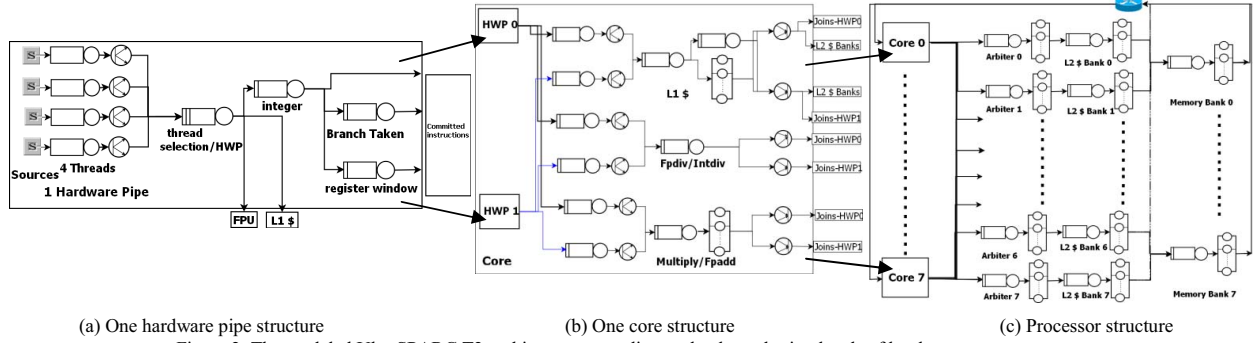


Figure 1. Non-pipelined and pipelined structures



(a) One hardware pipe structure (b) One core structure (c) Processor structure
 Figure 2. The modeled UltraSPARC T2 architecture according to the three sharing levels of hardware resources

Fig. 1 presents the queuing modeling of non-pipelined (on the left) and pipelined (on the right) structures. Non-pipelined execution units (e.g. floating point divider) are modeled using a queue with a single server. It can serve a single customer at the time with a service time of N time units (cycles), where N is the time required to process the particular instruction type. On the other hand, pipelined execution units (e.g. integer multiplier) are modeled with a 1-cycle non-pipelined unit followed by a delay station (infinite servers) of $N-1$ cycles. Some structures (e.g. L2 cache banks) accept one request every X cycles, where $X > 1$. To model such blocks, the non-pipelined front-end presents a latency of X cycles, while the delay station shows a latency of $N-X$ cycles. We call these blocks hybrid structures.

Following the proposed abstraction methodology, we define the sharing levels of the T2 architecture. The UltraSPARC T2 processor consists of eight cores connected through a crossbar to a shared L2 cache. Each of the cores supports eight hardware contexts (strands), altogether executing 64 threads simultaneously. Strands inside of a hardware core are divided into two groups of four strands, forming two hardware execution pipelines (hardware pipes). Thus, the resources of the processor are shared among three different levels as in [22], from the inner to outer; IntraPipe, among strands running in the same hardware pipeline; IntraCore, among strands running on the same core; and InterCore, among strands executing on different cores.

Applying the iterative steps of the methodology described in Section II, we abstract and model (steps 1-4) the resources at the IntraPipe sharing level (Fig. 2.a). The main resources modeled in this level are the integer execution unit, the static branch predictor and the register window. Besides, thread selection policy requires additional modeling. In the next iteration of the methodology, we abstract and model at the IntraCore sharing level (Fig. 2.b), in which 2 hardware pipes share the floating point execution unit (we distinguish different types of operations) and L1 data cache. In this case, we include additional modeling for read-after-write dependencies among memory instructions. Finally, we address the InterCore sharing level (Fig. 2.c), in which 8 cores share the crossbar arbiters connected to L2 cache banks, and these are connected to memory controllers. We parameterize all these blocks according to the UltraSPARC T2 hardware reference manual [20].

We model the instruction execution of running threads as tokens generated according to the defined classes (to identify

different applications) and sub-classes (to distinguish the instruction mix). In the current model, we assume the processor executes multiple independent threads, since the current release of Q-MAS does not model software lock dependencies. Thus, multiple threads can run several independent instances of the same application and/or they can execute different applications.

Q-MAS can be represented by an open or a closed queuing network, depending on the complexity of the running applications' instruction mix characteristics. For complex parameterization, we use open queuing network since it allows defining better and detailed execution scenarios. For instance, customer classes can be generated using different inter-arrival time distributions and the required instruction mix can be accurately represented. However, for simple instruction mix properties closed queuing network is good enough.

At this stage, we do not implement the network traffic part of the simulator (see Section IV for more details). For the validation experiments, we assume that a packet is present as soon as needed. Thus, we validate maximum throughput.

We have run a variety of benchmark sets to assess the different components of Q-MAS for isolated and combined cases. Nonetheless, in this paper we present some of the results obtained executing a benchmark based on hash processing of a given value of incoming packets. For all the experiments, we profile the application beforehand to configure Q-MAS according to the benchmark characteristics. Nevertheless, this is only needed to validate the results, and it is not required for common use of Q-MAS, due to the what-if analysis purpose of the tool.

Fig.3 depicts the performance of a given thread (inspected thread) running with other co-scheduled threads. We assume all threads execute different instances of the same hash application. We compare the performances of measured real hardware (light bars) with Q-MAS results (dark bars). The Y-axis indicates the performance of the inspected thread in terms of instructions per cycle (IPC). We remind that the maximum IPC of a given thread in the UltraSPARC T2 processor is 1. The X-axis shows the scenario setup. That is, the distribution of inspected and co-scheduled threads in the processor using the format “ $\{A|B\}/C$ ”, where C is the number of active cores in the processor, and where A and B are the numbers of active threads running in hardware pipe 1 and 2, respectively, of each active core. We show the IPC of the inspected thread, which is running in the hardware pipe 1 of the first active core in the processor.

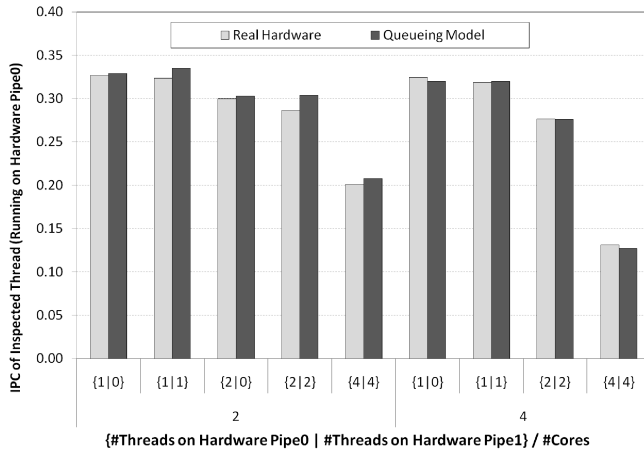


Figure 3. Comparison of performance measured in Real Hardware vs Queuing Model

We can observe in Fig. 3 that the results obtained by queuing model simulations are similar to the performance measured from the real hardware. There is an average error rate of 2.23%, with a minimum of 0.15% (using the $\{2|2\}/4$ configuration) and a maximum of 2.90% (with the setup $\{2|2\}/2$). These error rates are representative of the results we obtain in most of the experiments we run for other benchmarks and scenarios.

IV. FUTURE WORKS

The constraints of JMT framework [21] prevent us to implement different queuing techniques. Thus, the abstraction methodology has not been completely validated, especially the abstraction of several application and network traffic characteristics. Nevertheless, the experiments we do to compare the current Q-MAS and real hardware performances, guarantee the accuracy of the results.

To overcome such problems, we are implementing the Q-MAS using another framework that allows us to apply our methodology. This validation pays special attention to: particular items of the architecture (e.g. fetch unit, out-of-order architectures), software locks, and integration of network traffic characteristics to the queuing model.

V. CONCLUSION

The paper presents a methodology to abstract processor architectures at a level of complexity in between the cycle-accurate and analytical simulators. The methodology is based on a fine grain characterization of each architectural block. The aim of this methodology is to build models with similar speed to analytical models and with the details of the cycle-accurate simulators that allow analyzing trade-offs during the design phase of multi-core/multi-threaded processor architectures.

We introduce Q-MAS, a queuing based simulator that captures most of the essential architectural features of the SUN T2 processor and allows doing varieties of what-if analyses. With the use of this simulator, many architectural modifications can be easily considered. Moreover, it is very flexible since many workloads can be easily tested.

We validate Q-MAS running a set of synthetic, mixed and network workloads on real hardware (the SUN T2 processor). The tool reproduces an almost precise number of cycles needed on the execution of each T2 structure and its contention. Unlike

cycle-accurate micro-architectural simulators which are highly complex and elaborated, our model allows doing any kind of what-if scenarios easily and quickly and also reduces the complexities.

REFERENCES

- [1] J. Vera, F. J. Cazorla, A. Pajuelo, O. J. Santana, E. Fernandez and M. Valero. FAME: FAirly MEasuring Multithreaded Architectures. In *Procs of PACT*. Brasov, Romania, Sept. 2007.
- [2] D. Genbrugge, S. Eyerman, and L. Eeckhout. Interval Simulation: Raising the Level of Abstractions in Architectural Simulation. In *Procs of HPCA*. San Antonio, TX, USA, Feb. 2010.
- [3] L. Eeckhout, R. B. Jr., B. Stougie, K. D. Bosschere, and L. John. Control Flow Modeling in Statistical Simulation for Accurate and Efficient Processor Design Studies. In *Procs of ISCA*. Munchen, Germany, June 2004.
- [4] D. Genbrugge and L. Eeckhout. Chip multiprocessor design space exploration through statistical simulation. *IEEE Trans. Comput.*, 58(12):1668–1681, 2009.
- [5] S. Nussbaum and J. E. Smith. Modeling superscalar processors via statistical simulation. In *Procs of PACT*. Barcelona, Spain, Sept. 2001.
- [6] T. M. Conte, M. A. Hirsch, and K. N. Menezes. Reducing state loss for effective trace sampling of superscalar processors. In *Procs of ICCD*. Austin, TX, USA, Oct. 1996.
- [7] R. E. Wunderlich, T. F. Wenisch, B. Falsafi, and J. C. Hoe. Smarts: accelerating microarchitecture simulation via rigorous statistical sampling. In *Procs of ISCA*. San Diego, CA, USA, June 2003.
- [8] M. Van Biesbrouck, T. Sherwood, and B. Calder. A co-phase matrix to guide simultaneous multithreading simulation. In *Procs of ISPASS*. Austin, TX, USA, Mar. 2004.
- [9] T. Karkhanis and J. Smith. A first-order superscalar processor model. In *Procs of ISCA*. Munchen, Germany, June 2004.
- [10] D. Noonburg and J. Shen. A framework for statistical modeling of superscalar processor performance. In *Procs of HPCA*. San Antonio, TX, USA, Feb. 1997.
- [11] T. M. Taha and S. Wills. An Instruction Throughput Model of Superscalar Processors. *IEEE Transactions on Computers*, 57:389–403, 2008.
- [12] X. E. Chen and T. M. Aamodt. A First-Order Fine-Grained Multithreaded Throughput Model. In *Procs of HPCA*. Raleigh, NC, USA, Feb. 2009.
- [13] S. Hong and H. Kim. An Analytical Model for a GPU Architecture with Memory-level and Thread-level Parallelism Awareness. In *Procs of ISCA*. Austin, TX, USA, June 2009.
- [14] T.-F. Tsuei and W. Yamamoto. Queuing simulation model for multiprocessor systems. In *Computer*, 36(2):58–64, 2003.
- [15] W. Yamamoto, M. Serrano, A. Talcott, R. Wood, and M. Nemirosky. Performance estimation of multistreamed, superscalar processors. In *System Sciences, 1994. Vol. I: Architecture, Proceedings of the Twenty-Seventh Hawaii Internation Conference on*, volume 1, 1994.
- [16] V. Bhaskar. A closed queuing network model with multiple servers for multi-threaded architecture. In *Comput. Commun.*, 31(14):3078–3089, 2008.
- [17] R. H. Saavedra-Barrera, and D. E. Culler. An analytical solution for a markov chain modeling multithreaded execution. TR UCB/CSD-91-623, EECS Dept., University of California, Berkeley, Apr 1991.
- [18] V. Vlassov and R. Ayani. Analytical modeling of multithreaded architectures. *J. Syst. Archit.*, 46(13):1205–1230, 2000.
- [19] A. Navarro, R. Asenjo, S. Tabik, and C. Cascaval. Analytical modeling of pipeline parallelism. In *Procs of PACT*. Raleigh, NC, USA, Sept. 2009.
- [20] OpenSPARCTM T2 Core Microarchitecture Specification, 2007.
- [21] M. Bertoli, G. Casale, and G. Serazzi. JMT: performance engineering tools for system modeling. In *Procs of SIGMETRICS*. Seattle, WA, USA, June 2009.
- [22] V. Cakarevic, P. Radojkovic, J. Verdú, A. Pajuelo, F. J. Cazorla, M. Nemirovsky, and M. Valero. Characterizing the Resource-Sharing Levels in the UltraSPARC T2 Processor. In *Procs of MICRO*. New York, NY, USA, Dec. 2009.