A VISUAL SIMULATION FRAMEWORK FOR SIMULTANEOUS MULTITHREADING ARCHITECTURES

Adrian Florea[1], Alexandru Ratiu[1], Arpad Gellert[1] and Lucian N. Vinţan[1,2]

[1] Computer Engineering Department, "Lucian Blaga" University of Sibiu, Emil Cioran Street, No. 4, 550025 Sibiu, Romania
[2] Academy of Technical Sciences from Romania

E-mail:{adrian.florea, arpad.gellert, lucian.vintan}@ulbsibiu.ro, ratiu_alex@yahoo.com

KEYWORDS
Simulation, Education, Computer Architecture, Simultaneous Multithreading, Benchmarking.

ABSTRACT

The computing systems, and particularly microarchitectures, are in a continuous expansion reaching an unmanageable complexity by the human mind. In order to understand and control this expansion, researchers need to design and implement larger and more complex systems' simulators. In the current paradigm the simulators play the key role in going further, by translating all complex processing mechanisms in relevant and easy to understand information. This paper aims to make a suggestive description of the concepts and principles implemented into a Simultaneous Multithreading Architecture. We introduce the SMTAHSim framework, an educational tool that simulates in an interactive manner the important aspects of this particular microarchitecture. The graphical simulation and the results reporting techniques provide a lot of easy to understand information that outline an expressive image of Simultaneous Multithreading (SMT) processing mechanisms. Our developed software tool facilitates the understanding of theoretical questions, thus allowing students to feel more confident when studying SMT-related issues.

1. INTRODUCTION

The computer science (CS) domain is a very complex one, representing the result of one of the largest and fastest scientific developments known to mankind. This gradual evolution has engaged, during the last six decades, hundreds of bright minds from different fields (mathematics, physics, electronics, automation, and informatics), giving birth to a new science (CS), which has revolutionized everyday lives of the people. However, the main responsible for computers progress are microprocessors. The continuous expansion of microarchitectures has lead to a hard to control and understand complexity explored with the help of larger and more sophisticated software simulators.

Also, in today's world, there is an ever-increasing need for intelligent systems, especially in educational domain. Without modernize our teaching tools in computer architecture, based on the latest research achievements but also on trade, we risk losing contact with the development of computer engineering. Therefore, it is a stringent necessity to develop teaching resources (software simulators) related to a hard kernel of the fundamental disciplines in computer engineering, like computer architecture, compilers, operating systems and computer networks. Developing effective learning tools targeting these disciplines is a continuous challenge.

In this paper we try to give a better understanding of SMT microprocessor architectures by developing a visual simulation framework. Due to the complexity level, we make the learning steps easier, driven by expressive simulations which can provide us, based on the general picture of the system, a detailed one (*top to down* approach). But why SMT architectures request interest? The current microarchitectures have three major limiters (the so called "brick wall" concept):

- *Memory wall* – the increasing gap created between processor clock cycle time and the main memory access time;
- *Instruction Level Parallelism (ILP) wall* – generated by the present-day impossibility to issue a continuously higher number of instructions in parallel;
- *Power wall* – favorized by the frequency scaling as the number of transistors on chip increase.

The SMT architectures come as a solution to the first two limitations by combining the superscalar instruction issue with the multithreading approach. Thus, instructions from multiple threads could be simultaneously issued in a single clock cycle. Latencies that occur in the execution of single threads are bridged by issuing operations of the remaining threads. Other arguments refers to the fact that, although single-core SMT architectures are on the market since 2002 (Intel Pentium 4 Northwood Hyperthreaded) until now – in 2010 Intel released the Core™ i3, i5, i7 with Hyperthreaded technology on each core (Intel 2010) –, in the authors' opinion, there are not efficient pedagogical tools dedicated to teach SMT concepts easier and more intuitively with interactive animation.

Proceedings 25th European Conference on Modelling and Simulation ©ECMS Tadeusz Burczynski, Joanna Kolodziej Aleksander Byrski, Marco Carvalho (Editors)
ISBN: 978-0-9564944-2-9 / ISBN: 978-0-9564944-3-6 (CD)

The fast development of computer science and computer architecture especially, have determined that many software tools, used not long ago in research, are enhanced with an interactive graphical interface and are taught in Computer Architecture courses. The lack of simulators dedicated to simultaneous multithreading architectures used for didactical purposes, despite they are highly used in research goals, represents the starting point of this paper. In order to better achieve this purpose, we try to develop a compact hybrid simulator, which integrates microprocessor instruction stream, branch prediction and cache memory simulation. Judging from educational goal, through this work we propose few new ideas:

- Hybrid simulation (trace- and execution-driven) of a SMT architecture using interactive animation.
- Introducing real branch predictors dedicated to each simulated thread (branch prediction was only statistically generated in other similar simulators (Smullen and Taha 2006)). For example we implemented *gshare* (a two-level adaptive branch predictor (Yeh and Patt 1992)) and two state of the art dynamic predictors: *FPBNP* (a fast path-based neural branch predictor (Jiménez 2003)) and *OGEHL* (Optimized GEometric History Length branch predictor (Seznec 2005)). The last one was classified on 2nd place at World Championship of Branch Prediction (CBP 2004) and received the best practice award for "*the predictor the closest to a possible hardware implementation*". The branch predictors can be used also as a third party lesson / application.
- Introducing a parameterized instruction cache shared between threads (both instruction and data caches were only statistically generated in other similar simulators (Smullen and Taha 2006)).

From a didactical point of view, the developed tool (SMTAHSim) has benefits in the learning process because it helps students to observe the influence of each parameter on the simulation model. The SMTAHSim simulator provides a wider variety of configuration options. Thus, it can be determined how branch prediction accuracy or resource usage varies with input parameters (number of entries in prediction tables, history length, number of bits for weights representation, etc). The execution-driven simulation allows SMTAHSim's tool to give fine-grained results regarding every microarchitectural unit during and at the end of the benchmarks' simulation. All final simulation results are stored in a database and can be used further to generate a large palette of reports regarding units' performance in correlation with almost every parameter. The SMTAHSim simulator assures three of the features specific to almost all high-performance academic standard simulators: free availability for use, extensibility and portability. Full inheritance and polymorphism is used in the simulator's source code, allowing easier extension in the future, adding new functionalities.

We developed SMTAHSim simulator using the Microsoft .NET Framework 3.5 writing over 7K lines code. The simulator is running on Windows 2k/XP/Vista/7 and is currently used in undergraduate and graduate courses / laboratories in (Advanced) Computer Architecture at "Lucian Blaga" University of Sibiu. The simulator can be found at http://webspace.ulbsibiu.ro/adrian.florea/html/simulatoare/SMTAHSim.html

The organization of the rest of this paper is as follows. In section 2 we review the Related Work in software simulators domain dedicated to microarchitectures. Section 3 describes the theoretical background related to SMT, whereas section 4 presents the used benchmarks and simulation methodology. Section 5 illustrates the simulator software architecture, the simulator kernel from hardware viewpoint and the SMTAHSim user interface. Based on a short interactive animated example, we explain the SMT functionality. Finally, section 6 suggests directions for future work and concludes the paper.

2. RELATED WORK

After almost four decades of concerning in microprocessors design, implementation and exploitation, the researchers from computer science domain got the conclusion that simulators have become an integral part of the computer architecture research and design process (Yi and Lilja 2006) and simulation technology and methodology represents the crux of computer architecture research and development (De Bosschere et al. 2007).

Besides their importance proved in computer architecture research field, in the latest time, simulators have been extensively employed as a valuable pedagogical tool as they enable students to understand better the theoretical concepts and to visualize how microarchitectures components work and interact with each other (Yi and Lilja 2006).

In microprocessor systems' domain, as microarchitectural complexity increases, (crossing from instruction-level-parallelism to thread-level-parallelism and toward multi- and many-core architectures), it is more difficult to explain concepts like caches, out-of-order and speculative execution, power consumption, and the interactions among the architecture components without visual aids. Graphical simulations of these architectures allow students to easily grasp the architecture concepts by observing the flow of instructions in time, also by exploring the impact of different processors configuration on performance, dissipated energy and temperature. The static visual office tools (such as graphical charts, diagrams, slides etc.) are limited in efficiency: they cannot simultaneously exhibit both the structural relationships between microarchitectural components and the temporal dependences between executed instructions that are in-flight in the pipeline structures and cannot explain the functionality of coherence mechanism in

multicore architectures, etc. Some of the present-day most used didactical simulators are:

- **WinDLX** was developed for Windows operating system by Herbert Grünbacher (Grünbacher 1998) and simulates Hennessy and Patterson's DLX (DeLuXe) architecture (Hennessy and Patterson 2007). The DLX is a didactic microprocessor designed in accordance with the most popular RISC microprocessors (SPARC, MIPS, etc.). Simulation exposes in an expressive manner the principle of in-order pipelined execution (execution steps, data hazards, forwarding) and performance penalty involved by high latency instructions (delay slots) but, because it is modeled at architecture level quite few information is given about the processor.
- **VLIW-DLX** extends the WinDLX simulator to a VLIW model, using the same DLX ISA. It is implemented in Java and allows modifications of the architecture, including ISA (Bečvář and Kahánek 2007).
- **PCSpim-Cache** is an execution-driven simulator indented to be used in undergraduate courses for teaching cache memories within MIPS architecture. The tool allows to run step-by-step a selected code on a proposed cache organization and meanwhile observe dynamic changes in its structure (Petit et al. 2006).
- **PSATSim** is a powerful graphical simulator which offers support for students in better understanding the tradeoff between processors' performance and power consumption. The simulated microarchitecture is a configurable superscalar architecture with speculative out-of-order execution. The GUI allows in an interactive and easy way to simulate different microarchitectural configurations and assures a quick feedback (Smullen and Taha 2006).

However, unlike SMTAHSim, part of the existing simulators (Hostetler and Mirtich 1996; Burger and Austin 1997; Skadron et al. 2003; Sharkey et al. 2005; August et al. 2007) were designed primarily for research, the emphasis is on modeling the effects of architectural mechanisms. Most of these simulators are not trying to visually express the behavior of architectural mechanisms and the interaction between them. They are often designed to model a specific architecture and are also too complex to be studied by students who are beginners in concepts such as SMT. On the other hand most of the didactic simulators used in Computer Architecture are simulating only some simplistic toy-benchmarks. As it will be further presented, our developed simulator can process complex benchmarks that are intensively used in research activities, too. The interactivity of SMTAHSim simulator allows both to know in every machine cycle the content of CPU resources (reservation stations, functional units, reorder buffer, rename buffer, pipeline structure) and to experiment unforeseen circumstances like forcing a miss in D-Cache (this cache module is modeled statistically based on benchmark characteristics).

3. THEORETICAL BACKGROUND

It is well known that superscalar architectures exploit Instruction Level Parallelism (ILP) by fetching and executing more than one independent instruction per cycle. Despite that, the instruction-per-cycle (IPC) rate is limited to relatively low values, due to a lot of factors (Hennessy J., Patterson D., 2007).

The SMT architecture comes as a solution to the above mentioned limitation by combining the superscalar mechanism with the multithreading approach, which allows exploitation of both thread-level parallelism (TLP) and ILP. In order to achieve this performance, processor keeps different context information (program counter, stack pointer, etc.) for each active thread. Latencies which normally occur in single thread execution are, in this case, (partially) hidden by switching to another thread. This architecture represents the mapping of high level languages' explicit and implicit concurrencies (threads or/and micro-threads) into a processor having implemented multiple contexts. A thread from hardware level can be a task or a software thread within a task, but also can be made of software entities of smaller granularity as loops, routines or code blocks (micro-threads), which may be executed in parallel (Eggers et al. 1997; Vintan and Florea 2000).

SMT architectures inherit the superscalar processing mechanism and extend it with multithreading architecture specific components. Mechanisms as out-of-order speculative execution, register renaming and in-order completion are also met in SMT architectures. For assuring a different context, some hardware resources are private for each thread (branch predictors, renaming tables, logical register files, ROBs, Load/Store Queues, commit units) and others are shared among threads (fetch unit, decode unit, issue queue, physical register files, execution units and cache memory), using a tag information in instruction encoding to make the difference.

To ensure a high throughput, SMTs need a scheduling policy that arbitrates between threads for optimizing shared resources' utilization. The most common scheme is the very simple Round-Robin policy, which switches between threads in a circular way, regardless of their behavior. A better strategy is implemented in the ICOUNT policy which give higher priority to threads with the fewest instructions in decode, rename and instruction queues. The motivation is to give higher priority to fast-moving threads and, at the same time, to prevent starvation. ICOUNT tries to balance the number of instructions in the pipeline among the various threads so that all threads have an approximately equal number of instructions in the front-end pipeline and instruction queues (Manadhata and Sekar 2003; Eyerman and Eeckhout 2009).

SMTAHSim benefits of both mentioned fetch policies and gives user the possibility to understand how these are influencing the IPC rate and other parameters, driven by simulation monitoring tool.

4. SIMULATION METHODOLOGY

The SMTAHSim tool intends to help students in teaching superscalar and SMT architectures, by simulating a large palette of hardware configurations in step-by-step or full trace simulation mode. In order to obtain finest results, a hybrid simulation is performed. The results are collected at the end of each processing cycle by the Monitoring Tool and reported according to user preferences (see Figure 1).

SMTAHSim's execution-driven simulation is sustained by GUI which exposes in an interactive way the SMT's architectural structure and execution-time information. The step-by-step simulation gives a better perspective above the instruction stream through processing architecture and enables the user to visualize how basic superscalar and SMT mechanisms work.

For result validation, a set of benchmarks are used as simulator inputs, remaining to user choice which file is used as input for each hardware thread. The benchmarks represent a selection from the SPEC '95 (*applu, compress, fpppp, ijpeg, perl* (SPEC 1995)) and MediaBench 1.0 (*epic, mpeg2d, mpeg2e, pegwitd, toast* (Lee et al. 1997)) benchmark suites compiled for SimpleScalar Portable ISA (PISA). All these benchmarks cover a lot of applications ranging from compression to word processing, from compilers and architectures to games enhanced with artificial intelligence, etc. We choose to use different benchmarks in order to discover how these different testing programs influence the processing performances.

5. THE SMTAHSim FRAMEWORK

The developed simulator must support the learning process of students in SMT microarchitecture and search for possible changes (architectural or optimization techniques) to improve it. Providing a highly parameterized model for every microarchitectural instance, the performance obtained by simulation will represent a quick feedback mechanism related to the proposed changes, permitting thus an efficient design space exploration process. The simulator's execution consists in the following sequential steps:

1) *Initialization phase (configuring the micro-architecture with the input parameters including the benchmarks)*
2) *Simulation and monitoring phase*
3) *Results' reporting*

For the initialization phase the SMTAHSim provides help with a quick and easy to use Configuration Manager. This internal tool gives users the possibility to load preconfigured or saved configurations from the Configuration Repository or guides them through the configuration process. The last simulated configuration is loaded as default.

Some important architectural modules (called suggestively ISA, Branch Predictor, I-Cache, Fetch Policy) are implemented as interfaces and can be loaded by the Add-Ins Manager as precompiled libraries. The framework is easily extendable with our independent modules which are inheriting the provided interface. The Add-Ins can come also with their own configuration and simulation GUIs.

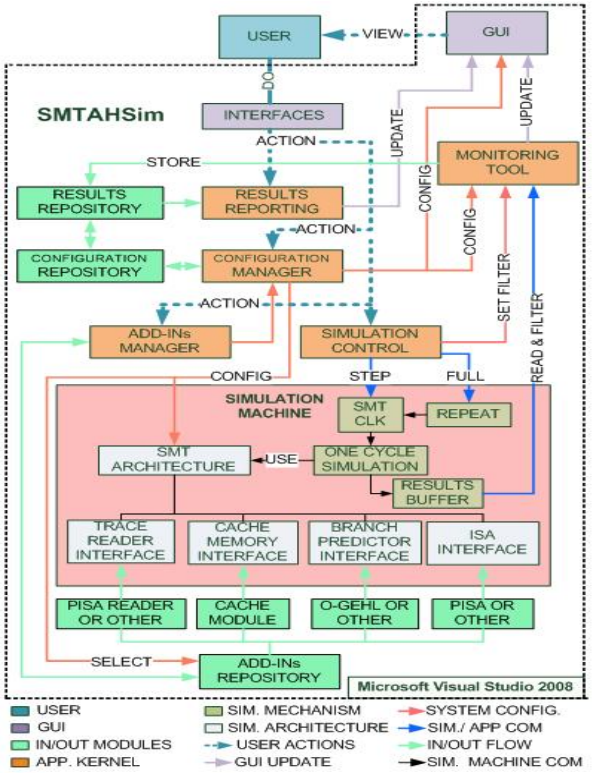

Figure 1: SMTAHSim Architecture

The SMTAHSim framework provides two simulation modes: a step-by-step simulation or a full unanimated simulation. The user can easily switch between these two modes by interacting with the Simulation Control module. Depending on the running simulation mode, the Monitoring Tool filters the results stored in the Simulator Kernel's Results Buffer. The simulation process is carried out by the Simulation Machine which performs independently of the user interfacing tools. The Results Buffer is updated at the end of every processing cycle with relevant information regarding performance and with a current context copy, which are later processed by the Monitoring Tool. This mechanism speeds up the simulation because the Simulation Machine is not interrupted by the graphical tools' operations, only by the buffer's overflow. The *producer-consumer* design pattern is implemented: as the Simulation Machine produces data, the Monitoring Tool is using it to update the Presentation layer (GUI). When the buffer is full, the simulation is suspended until the data are consumed. All final results are stored in the Results Repository and can be used to generate finest reports with the Results Reporting tool. User is able to get relevant graphics of SMT's performance indices in correlation with almost every architectural parameter.

5.1. The SMTAHSim Software Architecture

As we reveal in Figure 1, the framework is structured in four main software packages:

- **GUI** (Graphical User Interface) plays an important role as the highest level (Presentation Layer) of the framework, which manages all USER's interactions. This package is developed around two basic principles: ACTION and REACTION. All user actions have a quick feedback from the system, and all this reactions are managed carefully by GUI which makes the results representation in an interactive and easily understandable manner. Overall, this package makes the framework a friendly and easy to use application.
- **Input/Output** package is the low level management of all the simulation inputs and outputs giving the extensibility and accessibility dimensions to the framework. The aim of this approach is to make the user to easily access the final results and architecture configurations and, eventually, to develop his/her own configurations and extensions to the basic architecture. The framework came with some basic configurations which allow a proper evaluation of the SMT architecture's performances. For others configurations, a wizard is guiding the user step by step through the new configuration defining process. All new simulated configurations are stored in the Configuration Repository at the user's decision. The simulation results of these configurations are also stored at the user's decision, in the Results Repository, and linked to the simulated configuration. Due to this, software architecture results can be used to generate fine-grained reports regarding performance indices in correlation with almost every parameter, directly from the Results Repository. The Results Reporting tool supports users through this process and allows generating a large diversity of figures. The Add-Ins Repository plays a very important role because it stores all third party modules added by developers. The management of this collection is carried out by the Add-Ins Manager.
- **Application Kernel** is the middle level (middleware) which manages all user communications with the application. GUIs are assured for each middle level manager module in order to give user the access to low level packages. The simulation is initialized via the Configuration Manager and is run via the Simulation Control module (step by step or full trace simulation). The Monitoring Tool manages the feedback information and supplies the user with interactive animation by GUI update. Another important tool is the Add-Ins Manager which has the responsibility to manage all third party components added by developers. This module gives the SMTAHSim the "framework" dimension by allowing developers to extend the basic SMT architecture with other modules (ISA, branch predictor, data cache, etc.). The Add-Ins can provide their own configuration panel which will be loaded by the Configuration Manager at the configuration phase, and their parameters set will be then stored in the Configuration Repository together with the basic one. The developer must only implement the interfaces provided by the Add-Ins Manager, compile it in a library and then load it in the SMTAHSim Add-Ins Repository.
- **Simulation Machine** is the most important package, situated at low application level, which makes the effective simulation.

5.2 SMTAHSim framework: Simulation Machine

SMTAHSim models a configurable SMT architecture (Figure 2) designed in accordance with the M-SIM architecture (Sharkey et al. 2005) which has at base a superscalar architecture with speculative and out-of-order execution. The pipeline structure of SMTAHSim is based on that of PowerPC 5+ comercial processor (Sinharoy et al. 2005). Actually, M-SIM extends the SimpleScalar toolset (Burger and Austin 1997) with accurate models of the pipeline structures, including explicit register renaming, and support for the concurrent execution of multiple threads. Basic superscalar units are shared among micro-threads (Cache, Fetch Unit, Decode Unit, Dispatch Queue, Execution Units, Physical Registers), but in order to assure different contexts some resources are private for each micro-thread (Branch Predictors, Rename Tables, Reorder Buffers, Commit Units, Logic Registers).

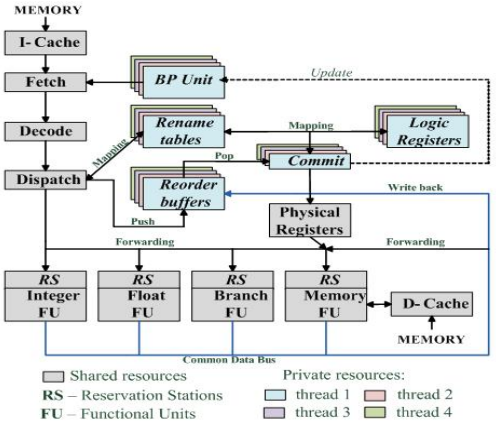

Figure 2: Simulated architecture

Simulation involves getting instructions from benchmarks and passes them step by step through the pipeline stages (Figure 3). There are three sections in the pipeline: in-order frontend (fetch the instructions from memory, make the branch prediction, decoding, rename registers and dispatching), out-of-order execution (the number of execution cycles is distinct for each instruction type) and in-order backend (gets finished instructions and updates the branch predictor). All essential architectural parameters (superscalar factor, number of micro-threads, number of execution units and their execution cycles, etc.) are configurable through the Configuration Manager.

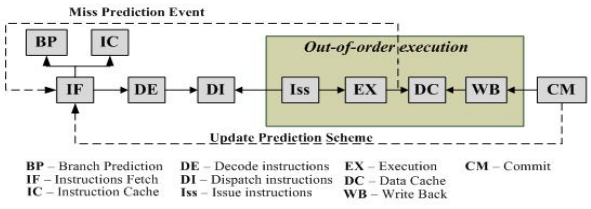

Figure 3: Simulated pipeline

Due to the benchmarks' characteristics, the effective execution can't be accurately simulated, because the registers' values are not known all the time. As a result of this limitation, the single feasible D-Cache implementation is based on an analytical model. Besides these, another degree of abstractization is that branch prediction is made in a single pipeline stage (Instruction Fetch) even if in reality it could take more cycles.

5.3. SMTAHSim Framework: GUI

Projects supported by the SMTAHSim simulator are dedicated to teach students about concepts related to superscalar and SMT architectures (processing mechanisms, constraints, limitation of ILP rate, etc.), and are fairly sustained by GUI. Being the closest to the user, this level of application has benefited the most of our attention in order to give easy and interactive access to all its features. Therefore, user can easily configure, simulate and track the step-by-step results. In order to get a big picture of SMT architecture performances, GUI also supports user with a reporting tool.

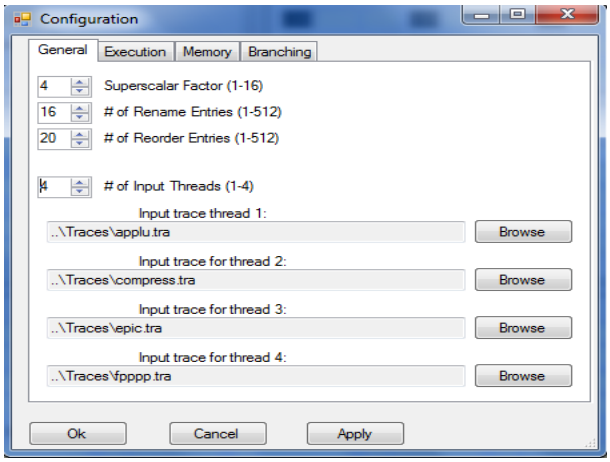

Figure 4: Configuration Manager Interface

The Configuration Manager Interface (Figure 4) makes possible to configure the simulated architecture from a classic superscalar one to a 4-threaded SMT one. Each micro-thread input can be settled independently. After the architecture's configuration the user can control simulation by Simulation Control Interface and make a step-by-step simulation: one simulated CPU cycle each step ("Next" button) or simulating the input traces entirely ("Go To End" button). In both cases the IPC rate is updated in every CPU cycle (Figure 5).

Figure 5: Part of Simulation Control Interface

When fine step simulation is chosen, the Monitoring Tool helps user to track the instruction flow from fetching to committing by animated visualization of each architecture units. Each instruction has a thread identification number and a unique per thread identifier, which are both distinctively colored, allowing to easily following the pipelined execution process (Figure 6). After the prediction of each branch instruction the subsequent instructions are marked as speculative and strike-lined until the branch execution ends and it turns out that the prediction is correct. In case of a mispredicted branch, after its execution, all speculative instructions from the afferent thread are squashed and the correct fetch path is taken.

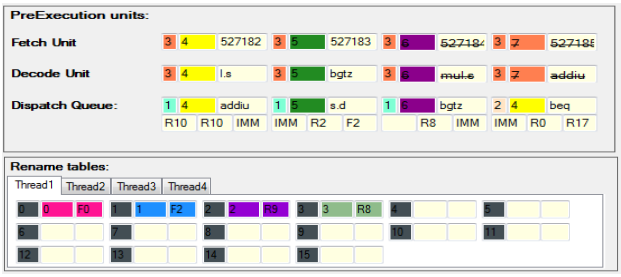

Figure 6: Monitoring Simulation

After each full trace simulation a summary of simulation results is shown.

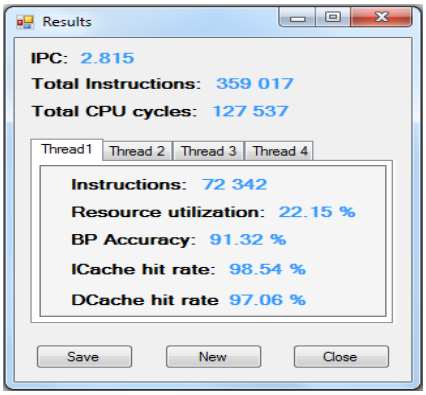

Figure 7: Results

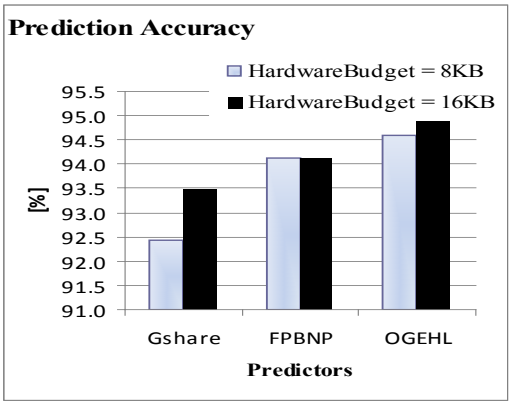

Figure 8: Average branch prediction accuracies

As a concrete example, Figure 8 illustrates comparatively the simulation results obtained with SMTAHSim using three prediction structures: *gshare* (Yeh and Patt 1992), *FPBNP* (Jiménez 2003) and *OGEHL* predictors (Seznec 2005). The statistics are collected after running the benchmarks described in section 4 on two configurations (one of them imposed by the hardware constraints of Championship Branch Prediction (CBP 2004)) and represent the average branch prediction accuracies.

6. CONCLUSIONS AND FURTHER WORK

The classical approach in teaching SMT concepts is based largely on oral communication of professors. They spend a lot of time in computer architecture research or use paper and pencil to follow the execution of the instructions flow. Although their efforts are to emphasize the processor kernel activities, many times they ignore the branch prediction and cache memory simulation. Our approach represents a formative necessity since computer architectures are mainly approached in a descriptive manner. Through our approach, students have the opportunity to be creative and innovative in computer architecture or in other research and didactical domains of computer science, even in countries not very developed from economical and technological points of view. Based on highly parameterized developed simulation tools, students can understand more in depth and in an integrated approach the theoretical concepts related to SMT, branch prediction constraints, limits of instruction level parallelism, TLP benefits, cache memories, etc.

Although SMT architectures outperform its predecessors, the evolution trend is maintained on vertical by growing the technologic complexity. Therefore a more aggressive approach (many micro-threads) is heavily limited by the management logic's complexity growth. It is clear that a new evolution trend is needed, on horizontal approach, by decentralization of processing power (multi-core). For further work we are mainly concerned to solve the following issues:

- Simulating on benchmark sets which allow a real implementation of data cache.
- Implementing a module for power consumption calculation; this can help to evaluate the SMT architectures based on this objective, too. It is well-known that SMTs are energy-intensive due to their complex and concentrated control logic. This module is also necessary for evaluation of hardware branch predictor within a given chip area budget, from both power consumption and performance points of view.
- Adding modules to improve the processing rate, such as *value prediction*, *dynamic instruction reuse* and an *execution trace cache.*

REFERENCES

August D., Chang J., Girbal S., Gracia Perez D., Mouchard G., Penry D., Temam O., Vachharajani N., 2007 *"UNISIM: An Open Simulation Environment and Library for Complex Architecture Design and Collaborative Development"*, IEEE Computer Architecture Letters, 20.

Bečvář M., Kahánek S., 2007, *"VLIW-DLX Simulator for Educational Purposes"*, Proceedings of the 2007 Workshop on Computer Architecture Education.

Burger D., Austin T., June 1997, *"The SimpleScalar Tool Set, Version 2.0"*, University of Wisconsin Madison, USA, CSD TR #1342.

CBP: The 1st Journal of Instruction Level Parallelism Championship Branch Prediction Competition (CBP-1), Oregon, USA, 2004.

De Bosschere K. et al., 2007, "*High-Performance Embedded Architecture and Compilation Roadmap"*, Transactions on HiPEAC I, Lecture Notes in Computer Science 4050, Springer-Verlag, pp 5-29.

Eggers S. Emer J., Levy H., Lo J., Stamm R., Tullsen D., 1997, *"Simultaneous Multithreading: A Platform for Next-Generation Processors"*, IEEE Micro, Vol 17, Issue 5, 12-19.

Eyerman S., Eeckhout L., March 2009, "*Memory-Level Parallelism Aware Fetch Policies for Simultaneous Multithreading Processors"*, ACM Transactions on Architecture and Code Optimization, Vol. 6, No. 1.

Grünbacher H., 1998, "*Teaching Computer Architecture / Organisation using simulators"*, Proceedings of the 28th Frontiers in Education, IEEE Computer Society, Vol. 03.

Hennessy J., Patterson D., 2007, "*Computer Architecture: A Quantitative Approach"*, Morgan Kaufmann, 4th Edition.

Hostetler L.B., Mirtich B., 1996*, "DLXsim - A Simulator for DLX"*.

©Intel Corporation, 2010, *http://www.intel.com/*

Jiménez D., 2003, "*Fast Path-Based Neural Branch Prediction*", Proceedings of the 36th International Symposium on Microarchitecture.

Lee C., Potkonjak M. and Mangione-Smith W., 1997, *"MediaBench: A Tool for Evaluating and Synthesizing Multimedia and Communications Systems"*.

Manadhata P., Sekar V., 2003, *"Evaluating Throughput and Fairness of Thread Fetch Policies for SMT Processors"*, www.cs.cmu.edu/~vyass/Fall03/15740/class-project/project_report.pdf

Petit S., Tomás N., Sahuquillo J., Pont A., 2006, *"An Execution-Driven Simulation Tool for Teaching Cache Memories in Introductory Computer Organization Courses"***,** Proceedings of the 2006 Workshop on Computer Architecture Education.

Seznec A., 2005, *"Analysis of the OGEHL predictor"*, Proceedings of the 32nd International Symposium on Computer Architecture (IEEE-ACM), Madison.

Sharkey J., Ponomarev D., Ghose K, 2005, *"M-SIM: A Flexible, Multithreaded Architectural Simulation Environment"*. Technical Report CSTR-05-DP01, Department of Computer Science, State University of New York at Binghamton.

Sinharoy B., Kalla R.N., Tendler J.M., Eickemeyer R.J. and Joyner J. B. 2005,"POWER5 System Microarchitecture", *IBM Journal of Research and Development*, Vol. 49, Num. 4/5, pp. 505-521.

Skadron K., Stan M.R., Huang W., Velusamy S., Sankaranarayanan K., Tarjan D., 2003, *"Temperature-Aware Microarchitecture."*. Proceedings of the 30th International Symposium on Computer Architecture.

Smullen W., Taha T., 2006, *"PSATSim: An Interactive Graphical Superscalar Architecture Simulator for Power and Performance Analysis"*, Proceedings of the 2006 Workshop on Computer Architecture Education.

SPEC 1995, *The SPEC benchmark programs,* http://www.spec.org/cpu95/

Vintan L., Florea A., 2000, *"Microarhitecturi de procesare a informaţiei"* (in Romanian), Editura Tehnică, Bucureşti.

Yeh T., Patt Y., 1992, "*Alternative Implementations of Two-Level Adaptive Branch Prediction*". Proceedings of the 19th International Symposium on Computer Architecture.

Yi J.J., Lilja D.J., 2006, *"Simulation of Computer Architectures: Simulators, Benchmarks, Methodologies, and Recommendations"*, IEEE Transactions on Computers, vol. 55, No. 3.