Bringing Real Processors to Labs

C. Gómez, M.E. Gómez, and J. Sahuquillo

Universitat Politècnica de València, Valencia, Spain

Abstract—The architecture of current processors has experienced great changes in the last years, leading to sophisticated multithreaded multicore processors. The inherent complexity of such processors makes difficult to update processor teaching to include current commercial products, especially at lab sessions where simplistic simulators are usually used. However, instructors are forced to reduce this gap if they want to properly prepare students in this topic. Dealing with these complex concepts at Labs does not only help reinforce theoretical concepts but also has a positive effect in the students' motivation. This paper presents a methodology designed for the study of current microprocessor mechanisms in a gradual way without overwhelming students. The methodology is based on the use of a detailed simulation framework, used both in the academia and in the industry, which accurately models features from current processors. Due to the huge simulator complexity, it is introduced through several learning phases. Qualitative and quantitative results demonstrate that students are able to develop skills in a detailed simulator in a reasonable time period and, at the same time they learn the details of complex architectural mechanisms of commercial microprocessors.

Index Terms—Lab sessions, Computer architecture, Processor simulation, Multicore processors.

I. INTRODUCTION

Nowadays, electronic devices are present in all the areas of our everyday life. This has been possible thanks to the great computational power reached by current microprocessors and to the low power they consume. This huge computational power has come from an increase in hardware and functional complexities. Computer architecture has evolved from monolithic processors to multicore processors with complex cache hierarchies and efficient on-chip interconnects. As an example, Figure 1 presents a recent commercial multicore processor, which incorporates a three-level cache hierarchy and supports the execution of multiple threads on several cores. Nevertheless, in many universities students are not taught in these new features, or at least they do not work them in laboratories. This means that these complex concepts or mechanisms are not fixed in lab sessions, mainly due to the high complexity of

current microprocessor generations, which is not modeled in the simple simulators usually used in laboratories, as well as the constant evolution of commercial processors. This widely extended methodology is a mistake because these concepts are precisely the most difficult to understand and should be worked out in laboratories for better understanding. In addition, students feel that the studied concepts widely differ from the real hardware, which discourage them to the study of these topics. Thus, it is important for instructors in this area to spark students' motivation. From our view, this emphasis should be persistent and linked with commercial products showing how real and recent processors work.
This paper proposes a methodology aimed at bringing the industry to the academia in general, and in particular, to laboratories, by training students to work with real mechanisms at laboratory sessions. For this purpose, the proposed approach relies on the use of a detailed simulation framework, able to accurately model current microprocessors. In addition to this issue, it is highly recommendable that the simulation tool fulfills the following features to increase the student motivation: i) it should model advanced processor features, and ii) it should be used both in the academia and the industry. The second feature is important not only to encourage students in their work but also to help them when looking for a job in the future. A major barrier to the widespread use of these simulators for undergraduates is their intrinsic difficulty since they are really complex, as the systems they model. An interesting tentative has been carried out in courses offered in top USA universities like Stanford [1] where Zsim [2] is used. However, even in these courses, the simulator is used as a black box that provides performance for a set of machine parameters.
In contrast, the approach proposed in this paper goes a step ahead, looking at the details within the box. The methodology has been applied sing the Multi2Sim [7] simulator, a cycle-accurate simulator of multicore and multithreaded processors widely used by PhD students and researchers (both from the industry and academia). To tackle the simulator complexity, we propose the use of the same simulator in several courses jointly with a carefully designed progressive methodology. The methodology introduces

the simulator in a progressive manner in such a way that students at the end of the offered courses are able to modify small pieces of the simulator code, which allow them to experiment with new processor features. At the early stages, students begin with a very guided use of the simulator as a black box (i.e. without modifying the source code). After that, students are progressively introduced in the simulator code, ranging from minor modifications of the source code to the completion of their Degree Thesis[1] and their participation as co-authors in research papers. The gradual project-based learning approach has been already followed by instructors ([3],[4]) in other areas. However, unlike these works, this paper proposes the combination of using both a detailed simulator that models realistic processors and the gradual project-based methodology with the aim of training students in advanced processor architecture concepts. The great advantage of using both together is that students are allowed to experience how real processors work without feeling overwhelmed by the simulator complexity.

Quantitative and qualitative results have been analyzed to assess the proposed methodology. Results show that since the proposed methodology was introduced, students have obtained better marks in the related courses and moreover, they are more encouraged in the study of computer architecture. As a consequence, the number of students that enroll the courses (they are elective courses) has appreciably grown.

The paper is organized as follows. Section II introduces computer architecture courses at the Universitat Politècnica de València. Section III describes the methodology. Section IV presents the simulation framework. Section V shows a case study. Section VI analyzes the proposal effectiveness. Finally, Section VII presents some concluding remarks.

[1] A Degree Thesis is required at the UPV to earn the Computer Engineering degree.

II. COMPUTER ENGINEERING TEACHING

A. Computer Architecture Courses at UPV

After some compulsory computer architecture courses, two main elective courses are offered at Universitat Politècnica de València (UPV) that present a close approach to commercial products: “Advanced processor architectures” offered both in the last year of the Computer Engineering Degree and in the Master in Computer Engineering; and “Networks on-chip” offered in the Computer Engineering Master. These courses are complementary and students are suggested to follow both of them.

During the last two academic years we have developed and implemented a new methodological approach that includes the mentioned courses as if they were only one. The methodology is based on the fact that students can be trained in laboratories with complex features of commercial processors that have been taught in theoretical lectures, without being overwhelmed. For doing this, both courses use the same simulation framework that allows students to model realistic commercial processors mechanisms. The benefits of this approach are twofold. On the one hand, students analyze and realize how the different system components interact with each other and affect the overall system performance. On the other hand, students only need to learn a single tool in an incremental way, which helps the learning process. Both benefits provide students worthy and invaluable skills in future research or professional activities. Another key element in this approach is the coordination between instructors of both courses, which again influences on the student motivation since students really appreciate the interactions among topics of both courses.

B. Traditional Computer Architecture Teaching

Commercial processors are continually evolving. This constant evolution has forced the continuous update of teaching contents, both theoretical ones in lectures and practical ones in laboratories.

Concerning laboratories, several simulators are often used depending on their intended use. Simulation tools can be simple and easy to use or complex and harder to understand. As an example of simple simulators, we can find DLXV that is typically used to study the processor pipeline. DLXV comes in handy for understanding simple scalar processors, which are the basics of superscalar processors. Nevertheless, for more advanced concepts such as superscalar or multithreaded processors, DLXV does not provide any teaching support.

Most universities do not cover this gap for undergraduates and complex simulators like SimpleScalar [5], Sniper [6], and Multi2Sim [7], are exclusively used by PhD students. This is a clear mistake due to two main reasons. First, students are not trained with those concepts at labs in order to get a sound understanding of them. Second, but not less important, the chance of having highly motivated students in computer architecture is missed.

In contrast to the traditional approach, this paper proposes the use of a detailed simulator jointly with a methodology to overcome the simulator complexity through several learning stages.

III. PROPOSED METHODOLOGY

The overall objective of the proposed methodology is twofold, to provide students a solid knowledge about realistic processor architectures and to encourage them in the study of computer architecture topics. The methodology consists of four different phases with an increasing difficulty degree. The phases are: i) modification of simulation parameters, ii) modification of simple parts of the simulator code, iii) implementation of complete functionalities, and iv) full autonomy.

At the first stage, students are asked to only modify simulation parameters to analyze how they impact on performance. Students must first modify the values of some key system parameters previously studied at lectures (e.g., the branch predictor or issue width), then they must launch simulations to obtain the performance results, and finally, they must analyze the performance. This first step is done in a guided way at laboratories. Students are provided with a detailed instruction booklet that explains all the steps that must be followed and instruct them in the analysis of the obtained results.

The second stage goes one step further and students are asked to modify simple parts of the code. They modify specific and limited parts of the code like prefetching mechanisms or the scheduling policy at the memory controller. As in the previous stage, this work is done with a careful instructors' guidance; for instance, instructors provide the code of a very simple prefetching mechanism to the students and they are asked to implement other prefetching mechanisms from it. For this purpose, instructors provide the students with another booklet describing the source code of the simulator. From our experience, this phase is best performed as final course project. We consider this phase, which is not usually considered, important since the study of the source code provides a complete understanding of the details of a given processor feature to the students. In general, those students who successfully complete this stage are willing to face greater challenges by implementing full functionalities in different processor components, and playing around with the simulator.

At the third phase, we try to provide the students with autonomy and/or self-confidence to implement a full basic or novel mechanism from scratch. This step can best fit for a Degree Thesis or Master Thesis. The results obtained by our students during this phase have been astonishing and they have been even published in top-notch computer architecture conferences like IPDPS, PACT, and Euro-Par. This stage stimulates the interest of students in research on computer architecture topics.

Those students who successfully complete the three previous stages are in a privileged position to start their PhD studies, as they are at least one year or even several ones ahead of those who have not followed the proposed methodology.

IV. SIMULATION FRAMEWORK FEATURES

A major aim of the proposal is to provide the methodology with a single relatively long-life simulator able to be used both in computer architecture related courses and, if students are interested, in the development of their future PhD Theses. To cover the mentioned courses and the wide spectrum of computer architecture topics where the research is currently focusing on, we have used Multi2Sim, which is capable to model multithreaded and multicore superscalar processors as well as graphics processing units (GPUs). In addition, recent extensions include a detailed modeling of the on-chip network and the memory controller. The simulation framework is being used by major microprocessor companies like AMD and Intel, and can be freely downloaded from [8]. Multi2Sim provides both functional and detailed simulation of the major system structures: core, cache hierarchy, interconnection network, and memory controller.

- The core is pipelined in six stages (fetch, decode, dispatch, issue, writeback, and commit), and it supports speculative execution. The number of cores and number of threads sharing a core are configurable. The framework implements the three main multithreading paradigms: coarse grain (CGMT), fine grain (FGMT), and simultaneous (SMT).
- Cache hierarchy configuration is highly flexible. The user can define as many cache memory levels and number of caches in each level as desired, and the caches can have any size and geometry. Cache coherence is guaranteed by means of a MOESI protocol.
- The interconnection network among the different levels of the memory hierarchy is also freely configurable with a simple model that allows the definition of end nodes (memories or cores), switches, links, topology, etc.
- The memory controller mimics the behavior of real DDR memory controllers.
- NVIDIA Fermi-like GPU architectures are also modeled.

The simulator provides extensive performance, power, and area reports for the different processor components.

Although authors have applied the methodology using Multi2sim, other platforms that provide detailed simulation of the processor like Sniper [6] can be used. The learning phase of these sophisticated tools, however, is really long and it can take a long time to understand only a few modules of the simulator. Because of this reason it is important for instructors to have a deep knowledge of the selected tool. Since Multi2sim [7] was originally developed at UPV, we are strongly familiarized with this framework and its internal details. In fact, the authors of this work have advised several PhD theses, which have widely extended multiple parts of the simulation framework.

V. CASE STUDY

This section focuses on the implementation of the first two stages of the methodology. A key point for the success of any methodology is that the work to be done by students motivates them. With this aim, we chose an 8-tile multicore processor that implements a mesh NoC as baseline system. This architecture was chosen since it resembles to those that could be found in commercial multicore processors. With this processor, students have to complete several learning stages in a progressive way. At each learning stage, students only have to deal with the simulator part that is required for that stage, hiding the rest of the simulator. Unlike what is typically done with PhD students, it is not advisable to make a thorough tutorial about the simulator before starting the work, as this would likely strongly discourage most of the students. To overcome such a problem we opted for a "learn as you work" methodology. Keeping this idea in mind, the following four main stages were designed: baseline system modeling, memory hierarchy modeling, execution of parallel benchmarks, and prefetching mechanisms implementation.

A. Baseline system modeling

As an initial step to make the proposed approach feasible, instructors prepared a detailed simulator guide for those aspects that are required for this stage. This is not a typical simulator guide but it combines and makes relations between theoretical concepts studied in lectures and practical aspects. This guide describes the three main subsystems that can be modeled in the simulator: cores, cache memories, and interconnection network. For each subsystem, the guide explains its operation and role in the system, and provides a brief description of the main configuration parameters. For instance, the guide shows how to change the branch predictor in the core subsystem; in the case of cache memories, the guide explains how to change the size of the cache line or the access latency; and for the interconnection network subsystem, the guide presents how the bandwidth of the links and the size of the buffers at the switches can be changed. Moreover, regarding caches, the guide shows how to model the entire cache hierarchy, interconnecting the caches of the distinct levels among them, to the cores and to the main memory. In a similar way, the guide explains the correct way of defining an interconnection on-chip network to connect all the different cache memories or network nodes.

After explaining the modeling of the system components, instructors define the baseline system to be modeled mentioned above, which is a typical configuration of commercial processors as shown in Figure 2. We start with a single cache level to help students to observe the effects of caches in the global system performance. In this case, each tile in Figure 2 is composed of one processing core and one 32KB L1 cache memory. After that, an additional cache level is introduced in two flavors: private and shared cache memories. We provide the students with the baseline model and j the required configuration files (core, memory hierarchy, and interconnection network). As an example, two small fragments of two configuration files are shown. Example 1 shows the configuration file for the on-chip network called mesh. This file configures the global network parameters like link bandwidth and buffer size, the topology, and the routing algorithm. The example illustrates the connection of node 0 and switch 1, and part of the routing table associated to switch zero. On the other hand, Example 2 presents part of the configuration file for the memory hierarchy. It shows the L1 cache geometry (number of sets, number of ways, block size, and cache latency), the definition of the caches of node 0 and their location in the memory hierarchy, and the connection of L1 caches to the main memory through the mesh defined in Example 1.

B. Memory hierarchy modeling

Once students become familiar with the configuration files, they are asked to upgrade the system by adding 512KB L2 private caches. In this way, each L1 cache connects to a L2 cache, and all these L2 caches connect to the main memory via the interconnection network. Thus, each tile in Figure 2 consists of a core with its L1 and L2 caches. It has to be remarked that in order to perform this study the student only has to modify the memory hierarchy configuration file since both the interconnection network and the processing cores remain the same. With this extension we aim to provide the students with more self-confidence with the simulator.

The next step in the proposed simulator learning process is to replace L2 private caches by shared L2 caches. In this model, each L2 cache memory is shared by a pair of cores. That is, core #0 and core #1 share the L2 cache, core #2 and #3 also do that, and so on. As the number of L2 caches is halved, their size is doubled to keep the L2 storage capacity constant, and their latency is accordingly increased from 6 to 8 cycles. To deal with this extension, students must modify both the memory hierarchy and the interconnection configuration files since the number of L2 caches has changed.

C. Execution of multiprogrammed and parallel workloads

Students experience with standard benchmarks, used to evaluate actual processors and extensively accepted by the scientific community. In particular, we choose the SPLASH2 parallel benchmark suite [9]. To relax the amount of work, we provide the students with the scripts to launch the benchmarks of the suite and they are asked to evaluate the system performance by measuring three main performance metrics: execution time, network latency, and number of on-chip network packets. In this way, students can relate the execution time with the network traffic and the required time to send packets.

Students can observe that the system performance depends on the memory hierarchy and the parallel application. To analyze the obtained performance metrics, students are suggested to arrange their results in a graphical manner similar as the depicted in Figure 3. The three aforementioned performance metrics are normalized to the values obtained with private L2 caches for three (Cholesky, FFT and Radix) benchmarks. This kind of plot permits students to clearly see that adding an L2 cache speeds up the execution time since it significantly reduces the number of accesses to main memory. For instance, the execution of Radix without L2 takes 23x longer.

The study also allows students to realize about the benefits of having a shared cache memory, such as the modeled in the last configuration, for parallel workloads. This effect comes from the fact that parallel applications share code and data among the processor cores, so by implementing shared caches, coherency traffic can be significantly reduced. Moreover, if two cores sharing the L2 cache access a shared block, this cache will hold a single copy of the block for both cores, whereas two copies of the shared block are necessary in L2 private caches. Thus, shared caches make a more efficient use of the available storage capacity. Looking at the results, students are able to observe that the execution time of some applications is reduced with shared caches in spite of the average cache access time increases, thanks to the fact that the number of packets in the network is strongly reduced. This is the case of FFT in Figure 3.

D. Final project

Finally, instructors have prepared a list of Final Projects for these courses aimed at encouraging students that are further interested in computer architecture topics. The work done in these projects will improve the already accomplished simulator skills. As mentioned above, in this final phase students are asked to implement a complete basic processor mechanism. The student can select either any of the available projects or alternatively suggest her own proposal and submit it to the instructor who will check the properness. Below we present some examples of final projects:

- Hardware prefetchers. This project consists in implementing L2 hardware prefetchers. Students are asked to implement different prefetchers: a *n*-block sequential prefetcher and a stride prefetcher that detects constant strides from the cache accesses.
- Memory controller scheduling policies. In this project students must implement distinct scheduling policies in the memory controller. For instance the first-ready fist-come first-served (FR-FCFS) policy.
- Thread to core allocation. When running multiprogrammed workloads, the system performance depends on the core in which each benchmark is allocated. This project evaluates the impact of several thread-to-core allocation policies in SMT processors with shared L2 caches.
- NoC virtual channels. In this project the implementation of virtual channels in the NoC is evaluated analyzing how they affect the system performance and the area of the switches.
- NoC topology. This project consists in the implementation and evaluation of different NoC topologies (e.g. C-Mesh and WK-recursive) and the associated routing algorithms.

Due to the complexity of the project, three progressive steps are followed to ease the learning process: preliminary work, study of the baseline mechanism, and implementation of a more complex mechanism. For illustrative purposes, below these three steps are detailed for the *hardware prefetchers* final project.

D.1. Hardware prefetcher final project

*Preliminary Work***.** To start with, students study the impact on performance of varying the L2 cache block size with two main limitations. The L2 block size has to be multiple of the L1 block size while the total L2 cache size must remain the same. In this case, when a miss rises in both cache levels, the L2 fetches the missing block from main memory. This action brings *n* L1 blocks, being *n* the relation between the L2 and the L1 block size. This study shows a similar effect on performance as prefetching. Students run simulations to obtain the performance for the studied benchmarks varying the L2 block size from 64B up to 2KB. Then, they study the performance trend as the block size grows. The overall system performance is usually quantified in terms of the IPC (Instructions Per Cycle), and results must be delivered in a plot like the shown in Figure 4. All the values are normalized with respect to the performance achieved with a 64B block size. As can be seen, the system performance improves with the block size up to 512B but increasing the block size beyond this value negatively impacts on performance.

Study of the baseline Prefetcher. After this first initial study, designed to provide a global overview of how prefetching works, students are asked to implement several prefetching mechanisms in the simulator. To help students with this learning phase, implementation stubs of the simplest prefetching technique (*One Block Look-Ahead* or OBL) jointly with a user's guide is provided to them. The guide illustrates how the core code must be modified step by step to implement other prefetching mechanisms. Basically, the provided stubs implement two main components: a queue for pending prefetches and the pattern detection/triggering mechanism, which on a cache miss, triggers new prefetches that are inserted in the queue. The queue is looked up at the *issue* stage and, if not empty, a new prefetch request is issued.

Implementation of more complex prefetchers. Once students become familiar with the code of the simple prefetcher, they are in an advantageous position to implement and analyze more complex prefetchers. The following are some examples:

- *n*-block sequential prefetching: This work consists in extending the provided prefetcher in order to prefetch not only one block in advance from memory but to bring *n* consecutive blocks from the main memory. Students must study the performance trends across all the benchmarks, varying the value of *n* (1, 2, 4, and 8).
- Constant based stride prefetching: In this work, students evaluate different implementations of prefetchers that detect constant strides. For each prefetcher, different configurations that put

the prefetched data in the cache or in stream buffers are evaluated varying the prefetcher aggressiveness.

VI. ASSESSMENT METHODOLOGY

Instructors have analyzed both quantitative and qualitative results to evaluate the proposal. Quantitative results refer to the grades achieved by students, while qualitative analysis includes i) students' enrollment to the course, ii) understanding of the course topics, and iii) a survey that was answered by students. Below the analysis of these items is discussed.

Grading. One important aspect that gives information about the effectiveness of the proposal is the marks obtained by students. Table I summarizes them for the year before the methodology was introduced and the year where it was applied. After applying the methodology their marks have noticeably improved; the percentage of students with a grade of "B" doubles with respect to the previous year and the number of students with a grade of "D" has significantly dropped.

The final marks consider both written exams and the assessment of the work performed in the laboratory sessions. To develop the lab work instructors provide the students with working guidelines. To grade the lab sessions students deliver to the professor a report containing the answers to a questionnaire that instructors previously prepared aimed at checking the correct interpretation and analysis of the obtained results.

The evaluation methodology weighted 70% and 30% exams and lab sessions, respectively, to obtain the final grade. Table II presents the final grade achieved by students broken down in exams and lab sessions. Two important observations can be appreciated. On the one hand, labs marks are slightly lower when applying the methodology due to the increase in complexity of the concepts worked in the lab sessions. However, this fact does not negatively affect the final grade because students have significantly improved their marks in written exams with similar difficulty.

Better understanding of the course topics. The increase in the final marks can be explained by the followed methodology. The use of a realistic processor simulator framework offers two important advantages. Students can read the detailed simulator code of specific components and study performance interactions among the different system components. Authors would like to remark that the same instructor works with students both in lectures and laboratory sessions. This way allows the professor to better follow the student learning process, and thanks to this, instructors can claim that improvements in the exams' marks come in part from the sound knowledge students have acquired in the lab sessions with the detailed simulator.

Enrollment. Since the course is elective, the enrollment grows or drops depending on how attractive the course is made to students. Most of them have really appreciated the proposed approach, since the enrollment has grown by 2.6*x* (from 18 students in 2013 to 55 students for the next course in 2014 after the methodology has been applied), making the AAV (Advanced Architecture) Course one of the most popular among students.

Survey. Finally, to provide qualitative results about this great success, a survey was made to analyze students' perception of both the course and methodology. The survey included twenty-five questions classified in four main categories to evaluate possible motivation factors: i) use of a single tool, ii) understanding complex microprocessor mechanisms, iii) hiding *non-necessary* structures, and iv) approaching labs to real processors. The questionnaire was completed by 22 students corresponding to those belonging to one of the two lab groups. Table III shows an excerpt of the questions and the provided marks. For illustrative purposes one question from each category is shown. Most of the students selected either the highest mark or the second one regardless the category to which the question belongs to, which means that all these categories really impact on motivation. An interesting observation is that the fourth category is the most appreciated by students, which is the main point of our approach.

VII. CONCLUSIONS

This paper has presented a new methodology that has been carried out during the last two academic years at the UPV with the aim of providing the students with a more realistic approach of how real processors work. The methodology relies on the use of a detailed simulator used both in the academia and the industry, and design a set of phases with gradual difficulty which enables students to acquire a solid understanding of real microprocessors mechanisms without overwhelming them. Due to the high complexity of the simulation framework, authors suggest to use it in several courses in order to gradually improve students' simulator skills, allowing them to reach a realistic understanding about how current processors work. Unlike other existing proposals, undergraduate students not only change simulator parameters but the provided skills also allow them to extend the simulator code by implementing new microprocessor mechanisms.

The obtained results have shown that students, in a reasonable time, can work quite fluently with some parts of the simulator, acquiring a deep knowledge of real hardware that has positively impacted on students' marks. In addition, students' attitude has significantly changed. The number of students interested in following the offered computer architecture courses and Degree Thesis has increased in a meaningful manner. Moreover, the results of the work developed in some of these projects have been published in top computer architecture conferences.

Finally, from the instructors' point of view, once this methodology has been developed and the simulator framework has been deployed, we have built up a platform for next editions of the offered courses that will allow us to obtain a higher number of Degree and Master Theses to help students in the study of processor architectures. The work to be done in these theses will be carefully selected to motivate both students and instructors.

ACKNOWLEDGMENTS

This work has been financed by the Spanish Government under grant TIN2012-38341-C04-04.

REFERENCES

[1] http://class2go.stanford.edu/EE282/Spring2013#

[2] Daniel Sanchez and Christos Kozyrakis. ZSim: fast and accurate microarchitectural simulation of thousand-core systems, ISCA, 2013, pages 475-486

[3] S. M. Aziz, E. Sicard and S. Ben Dhia. *Effective teaching of the physical design of integrated circuits using educational tools*, IEEE Transactions on Education, Vol. 53, No. 4, New York: IEEE, pp. 517-531, Nov. 2010.

[4] S. L. Dexter, R. E. Anderson, and H. J. Becker. *Teachers' views of computers as catalysts for changes in their teaching practice*. Journal of research on computing in education, 31, 221-239, 1999.

[5] Todd Austin, Eric Larson and Dan Ernst. *SimpleScalar: An Infrastructure for Computer System Modeling*. IEEE Computer vol. 35(2), 2002.

[6] Trevor E. Carlson, Wim Heirman and Lieven Eeckhout. *Sniper: exploring the level of abstraction for scalable and accurate parallel multi-core simulation*. Conference on High Performance Computing Networking, Storage and Analysis, page 52, 2011.

[7] Rafael Ubal, Julio Sahuquillo, Salvador Petit and Pedro Lopez. *Multi2Sim: A Simulation Framework to Evaluate Multicore-Multithreaded Processors*. 19th International Symposium on Computer Architecture and High Performance Computing, pages 62–68, 2007.

[8] The Multi2Sim Simulation Framework Website, http://www.multi2sim.org

[9] S. Woo, M. Ohara, E. Torrie, J. Singh and A. Gupta. *The Splash-2 programs: Characterization and methodological considerations*. In 22th International Symposium on Computer Architecture (ISCA), pages 24–36, 1995.