J. Parallel Distrib. Comput. 118 (2018) 128–139

Contents lists available at ScienceDirect

J. Parallel Distrib. Comput.

journal homepage: www.elsevier.com/locate/jpdc

Designing lab sessions focusing on real processors for computer architecture courses: A practical perspective

Josué Feliu *, Julio Sahuquillo, Salvador Petit

Department of Computer Architecture (DISCA), Universitat Politècnica de València, Camí de Vera s/n, 46022, València, Spain

HIGHLIGHTS

- We present a new approach for computer architecture labs based on real processors.
- We discuss the methodology and scheduling framework to support the labs.
- Five lab examples studying different topics illustrate the scope of the approach.

ARTICLE INFO

Article history:
Received 15 June 2017
Received in revised form 2 February 2018
Accepted 26 February 2018
Available online 8 March 2018

Keywords:
Lab sessions
Computer architecture
Real processors
Processor complexity
Scheduling framework

ABSTRACT

Computer architecture courses typically include lab sessions to reinforce, from a practical perspective, concepts and architectural mechanisms studied in lectures. Lab sessions are mainly based on simulation frameworks because they benefit learning. Reading the source code that models certain processor mechanisms allows students to acquire a sound knowledge of how hardware works. Unfortunately, simulators that model current multicore processors are getting more and more complex, which lengthens the learning phase and complicates their use in time-bounded lab sessions.

In this paper, we propose a new approach that complements the use of simulation frameworks in lab sessions of computer architecture courses. This approach is based on performing experiments on current commercial processors, where multiple hardware events related to the performance of the computer components under study are monitored. Then, students analyze the measured events and how they impact the overall performance. Such analysis motivates students and, not only helps reinforcing the theoretical concepts, but also increases their analysis skills. In this paper we present the methodology and scheduling framework that support the proposed approach and discuss five lab sessions, which can be applied in different courses, covering multiple computer architecture topics.

1. Introduction

Most electrical and engineering schools around the world offer two or three courses on computer organization and computer architecture topics. These courses usually comprise both conventional lectures at classroom and practical sessions at laboratories. Computer architecture courses are typically considered as difficult courses by students mainly due to the wide range of topics that are covered as well as the intrinsic difficulty of some of them. In addition, topics are usually studied from a theoretical perspective, which discourages many students from continuing their education in computer architecture.

Lab sessions are an excellent way to reinforce the theoretical concepts taught at conventional lectures. They provide a clear understanding about how the computer mechanisms studied at lectures work, which helps correcting any possible misunderstanding and motivates these students that might feel discouraged at classrooms. To this end, labs use computer simulation frameworks like Multi2sim [21] or Snipper [6], which model complex processors and their structures in detail. Working on small fragments of the simulator source code helps students to appreciate the details about how specific processor mechanisms work, allowing them to acquire a sound knowledge about internal architectural mechanisms from a practical perspective. Therefore, simulators play an important role in post-graduate courses, especially when a major goal of the course is to provide research skills to students.

While simulators are valuable tools to study the details of hardware, they fail to provide an overview of how the distinct components of the machine interact to each other. For example, how last level cache (LLC) misses impact processor performance. One reason that explains this drawback is that simulating a current multicore with complex cores and a huge cache hierarchy is time

* Corresponding author.
E-mail address: jofepre@gap.upv.com (J. Feliu).

https://doi.org/10.1016/j.jpdc.2018.02.026
0743-7315/

consuming. In fact, running just a millisecond of execution on real hardware can take several hours with the simulator.

To deal with such problems, we propose lab sessions where students work on real hardware. The proposed labs make use of the hardware performance counters implemented in most recent processors from the major manufacturers Intel [14], AMD [9], IBM [18] or ARM [2]. Performance counters consist of a set of special purpose registers that allow tracking advanced processor events such as committed instructions, run cycles, memory accesses, or branch misses, among many others.

In summary, this paper presents a new approach to study computer architecture topics at lab sessions focusing on real hardware and makes two main contributions.

- First, we present a methodology that can be used as a guide in the preparation of computer architecture labs using performance counters. It is aimed at reducing the long time required to prepare and develop this kind of labs and considers both performance counters and common benchmark suites used in research. The methodology employs an adapted version of a research framework that has been successfully used in PhD thesis at our research group, and the proposed lab sessions are based on the authors' expertise acquired while doing research on computer architecture in commercial processors during the last decade. While performance counters have been widely used in current research, to the best of our knowledge this is the first time that their use is applied to computer architecture labs.
- Second, we discuss five lab examples covering different levels of difficulty, depending on the course level. The main novelty of these labs is that they study the different topics by measuring multiple hardware events related with the topics under study on current commercial processors. The presented labs are aimed at illustrating how lab sessions for computer architecture courses based on real machines can be designed. Instructors can use, if they consider appropriate, either a subset of the proposed labs or design their own labs. Nevertheless, we would like to remark that these labs do not intend to replace simulators, but both kind of labs are orthogonal and fit a different range of learning goals.

The remainder of this work is organized as follows. Section 2 motivates the use of real machines for the proposed labs. Section 3 discusses how computer architecture courses are typically organized at universities. Section 4 presents the proposed methodology. Section 5 introduces the scheduling framework used at labs. Section 6 discusses the proposed labs. Section 7 provides some evidences for evaluating the proposed methodology. Finally, Section 8 presents some concluding remarks.

2. Motivation

Simulation has been, and continues being, an extensive methodology widely used across computer architects for research purposes. Detailed cycle-by-cycle simulators model what happens at the different processor stages every processor clock, which helps researchers to precisely understand how the processor and its internal mechanisms work. Because of this reason, computer architects either develop their own simulators or use other simulators widely spread across the scientific community such as [6,21,16].

The use of simulators has probably been the best way, if not the only one, to go deeper into the study of computer architecture topics, either for research or teaching purposes, in the last two decades. Thus, simulators have been widely used by professors and instructors both in the academia and the industry. To avoid the difficulties of complex simulation frameworks in lab sessions, a wide range of *in house* simulators have also been developed by instructors at universities such as DLX [5]. These simulators provide different complexity levels depending on the learning requirements, and usually provide some kind of graphical representation (e.g. display of simple pipelines) to ease the understanding of computer architecture topics.

However, as processors become more advanced and computer architecture courses need to cover the latests features of recent processors, simulators inevitably get more and more complex (e.g. they model hardware prefetchers, memory controllers, cache coherence protocols, etc.). Hence, simulation frameworks that model current advanced processors present two important disadvantages: (i) their complexity translates into a very long learning phase that is not adequate for undergraduates, and (ii) due to the fast evolution of current processors they often fail to model the newest advances of all the system components (e.g. the main memory or the network on chip).

In addition, writing a simulator that mimics the behavior of a specific machine is almost an impossible task. A major drawback that must be overcome is that many hardware details are not publicly available. Thus, the simulator developer may implement hardware functionality incorrectly. Besides, many components that significantly affect system performance like the main memory, the LLC caches, or their replacement algorithms, along with their interactions, need to be modeled. Many times researchers join different simulators (e.g. for the core and for the main memory) with the aim of more accurately modeling the entire system, but even so they are only able to model a machine that is still quite different from the real hardware.

In summary, on the one hand simulators fail to precisely model all the machine components (e.g. the main memory, the interconnects, the cores, the LLC replacement algorithm, etc.) and even more their most advanced features. On the other hand, the high number of simulated components makes simulation impractical for studying current multicores in lab sessions with a limited length (e.g. around two hours). Given the previous rationales, we believe that students need to use real hardware to obtain accurate and precise results and understand how the different parts of the machine interact among them.

3. Computer architecture courses in computer engineering curricula

3.1. General overview

Most Computer Engineering curricula include an introductory course, e.g. Computer Organization, where the basics of computer systems (i.e. the arithmetic unit, the processor pipeline, the memory system, the I/O unit, etc.) are introduced to the students. Then, at least two courses (e.g. Computer Architecture I and Computer Architecture II) study computer architecture topics in depth. The former computer architecture course is usually a core course in the curricula, while the second one covers more advanced topics and can be offered either as core or elective course depending on the syllabus.

Computer Architecture I courses cover a wide range of computer architecture topics such as superscalar architectures, branch prediction, out-of-order execution, or the memory hierarchy, among others. They might also introduce more advanced topics such as multicore or multithreaded architectures and/or interconnection networks. Instructors usually teach students to identify the main components of the system (i.e. the processor, the caches, and the main memory) and how different designs for each of these components impact on performance.

Computer Architecture II courses typically focus on parallel architectures or very advanced topics. Hence, they go further in the study of multicore and multithreaded architectures, as well as,

memory hierarchy particularities intended for these architectures. Other architectures such as dataflow and/or GPU architectures, and advanced topics of speculation and interconnection networks can also be covered in these courses.

In addition, most universities also offer computer architecture courses in a post-graduate master degree. These courses are usually directly focused on particular topics or architectures such as multicore architectures, GPU architectures, or networks on chip, among others. Besides, instructors usually give a research-oriented approach to these courses and lectures are based on research articles published at top conferences that introduce or built upon the covered topics. Thus, they can be considered as the most advanced courses taught on computer architecture.

In this work, we propose a methodology to study and reinforce the knowledge of different topics covered in all these types of courses through lab sessions carried out on real processors. The different courses, obviously, present different levels of difficulty derived from how advanced the topics are and how deeply they are studied. Following the proposed methodology, lab sessions can be prepared and/or adapted to study the topics corresponding to each course level.

3.2. Introduction of the proposed labs at the UPV architecture courses

The computer architecture courses in the Computing Engineering degrees of the Universitat Politècnica de València (UPV) follow the organization described above. A basic course in computer architecture, called *Computer Structures*, is taught in the second year of the Computer Engineering degree. This course is followed by two more advanced computer architecture courses. The third-year course, called *Architecture and Computer Engineering*, is a core course in the year, while the fourth-year course is called *Advanced Architectures* and is an optional course for students. The master degree in Computing Engineering offers the most advanced courses on computing architecture taught at the UPV. *Architecture and Technology of Multicore Processors* is the core course that covers computer architecture topics for post-graduate students, while other courses cover topics related to high performance systems and on-chip networks.

We have applied the proposed methodology in two of the computer architecture courses offered at the UPV during the 2016–2017 academic year: *Advanced Architectures* and *Architecture and Technology of Multicore Processors*, and we plan to introduce it in the *Architecture and Computer Engineering* in the next year. The two courses that apply the methodology are described below.

- *Advanced Architectures*. This course has a duration of 45 h, 30 of which correspond to theoretical sessions and the remaining 15 to labs. We have carried out one lab session of two hours following the proposed methodology. The lab session covered the topics of hardware prefetch and issue stalls, and it is discussed in Section 6.2.
- *Architecture and Technology of Multicore Processors*. This course has a duration of 40 h, 30 of which correspond to theoretical sessions and the remaining 10 to labs. Three lab sessions were performed in this course following the proposed methodology. These labs addressed the cache hierarchy performance and its relation with system performance (described in Section 6.1), the study of bandwidth contention through the memory hierarchy (discussed in Section 6.3), and intra-thread interference and process to core allocation policies in simultaneous multithreading (SMT) processors (explained in Section 6.5).

3.2.1. Labs equipment

The proposed labs leverage the performance accounting capability of current processors to work on computer architecture topics through experiments performed on real systems. Hence, it is required that all the computers in the labs incorporate the same processor, which must also be relatively recent to support monitoring as many hardware events as possible. Fortunately, most processors built during the last five years are able to monitor the hardware events used in the proposed lab sessions. Obviously, a lab intended to study SMT processors can only be carried out in a processor with this multithreading architecture.

Lab sessions of computer architecture courses at our university are taught in three different labs. Two labs have twenty personal computers equipped with Intel i5-3570 and Intel i5-4590 processors. Both of them are multicore processors with four single-thread cores. The third lab has twelve computers equipped with Intel i7-6700 processors. It is also a quad-core processor, but with SMT cores. Depending on the number of students of the course and the processor requirements (e.g. SMT processors), lab sessions are scheduled on an adequate laboratory.

Regarding performance counters capabilities and focusing on the proposed labs (see Section 6), all the mentioned processors are able to measure the same events. Table 1 presents a list of events used in the proposed labs and a brief explanation of what they account for. As it can be observed, most of them are basic events that should be available in most current processors independently of their manufacturer.

4. Proposed methodology

The proposed labs follow an active learning methodology [1,10], leaving students high responsibility in their learning process. We think that instructors should act as resource provides and guide students through the experiments that should be performed. However, students should take the primary role on the analysis and interpretation of the results. This is the key part that will help students achieve a high understanding of the studied topics. Active learning methodologies, additionally, allow students to develop cross-curricular skills such as critical analysis skills.

A major contribution of this work is that the learning process is experienced with real hardware. Working on real machines presents several drawbacks that should be overcome by the proposed methodology. The following guidelines address the different issues that rise when working on real hardware, avoiding unnecessary distractions and making it easier to carry out the proposed labs. Hence, students can strictly focus on the computer architecture topics under study.

First, performance monitoring capabilities differ among commercial processors of different microarchitectures (even for the same manufacturer) and thus, not all the labs we propose in this paper can be done on any machine. Although most recent processors should be able to monitor the hardware events used in the proposed labs, instructors must first check if the hardware events required to perform a lab session are available or not. In addition, some labs require processors that implement specific features such as the labs intended to study SMT processors. Therefore, the instructor should make sure that a given lab can be carried out. Since current processors are able to measure hundreds of hardware events, it is important that the instructor provides some guidance to the students about the events to be monitored. To prevent students from wasting time looking for the events that should be measured, we recommend to provide the precise set of events to be monitored in each lab session, or at least, a superset of them from which students can choose the correct ones.

Second, some sort of framework is required carry out the proposed experiments quickly and precisely. Performance monitoring libraries, such as *libpfm*, offer multiple functions to facilitate

Table 1
Overview of some of the events measured in the lab sessions.

Event	Description
unhalted_core_cycles	Count core clock cycles whenever the clock signal on the specific core is running (not halted).
retired_instructions	Count the number of instructions at retirement.
perf_count_hw_cache_l1d:access	L1 data cache hit access.
perf_count_hw_cache_l1d:miss	L1 data cache miss access.
l2_rqsts:all_demand_data_rd	Demand data read requests to L2 cache.
l2_rqsts:all_demand_data_rd_hit	Demand data read requests that hit L2.
l2_rqsts:all_pf	Any L2 HW prefetch request to L2 cache.
l2_rqsts:pf_hit	Requests from the L2 hardware prefetchers that hit L2 cache.
l2_rqsts:all_rfo	Any RFO (read for ownership) requests to L2 cache.
l2_lines_in:e	L2 lines allocated in E (exclusive) state.
l2_lines_in:s	L2 lines allocated in S (shared) state.
llc_references	Count each request originating from the core to reference a cache line in the last level cache.
llc_misses	Count each cache miss condition for references to the last level cache.
cycle_activity:stalls_l1_pending	Stalled cycles with pending L1 data load cache misses.
cycle_activity:stalls_l2_pending	Stalled cycles with pending L2 miss loads.
cycle_activity:stalls_ldm_pending	Stalled cycles with pending memory loads.
cycle_activity:cycles_no_execute	Stalled cycles where no instructions are dispatched to the execution ports.

the configuration of performance counters and monitoring of a set of events. In addition, *libpfm* also offers command line tools to monitor a single application. However, a scheduling framework is needed when multiple applications are to be launched and monitored concurrently. To face this problem the proposed methodology provides a scheduling framework to the students, as discussed in Section 5. The framework allows them to launch the experiments easily and concentrate on the analysis of the results, which is the key part of the proposed labs.

Third, the framework is introduced in an increasing level of difficulty depending on the course level. For lab sessions aimed at basic and intermediate courses, the framework can be seen as a *black box*. In this case, students only need to feed the framework with the correct inputs to carry out the proposed experiments, and then collect the output event counts to analyze the required metrics. On the other hand, in lab sessions aimed at advanced courses, students can modify the different modules of the framework to implement, for instance, new scheduling algorithms guided by performance monitoring.

Fourth, in order to obtain representative values when running experiments, the proposed labs should be as close as possible to real experiments performed in research. That is, we should use similar workloads instead of *toy applications* or synthetic benchmarks. With this aim we use applications from the SPEC CPU2006 benchmark suite, even though benchmarks from other suites can also be used. However, directly using these benchmarks can be inappropriate for a relatively short lab session length since many of them present long execution times. To deal with this drawback and allow performing multiple experiments in the lab sessions, the scheduling framework is able to limit the benchmarks' executions to the time needed to complete a given number of instructions (e.g. the instructions required to run the application alone during 1 minute). Finishing the benchmarks when they complete a given number of instructions instead of when they have run a given number of seconds allows comparing the same execution part in experiments with different configurations.

Fifth, the methodology proposes experiments running a single application and multiple applications concurrently (multiprogram workloads). In the former scenario, the performance and different metrics related with the computer architecture structures under study are analyzed, identifying their relation if it exists. In the later scenario, the experiments study either inter-core interferences in the resources shared among different cores (e.g., LLC or main memory) or intra-core interference within SMT cores.

In the computer architecture courses where we have applied this methodology, we offer students multiple voluntary course projects on different architecture topics. The projects present a research-oriented approach and make use of the scheduling framework to study novel topics within the research lines of our group. The projects can become *final degree thesis*, required to earn a graduate or a master degree at our university. With these projects, we offer students a project-based methodology. This methodology has been already used in computer architecture [15,17] and other areas [4,8], and has been shown to improve multiple students skills such as critical thinking, problem solving, sustained inquiry or collaboration, among others.

5. Framework features

To apply the proposed methodology, we developed a scheduling framework.[1] This framework is a simplified version of the scheduling framework designed in the Group of Parallel Architectures (GAP), developed for research purposes and used in many top-conference papers [12,20]. The framework is a key component to carry out the lab sessions. Basically, it consists of a user-level scheduler that controls the execution of a workload on the defined cores while gathering multiple hardware events. The main aim of the framework is to simplify the development of the lab sessions and to allow students to concentrate on the architectural concepts under study instead of on the specific issues of the experimental environment.

The basic operation of the framework is as follows. When an experiment starts, the framework launches the workload applications and initializes performance counters, setting the events that should be monitored. As introduced before, performance counters are configured and read using the functions provided by the *libpfm* library. Then, the framework enters a loop where the following steps are repeated:

1. The process selection policy selects which applications should run during the next quantum.
2. The process allocation policy assigns the applications to the cores using the CPU affinity mask of the Linux processes.
3. Applications run during the quantum length.
4. Once the quantum expires, the framework stops the running applications and reads their performance counters, whose values are printed if required.
5. If the workload execution is completed, the framework prints overall event counts and exits. Otherwise, it goes to step 1.

[1] The framework source code can be downloaded at https://github.com/jofepre/lab_sched_framework.

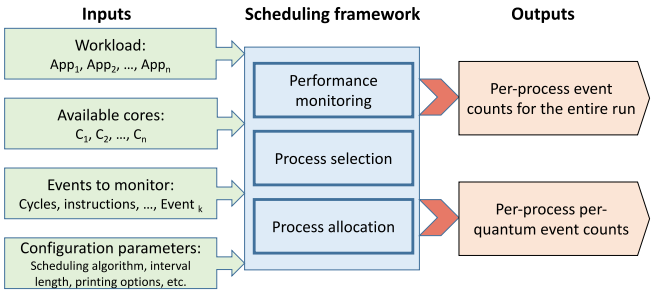

Fig. 1. Framework block diagram.

In lab sessions aimed at basic or intermediate courses, the framework can be used as a *black box* (see the *Lab Example 1*, presented in Section 6.1). In these lab sessions, students only need to configure the input parameters of the framework, which can be observed in Fig. 1, and then analyze the experimental results. The main parameters that can be configured are as follows: (i) the workload to be run, (ii) the cores to be used, (iii) the hardware events to be monitored, and (iv) the scheduling policy (process selection and process allocation).

The workload refers to the application or set of applications whose behavior is going to be monitored. The cores to be used indicate the set of cores where the applications will run. This parameter is critical in some architectures. For example, depending on the system configuration, two applications can run on cores sharing a given cache or not. Similarly, the cores configuration in SMT multicores determines if two applications run in the same SMT core or in different cores. The configured events to be monitored determine the metrics that will be analyzed. Regarding the scheduling policy, the framework also allows configuring algorithms for selecting which applications will be executed each quantum (if the number of applications exceeds the number of cores) and where (i.e., in which core) they will be executed. Finally, other parameters such the quantum length or the granularity at which event counts are printed can also be configured.

The framework is composed of three main modules according to the function they carry out: (i) *performance monitoring*, (ii) *process selection*, and (iii) *process allocation*, as depicted in Fig. 1. This abstraction level simplifies the modifications that can be proposed in advanced lab sessions. In the performance monitoring module, new performance metrics can be calculated from the events counts to guide the scheduling, which is usually performed in two steps: process selection and process allocation. For each step, different policies can be implemented in its corresponding module.

The modular structure of the framework facilitates modifying its source code since it hides the complex internal management of the processes (create, stop, and continue processes) and performance counters (configuration and reading). In this way, it is possible to implement a simple bandwidth-aware scheduling policy within a lab session length (or a couple if required), as proposed in the *Lab Example 4* (see Section 6.4).

6. Proposed labs

The proposed labs are aimed at studying the impact of the major system components on performance. System performance is typically quantified in terms of *instructions per cycle* or IPC, which can severely be damaged as processor *stalls* rise, limiting the number of instructions issued in a given cycle. Stalls appear because of data, control, or structural dependences. The major performance limiter in current systems is the memory subsystem, including the on-chip cache hierarchy and the external main memory. Because of this reason, the devised labs focus either on a given part of the memory subsystem itself or on specific mechanisms that end up affecting its performance (e.g., the prefetcher or the issue logic). In addition, some labs are devoted to study SMT multicores, which are the unique processors that share computational resources among multiple threads within a core.

The general organization of the lab sessions consists of the following parts:

1. Description of the problem and learning goals.
2. Summary of the theoretical aspects focusing on how they are implemented in the commercial processor used at labs.
3. Introduction of the involved hardware events.
4. Configuration and launch of experiments.
5. Graphical representation of results.
6. Analysis of results and conclusions.

We provide students with a booklet that guides them through the lab session. Such a booklet should be adapted to the course level, being more specific in the steps to be performed when the lab session is intended for under-graduate students. However, the booklet cannot replace the instructor, who plays a key role describing the problem, learning goals, and theoretical background, as well as, guiding students towards the correct interpretation of the experimental results.

The booklet guides students on the experiments that should be carried out and how to prepare the report to assess the lab session. This report must be delivered to the instructor, including both the obtained results and the requested analysis. It is emphasized the need of presenting clear figures that ease the interpretation of the results.

To illustrate the wide scope of the proposed approach, we present five lab sessions that cover multiple computer architecture topics intended for different course levels.[2] Note that current performance counters can monitor hundreds of hardware events, allowing the study of other components that are not studied in the proposed labs. For instance, our research scheduling framework already supports the execution of parallel applications and we plan to prepare new lab sessions with these applications in the next years.

6.1. Lab Example 1. Understanding the basics on cache hierarchy performance and system performance (basic level)

Lectures typically study cache basics focusing on internal cache organization and working behavior. However, when teaching about the different cache levels, instructors make scarce or no difference among them, with some exceptions (e.g. cache size, associativity, or access time). In some cases, it is taught that L1 caches are intended to provide as fast as possible access to the most recently used data (designed for performance), whereas LLCs caches focus on avoiding main memory accesses to data with less locality (designed for capacity). Unfortunately, these ideas are only covered from a theoretical perspective, which makes it hard for students to understand and differentiate the specific role of each cache level on the overall performance, as well as the possible relationship between cache performance and processor performance. This lab session is aimed at dealing with these learning issues.

6.1.1. Learning goals

This lab session pursues to achieve the following learning goals:

- Familiarize with cache performance metrics; e.g. cache misses per kilo-instruction (MPKI), bandwidth (BW), and hit ratio (HR).

[2] The booklet and files related to each lab session can be downloaded at https://github.com/jofepre/lab_sched_framework/tree/master/lab_sessions/.

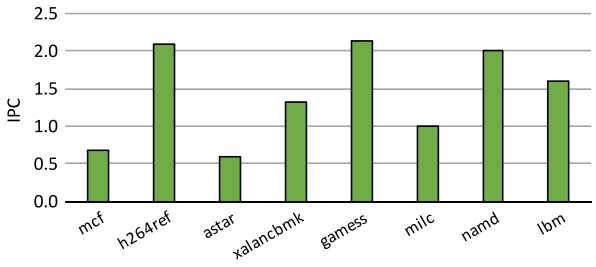

Fig. 2. IPC of a subset of benchmarks.

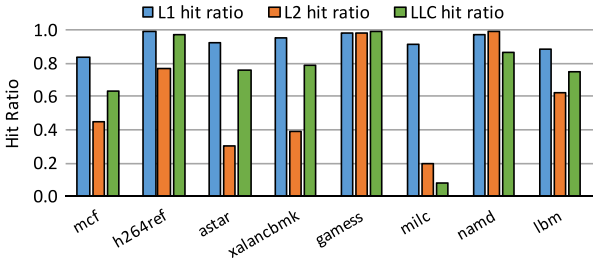

Fig. 3. L1, L2, and LLC hit ratios of a subset of benchmarks.

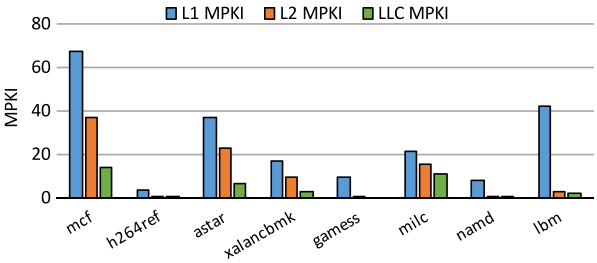

Fig. 4. L1, L2, and LLC MPKIs of a subset of benchmarks.

- Understand the role of each cache level in system performance.
- Identify the relationship between cache and system performance.

6.1.2. Session development and discussion

The instructor starts the session refreshing the memory hierarchy organization concepts studied at lectures, but focusing on the memory hierarchy of the processor equipped at the lab's computers. After that, and assuming that this is the first lab session of the course, an overview of the framework and its use should be introduced.

Students typically work with hit ratios when studying cache basics. However, this metric usually yields students to misleading conclusions when trying to identify its relation with processor performance. Instead, MPKI is a metric that correlates better with performance and thus, it is typically used in research papers.

To get familiarized with these performance metrics and their typical values in current systems, students are asked to obtain the cache hit ratio and MPKI at each cache level, that is, HR_{DL1}, HR_{L2}, HR_{LLC}, $MPKI_{DL1}$, $MPKI_{L2}$, and $MPKI_{LLC}$. To perform this task, they use the scheduling framework as a *black box* that is fed with the appropriate inputs, which are the benchmarks to be run and the hardware events to be monitored. As explained before, these events can either be directly provided by the instructor or identified by the students from a provided subset of the available events.

Once the events to be monitored are identified, the launch of the experiments can be automatized in a bash script, which can be completely coded by students or partially provided to them. After running the experiments, students must collect the event counts for each application and calculate, for each benchmark, the IPC as well as the HR and MPKI for each cache level. Then, the studied metrics should be plotted (using an spreadsheet software or *gnuplot*) to be analyzed.

As an example, Figs. 2, 3 and 4 present, respectively, the IPC, the HR, and the MPKI of the three cache levels (L1, L2, and LLC) implemented by the processor used at our labs across a subset of SPEC2006 benchmarks. As observed, HRs and MPKIs widely differ. Regarding the hit ratios, Fig. 3 shows that the L1 data cache hit ratio is really high and catches, on average, more than 85% of the memory accesses, while L2 captures less than 40% of its accesses, and the huge LLC resolves around 75% of the L2 misses. This picture illustrates (i) the high locality exhibited in the small L1, (ii) the rather poor data locality in the much larger L2, and (iii) the importance of the LLC in the system, since it avoids a significant percentage of the memory accesses. However, there is not a clear relation between the obtained hit ratios and the processor performance. For instance, `astar` shows high hit ratios, which resemble the hit ratios achieved by `xalancbmk` but its IPC is below 0.6 while `xalancbmk` achieves an IPC of 1.4.

Next, students are asked to identify the relation between the system performance and the MPKIs. Important conclusions can be drawn with this comparison. For instance, the higher the MPKI of the LLC, the lower the IPC. It can be observed that `mcf`, `astar`, and `milc`, which are the three applications with the highest $MPKI_{L3}$ are those also showing the lowest IPC. With this kind of experiments and analysis, students realize about the importance of L3 caches, which comes from catching many memory requests.

6.2. Lab Example 2. Prefetching and issue stalls (intermediate level)

Current microprocessors implement aggressive hardware prefetchers at the multiple levels of the cache hierarchy to bring the data that will be requested soon closer to the processor. Prefetching is typically covered in lectures by focusing on the distinct components of a hardware prefetcher, how patterns are detected, and when new prefetches are triggered. Instructors emphasize that prefetching can highly hide the main memory access time, which is critical in current microprocessors, but again, the gap between theory and practice is not covered in most courses.

On the other hand, regarding *stalled* cycles, lecturers usually explain that *stalls* can appear at different stages of the processor pipeline, such as the dispatch and issue stages, and that they are strongly related with performance losses. Most dispatch stalls rise when instructions cannot be dispatched due to there is no free entry in the re-order buffer (ROB) or the issue queue. In contrast, most issue stalls occur when no instruction can be issued due to data hazards (e.g., due to a cache miss).

Unfortunately, lectures typically quantify the wasted issue slots, but do not discuss how prefetching may affect the number of stalls. To provide a sound understanding of both prefetching and stalls from a practical perspective, we combine their study in this lab session. This lab pursues, first, that students realize how prefetching can significantly rise the performance of a complex out-of-order processor (e.g. by a 3x factor), and second, that students familiarize with stalls as a way to quantify performance losses and analyze how prefetching is able to reduce the number of stalled cycles.

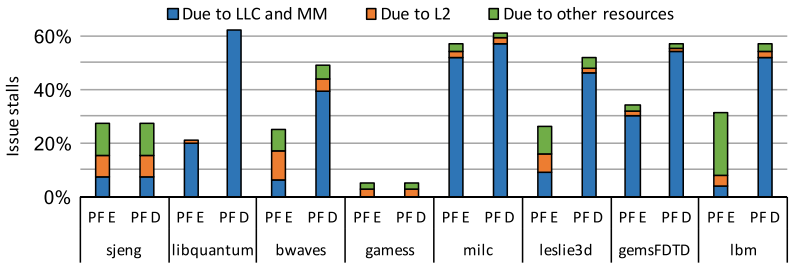

Fig. 5. Issue stalls of a subset of benchmarks with prefetch enabled and disabled.

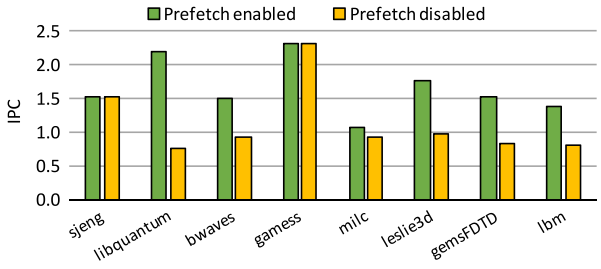

Fig. 6. IPC of a subset of benchmarks with prefetch enabled and disabled.

6.2.1. Learning goals

This lab session pursues to achieve the following learning goals:

- Reinforce the knowledge about how out-of-order processors work, focusing on the issue stage. Identify stalls as performance constraints.
- Characterize the issue stalls of a subset of benchmarks running on a real machine.
- Study the connection between issue stalls and processor performance.
- Learn how prefetching can be configured at runtime.
- Study the performance degradation that disabling the prefetcher causes to different applications in a real machine, and how performance losses are reflected as issue stalls.
- Analyze the characteristics of applications with respect to the benefits (or lack of them) that prefetching has on their performance.

6.2.2. Session development and discussion

The instructor starts the session reviewing the instruction flow on a generic out-of-order processor, and then, focuses the discussion on the execution pipeline of the processor available in the labs. The issue slots are identified and the issue slots waste concepts (horizontal waste and vertical waste) are refreshed. After that, the hardware events that the processor exposes to the performance monitoring unit related with the issue logic are discussed.

Students continue the development of the lab session following the booklet. First, they are asked to monitor the issue stalls of a subset of applications. To this end, they need to feed the scheduling framework with the appropriate inputs (i.e., stall-related and memory-related events). Using the collected event counts, students obtain and represent the percentage of issue stalls relative to the number of cycles, as well as the IPC of the applications.

Fig. 5 plots, in the *PF E* labeled bars, the issue stalls breakdown of the prefetch-enabled system configuration (default). Issue stalls are broken down according to three main causes: memory accesses resolved by the L1 or L2 caches, memory accesses resolved by the LLC or main memory, and other causes (e.g. dispatch stalls). It can be appreciated that the memory structures (i.e. the two former ones) represent a significant fraction in some applications that are memory intensive such as `milc` or `libquantum`, while this fraction is smaller in other benchmarks like `sjeng` or `lbm`. Students are asked to identify these aspects. Fig. 6 shows the performance of the applications. Comparing the IPC and stalls of the prefetch-enabled bars in Figs. 5 and 6, respectively, students realize that the achieved performance can be closely estimated in real machines by measuring the issue stalls.

Once students have realized that the stalls related to memory events are the dominant ones, we follow working on prefetching as a way to unclog the performance by reducing the issue stalls. With this aim, the instructor discusses the prefetching capabilities of the target processor. For instance, the Intel processors available in our labs implement four hardware prefetchers that can be enabled/disabled at runtime [14]. Next, the instructor explains how the prefetchers can be configured at runtime. On the Intel and AMD processors they can be enabled and disabled through a machine specific register (MSR), which can be read or written in Linux using the *rdmsr* and *wrmsr* commands, respectively.

Next students disable the prefetchers and obtain the same metrics that were measured with the prefetchers enabled. Figs. 5, 6 and 7 also show, labeled as prefetch disabled or *PF D*, these results for stalls, IPC and MPKI, respectively. Students are asked to compare *PF E* and *PF D* configurations and observe how enabling the prefetchers highly reduces the MPKI, which translates into an important reduction in the corresponding components of the *issue stalls* bars, and consequently, in a great performance improvement in some applications. This is the case of `libquantum` whose performance is boost by a 3.1x factor. Looking at these values, students realize why prefetching is so important for attacking the memory bottleneck. In other words, without these mechanisms, it would make no sense to improve the performance of computational cores, since the memory subsystem would strangle the system performance.

6.3. Lab Example 3. Inter-thread interferences: bandwidth contention through the memory hierarchy (intermediate level)

The previous labs have focused on single-threaded applications running alone in the system. However, multicore processors typically run multiprogram workloads where the performance of individual applications is harmed, with respect to their isolated execution, due to the inter-thread interference. That is, instructions from multiple concurrently running applications (co-runners) compete among them in the access to the shared resources, which translates into performance losses. The intensity of the interference depends on the shared resource demands of the co-runners, which dynamically vary at run time. In single-threaded cores, which is the focus of this lab, the interference rises in shared uncore resources, mainly the LLC and main memory. This lab takes a close look at how applications interfere in these resources and studies how contention at the LLC and main memory impact on the overall performance.

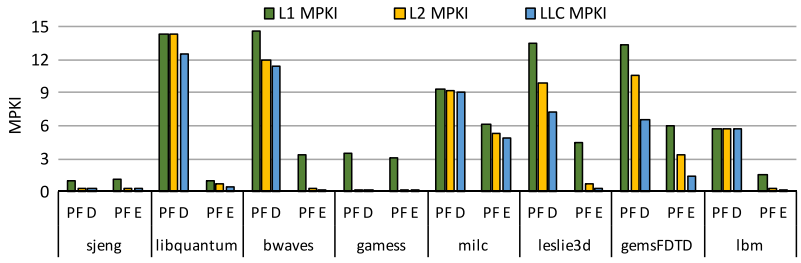

Fig. 7. L1, L2, and LLC MPKIs of a subset of benchmarks with prefetch enabled and disabled.

6.3.1. Learning goals

This lab session pursues to achieve the following learning goals:

- Realize to what extent thread interference at main memory can damage performance.
- Understand how important reducing cache contention is for performance.
- Work with synthetic applications (microbenchmarks) and understand how they can be used to model certain co-runner behaviors.
- Study the performance degradation due to bandwidth contention.

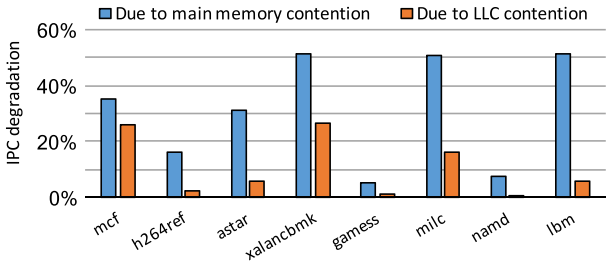

Fig. 8. Performance degradation due to main memory and LLC bandwidth contention.

6.3.2. Session development and discussion

At the beginning of the session the instructor refreshes key concepts about resource sharing focusing on the LLC and the main memory, which are the shared resources addressed in this lab session. It might also be required to remind memory-related hardware events that should be monitored during the experiments.

First, the *main memory bounded* microbenchmark is introduced. This microbenchmark is a synthetic program that is provided to the students and causes main memory contention by allocating a huge memory area, which is randomly accessed to avoid prefetch hits. Nonetheless, depending on the course level and the lab session length, the instructor can go deeper explaining how the microbenchmark works looking at its source code.

Students continue the lab session guided by the booklet. They characterize the microbenchmark to experimentally confirm its expected behavior; that is, a high main memory demand (i.e. high LLC MPKI) and an almost zero LLC hit ratio. Next, they launch several experiments where each benchmark runs jointly with *Num_cores* − 1 microbenchmarks, maximizing the main memory bandwidth contention.

Once the experiments are completed, students analyze the performance degradation of each application, which is obtained by comparing the IPC of the applications in the experiments over their IPC running alone. As an example, Fig. 8 shows the performance degradation of a subset of benchmarks due to main memory bandwidth contention (blue bars). In order to understand the specific impact on performance for each benchmark, students should compare these results with the LLC MPKI in isolated execution, which can be obtained previously (these results were already presented in Fig. 4). Using these figures, students are asked to provide a rough analysis with some quantitative values about the relationship between IPC degradation and main memory interference. For instance, looking at both figures it can be appreciated that the five benchmarks with higher MPKI at main memory are the ones that experience the highest IPC degradation. This performance drop ranges between 30% and 50%; that is, some benchmarks double their execution time.

Next, the session focuses on LLC bandwidth contention. The instructor can provide the students an *LLC bounded microbenchmark* or guide them to develop it from the source code of the main memory bounded microbenchmark. The LLC bounded microbenchmark must always miss in the L2 and hit in the LLC. This behavior can be achieved by configuring its allocated memory space and access pattern according to the LLC cache geometry [22]. Once this microbenchmark is developed, students analyze the performance degradation of the applications due to LLC bandwidth contention, similarly as how it was analyzed for main memory bandwidth contention. First, they verify that the LLC bounded microbenchmark works as expected. Then, the experiments to measure the performance degradation due to LLC bandwidth contention are launched. Fig. 8 presents the performance degradation of a subset of benchmarks due to LLC bandwidth contention, and Fig. 4 showed their L2 MPKIs. Students may conclude (with some guidance if required) that the five benchmarks that most frequently access to the LLC are the ones that experience the highest IPC degradation due to LLC contention. In this case, performance drops between 8% and 30%.

Notice that the performance degradation values obtained in this session are really significant (i.e. up to 50% due to main memory contention and up to 30% due to LLC contention), which makes students understand to what extent the memory subsystem can strangle system performance. This fact is usually explained at lectures, but it is important that students realize how serious this problem can be from a practical experience.

6.4. Lab Example 4. Main memory bandwidth-aware scheduling (advanced level)

In the *Lab Example* 3, we focused on how the interference among processes on the shared resources of the memory hierarchy impacts on performance. This is in line with what instructors teach in computer architecture lectures. However, once students understand these interactions, they can go a step further and learn that by running together *friendly or symbiotic* applications that present low run-time interference in the shared resources, contention can be reduced and performance improved. This lab addresses this issue by designing a main memory bandwidth-aware scheduler.

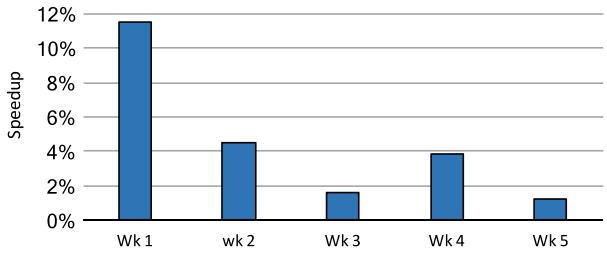

Fig. 9. Speedup of the main memory bandwidth-aware scheduler when running the evaluated workloads.

6.4.1. Learning goals

This lab session pursues a threefold goal:

- Realize how improving resource utilization yields to important performance benefits.
- Understand why architecture-aware scheduling can boost the performance of the systems.
- Minimize the interference through a simple scheduling policy and quantify the performance benefits it provides.

6.4.2. Session development and discussion

The instructor begins the lab session explaining its contents and learning goals. Then, the main parts of the scheduling framework are described both conceptually and going through its source code. The development that students must perform in this session is not too complex, but the instructor should give a clear overview of the framework in order for students to reach a successful implementation.

After the instructor's explanations, students start working on the performance monitoring module of the scheduling framework. They should configure the memory-related events to be monitored and implement the computation of the main memory bandwidth utilization, which will be updated by the scheduler after each quantum. The instructor can discuss the choice of a bandwidth utilization metric instead of other memory-related metrics like MPKI. Bandwidth is widely affected by interference, which makes it a better index to guide the scheduling.

To facilitate the implementation of the new policy, we assume a simple scenario consisting of two processor cores and four processes to be scheduled. Hence, the scheduling policy should select two processes to be run at each quantum. To minimize bandwidth contention, the following policy is proposed. The scheduler should select to be run each quantum the couple of processes with highest and lowest bandwidth utilization. Students should also implement a *worst case* scheduler, which will be used as baseline. Instead of balancing the bandwidth utilization, the worst case scheduler runs concurrently the processes with highest bandwidth utilization, increasing the interference.

Finally, students set the framework to evaluate a few workloads and quantify the performance benefits that the proposed scheduler provides over the worst case one. As an example, Fig. 9 shows the speedup achieved running the five 4-application workloads presented in Table 2 in our lab computers. For workloads with two memory-bounded and two cpu-bounded benchmarks, the speedup can be as high as 11.5%, as observed in workload 1. In other workloads the achieved benefits are lower (e.g., by 2% in workloads 3 and 5).

6.5. Lab Example 5. Process to core allocation in smt processors (advanced level)

Multithreaded processors support concurrent execution of multiple threads (or processes) in the same processor. Among them, SMT processors are the only type of multithreaded processors that are able to issue instructions from multiple threads in the same cycle. Most processor manufacturers implement SMT in their high-end products since it is a promising architecture paradigm that offers excellent performance when running a single process and high throughput when running multiple threads or applications.

Table 2
Workload evaluated in the lab session 4.

Workload	Benchmarks
Wk 1	gamess, h264ref, lbm, milc
Wk 2	astar, mcf, sjeng, xalancbmk
Wk 3	astar, hmmer, lbm, leslie3d
Wk 4	astar, bwaves, gammes, xalancbmk
Wk 5	bwaves, h264ref, leslie3d, milc

Lecturers explain this paradigm but it is difficult to understand how the intra-core interference among co-runners in the same core affects performance. This session pursues students to experimentally realize how important this interference can be and to what extent it can seriously damage the individual performance of the applications over their isolated execution on the same core. Consequently, process to core allocation policies are required to maximize the performance in SMT processors.

6.5.1. Learning goals

This lab session pursues to achieve the following learning goals:

- Identify the connection between L1 bandwidth utilization and performance of applications running on an SMT core.
- Study how the L1 bandwidth utilization of an application limits the use of this resource and, consequently, the performance of other applications running simultaneously in the same core.
- Understand how the performance of SMT processors can be boosted by an intelligent thread-to-core allocation policy.
- Design a simple bandwidth-aware thread-to-core allocation policy.

6.5.2. Session development and discussion

To accomplish the learning goals, this lab session comprises multiple experiments. According to the instructor's criterion and the available time in the course, it is possible to (i) limit the experiments to carry on the lab in a single session, (ii) extend the lab through a couple of sessions, allowing a deeper analysis of the effects of the intra-thread interference, or (iii) split the lab session in two parts and cover them in different courses.

At the beginning of the session, the instructor summarizes the main concepts about SMT processors, reminding their ability to launch multiple instructions of different threads in the same cycle. The following experiments, guided by the booklet and the instructor advises, will lead students to analyze and understand the effects of intra-thread interference. Such concepts are more easy to assimilate from a practical experience than from a theoretical point of view; hence the importance of this lab.

First, students analyze the relation between the L1 bandwidth utilization of the applications and their performance. The connection between both metrics can be observed by looking at the average L1 bandwidth and performance of multiple applications. However, the effect is much more noticeable by looking at how these metrics evolve dynamically along the execution time. The scheduling framework allows obtaining per-quantum event counts by enabling the per-quantum printing option. Then, the collected data can be processed using a spreadsheet or gnuplot to draw the L1 bandwidth utilization and IPC of a few applications. As

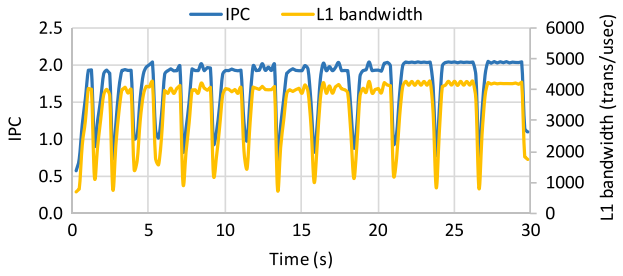

Fig. 10. IPC and L1 bandwidth of the benchmark `bwaves`.

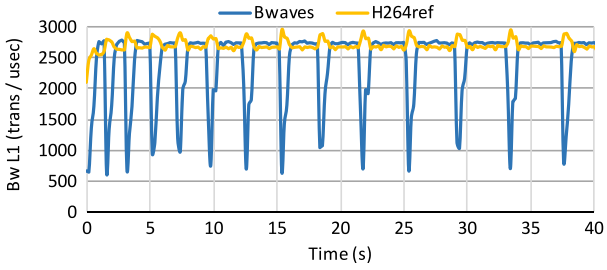

Fig. 11. L1 bandwidth of `bwaves` and `h264ref` running concurrently on an SMT core.

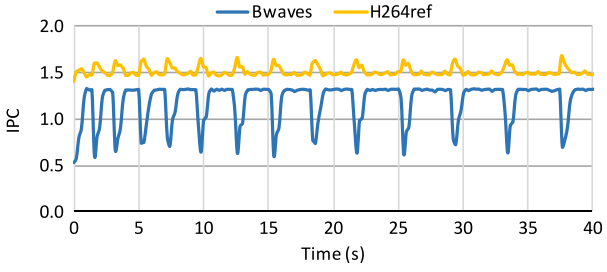

Fig. 12. IPC of `bwaves` and `h264ref` running concurrently on an SMT core.

an example, Fig. 10 shows the IPC and L1 bandwidth of `bwaves`. The connection between both metrics is quite evident, but the instructor should ensure that all students identify and understand it.

Next, the lab continues towards studying how intra-thread interference affects L1 bandwidth and performance of two applications running concurrently on the same SMT core. To run the experiments, students need to configure the framework to run a couple of applications on the same core. Instructors should ensure that students correctly allocate both applications to the same core. Otherwise, no intra-thread interference will occur. To better appreciate the connection between L1 bandwidth and performance, it is also recommended that at least one of the applications present a phase behavior easily identifiable in the plots.

Figs. 11 and 12 present the L1 bandwidth and IPC, respectively, of `bwaves` and `h264ref` running concurrently on the same SMT core. The figures show how the drops in the L1 bandwidth of `bwaves` reduce the intra-thread interference and allow `h264ref` to increment its bandwidth utilization, which presents a uniform value on isolated execution. Notice that the IPC of `h264ref` rises on the periods where the interference introduced by `bwaves` is reduced. At this point, the instructor should emphasize how the interference strongly impacts on performance (it can make the performance benefit of SMT processors nearly worthless), and discuss how a thread allocation policy aware of the intra-thread interference should improve the throughput of SMT cores.

The last part of the lab session proposes the implementation of a thread allocation policy that minimizes L1 bandwidth interference among threads, thus improving performance. To implement such a policy, students need to modify the scheduling framework to (i) calculate the L1 bandwidth utilization of the application at the end of each quantum (performance accounting module), and (ii) implement the new allocation policy on the process allocation module. Implementing the policy is not challenging, but instructors should guide students to skip difficulties and students frustration. The policy should simply allocate the application with the highest and lowest requirements together on the same SMT core, so that the L1 bandwidth is balanced among the cores. As in the *Lab Example 4*, it is not required to implement a policy covering all possible scenarios; for instance, it can be restricted to four application workloads that are run on two SMT cores.

Once the policy is implemented, students evaluate it by comparing its performance with respect to a *random* or a *worst case* process allocation policy. Students should report the achieved performance benefits to make sure that they identify the strong benefit that an interference-aware thread allocation policy can provide to SMT processors [11]. Through this lab session, students understand and reinforce the knowledge and importance of the intra-thread interference in SMT processors, something difficult to teach in a theoretical lecture but at the same time, easy to identify and work on in the proposed lab session.

7. Evaluation of proposed approach

This section provides a qualitative assessment of how the proposed methodology helps students to achieve their learning outcomes. In this regard, three major axes have been considered that provide evidence of the success of the proposed approach. These axes, discussed below, are lab and course grades, evaluation of transversal competences, and satisfaction surveys.

Majority of the students (75%) achieved a grade of A (score ≥ 90 out of 100) in the lab and rest (25%) achieved grade B (80–89 out of 100). Furthermore, we observed that lab marks are strongly related with final course marks. This fact suggests that the designed labs help to clarify and reinforce the concepts taught at lectures.

We also found that the proposed labs help students to develop transversal competencies. The purpose of a transversal competency is to acquire skills related with students' personal development that are not tied to a specific discipline or subject area and are commonly required in both professional and academic domains [13]. There have been several efforts to standardize transversal competences by educational organizations such as ABET [19], ENAEE [3], or ANECA [7]. Based on these efforts, UPV has defined 13 transversal competences [23]: CT-01: Understanding and integration, CT-02: Application and practical thinking, CT-03: Analysis and problem solving, CT-04: Innovation, creativity and undertaking, CT-05: Design and project, CT-06: Teamwork and leadership, CT-07: Ethical, environmental and professional responsibility, CT-08: Effective communication, CT-09: Critical thinking, CT-10: Knowledge of contemporary issues, CT-11: Lifelong learning, CT-12: Planning and time management, and CT-13: Specific instruments.

At UPV, each course must train and evaluate a given set of transversal competences. In particular, ATP is assigned CT-01, CT-10, and CT-11 competences. UPV recommends to evaluate transversal skills through student activities that leverage the subject content. In ATP labs, CT-01 is trained by writing lab reports where students, based on what they learn at lectures, identify cause–effect relationships when analyzing the data gathered in the experiments. On the other hand, working on the hardware of current processors also helps to raise the awareness about specific contemporary issues in the course area of knowledge (CT-10). In

fact, the most advanced labs provide students an opportunity to develop their own solutions to attack some of these issues. Finally, CT-11 is trained by encouraging students to look for and read technical documentation on processor architectures and specific hardware events that can be measured with performance counters as well as by modifying the scheduling framework in the most advanced labs.

Following UPV recommendations, lab activities play a major role in the evaluation of these competences. The evaluation is performed by supervising the students' work at labs and by reviewing lab reports. The assessment corroborates that the proposed methodology is useful to achieve high levels of proficiency in the three assigned competences. At UPV, transversal competences are evaluated apart from the course. The possible grades are as follows: A (excellent), B (fair), C (in development), and D (not achieved). The distribution of grades in ATP was roughly A: 67%, B: 25%, and C: 8%.

With respect to the satisfaction surveys, they are periodically conducted by UPV for every course, allowing students to express their opinion with each course methodology. Regarding ATP course, all the students backed the methodology and activities carried out at lab sessions. In fact, many of them highlighted that the proposed methodology drastically reduces the time required to prepare the lab session compared to simulation-based methodologies, where a huge effort is made to get familiarized and understand the complex simulation framework, and the source code involved to model a given processor structure under study.

Finally, apart from the discussed axes, we found great interest of students in carrying out volunteer course projects based on the proposed framework. In fact, four students (one from a master degree and three from a degree in Computing Engineering) chose (during this academic year) to carry out the proposed projects with the scheduling framework, with the aim of being their *final degree thesis*. One of the students is working on scheduling for real-time systems, another one is doing research on cache partitioning, and the remaining two are working on dynamic configuration of the hardware prefetching mechanisms. Furthermore, three of them have already shown interest in joining our research group and enroll the PhD program in Computer Science. Their effort will provide them an invaluable background with their PhDs.

8. Conclusions

This paper identifies the strengths and weaknesses, from a pedagogical perspective, of using computer simulation frameworks at labs, and presents a new approach to cover a wide range of studies on real machines, which cannot be handled by simulators in time-bounded labs. The proposal, based on the research expertise of the authors, exploits the performance monitoring capability of current processors, which allows the hardware to track many architectural events (e.g., committed instructions, cache misses, issue stalls, etc.). In this paper, we present the methodology, the scheduling framework, and five labs that illustrate the possibilities of the proposed methodology. The examples present distinct ranges of difficulty, and cover topics such as memory hierarchy, hardware prefetching, issue logic, or SMT cores.

We have applied the proposed methodology during the academic year 2016–2017 in two computer architecture courses at the UPV. We found that the proposed approach helps improving the understanding of the theoretical concepts taught at conventional lessons. Furthermore, it really motivates students to continue their education in computer architecture topics. After applying our methodology, three undergraduate students and one postgraduate that followed these courses are currently performing their *final degree thesis* and *master degree thesis*, respectively, in our research team. In addition, two PhD students that were working with simulators in their PhD have moved their work to real machines.

Finally, we would like to emphasize that a key goal of this paper is to serve as a guide to other colleagues to design their labs. With this aim, we have made the source code of the scheduling framework available and presented the lab examples in a rather detailed way.

Acknowledgments

Josué Feliu has been partially supported through a postdoctoral fellowship by the Generalitat Valenciana (APOSTD/2017/052). Additional support has been provided by the Spanish Ministerio de Economía y Competitividad (MINECO) and Plan E funds, under grants TIN2015-66972-C5-1-R and TIN2014-62246-EXP.

References

[1] O. Arbelaitz, J.I. Martn, J. Muguerza, Analysis of introducing active learning methodologies in a basic computer architecture course, IEEE Trans. Educ. 58 (2) (2015) 110–116.
[2] ARM, Cortex-A9. Technical Reference Manual, 2009. Available online: http://infocenter.arm.com/help/topic/com.arm.doc.ddi0388e/DDI0388E_cortex_a9_r2p0_trm.pdf.
[3] G. Augusti, J. Birch, E. Payzin, EUR-ACE: A system of accreditation of engineering programmes allowing national variants, in: INQAAHE 2011 Conference, Madrid, 2011, pp. 4–7.
[4] S.M. Aziz, E. Sicard, S.B. Dhia, Effective teaching of the physical design of integrated circuits using educational tools, IEEE Trans. Educ. 53 (4) (2010) 517–531.
[5] M. Bečvář, S. Kahánek, VLIW-DLX simulator for educational purposes, in: Proceedings of the 2007 Workshop on Computer Architecture Education, 2007, pp. 8–13.
[6] T.E. Carlson, W. Heirman, L. Eeckhout, Sniper: Exploring the level of abstraction for scalable and accurate parallel multi-core simulation, in: Proc. Int. Conf.High Perf. Computing, Networking, Storage and Analysis, 2011, pp. 52:1–52:12.
[7] A.N. de Evaluación de la Calidad y Acreditación, Libro Blanco. Título de Grado en Ingeniería Informática, 2005.
[8] S.L. Dexter, R.E. Anderson, H.J. Becker, Teachers views of computers as catalysts for changes in their teaching practice, J. Res. Comput. Educ. 31 (3) (1999) 221–239.
[9] P.J. Drongowski, Basic Performance Measurements for AMD Athlon 64, AMD Opteron and AMD Phenom Processors, September 2018. Available online: http://amd-dev.wpengine.netdna-cdn.com/wordpress/media/2012/10/Basic_Performance_Measurements.pdf.
[10] I. Estévez-Ayres, C. Alario-Hoyos, M. Pérez-Sanagustín, A. Pardo, R.M. Crespo-García, D. Leony, H.A. ParadaG., C. Delgado-Kloos, A methodology for improving active learning engineering courses with a large number of students and teachers through feedback gathering and iterative refinement, Int. J. Technol. Des. Educ. 25 (3) (2015) 387–408.
[11] J. Feliu, J. Sahuquillo, S. Petit, J. Duato, L1-bandwidth aware thread allocation in multicore SMT processors, in: Proc. 22nd Int. Conf. Parallel Archit. Compilation Tech., 2013, pp. 123–132.
[12] J. Feliu, J. Sahuquillo, S. Petit, J. Duato, Addressing fairness in SMT multicores with a progress-aware scheduler, in: Proc. IEEE 26th Int. Parallel Distrib. Process. Symp., 2015, pp. 187–196.
[13] J. González Ferreras, R. Wagenaar, Tuning Educational Structures in Europe. Final Report. Phase One, Universidad de Deusto. Bilbao, 2003.
[14] Intel Corporation, Intel 64 and IA-32 Architectures Software Developers Manual, 2017.
[15] C.M. Kellett, A project-based learning approach to programmable logic design and computer architecture, IEEE Trans. Educ. 55 (3) (2012) 378–383.
[16] M.M.K. Martin, D.J. Sorin, B.M. Beckmann, M.R. Marty, M. Xu, A.R. Alameldeen, K.E. Moore, M.D. Hill, D.A. Wood, Multifacet's general execution-driven multiprocessor simulator, GEMS, toolset, SIGARCH Comput. Archit. News 33 (4) (2005) 92–99.
[17] A. Martinez-Mones, E. Gomez-Sanchez, Y.A. Dimitriadis, I.M. Jorrin-Abellan, B. Rubia-Avi, G. Vega-Gorgojo, Multiple case studies to enhance project-based learning in a computer architecture course, IEEE Trans. Educ. 48 (3) (2005) 482–489.
[18] S.K. Sadasivam, POWER8 performance analysis, in: OpenPOWER Summit 2015, March 2015. Available online: https://openpowerfoundation.org/wp-content/uploads/2015/03/Sadasivam-Satish_OPFS2015_IBM_031615_final.pdf.
[19] L.J. Shuman, M. Besterfield-Sacre, J. McGourty, The "ABET professional skills"–Can they be taught? Can they be assessed?, J. Eng. Educ. 94 (1) (2005) 41–55.
[20] A. Snavely, D.M. Tullsen, Symbiotic jobscheduling for a simultaneous multithreaded processor, in: International Conference on Architectural Support for Programming Languages and Operating Systems, ASPLOS'00, 2000, pp. 234–244.

[21] R. Ubal, B. Jang, P. Mistry, D. Schaa, D. Kaeli, Multi2Sim: A simulation framework for CPU-GPU computing, in: Proc. of the 21st International Conference on Parallel Architectures and Compilation Techniques, 2012.
[22] Using huge pages and performance counters to determine the LLC architecture, in: International Conference on Computational Science, 2013, pp. 2557–2560.
[23] P. Vidal-Carreras, E. Guijarro, C. Santandreu-Mascarell, L. Canos-Daros, Analysis of the distribution of transversal skills at the universitat politècnica de valència, in: ICERI2017 Proceedings, 10th annual International Conference of Education, Research and Innovation, IATED, 2017, pp. 1365–1374. http://dx.doi.org/10.21125/iceri.2017.0445.

Josué Feliu received his M.Sc. and Ph.D. degrees in computer engineering from the Universitat Politècnica de València, Spain, in 2012 and 2017, respectively. He is currently working as a postdoctoral researcher at the Department of Computer Engineering of the same university. His research interests include scheduling strategies and performance modeling for multicore and multithreaded processors.

Julio Sahuquillo received his M.Sc., and Ph.D. degrees in computer engineering from the Universidad Politècnica de Valencia (Spain). Currently, he is a full professor at the Department of Computer Engineering. He has published more than 100 refereed conference and journal papers. His current research topics include multi- and manycore processors, memory hierarchy design, cache coherence, and power dissipation. He has co-chaired several workshops related with these topics, collocated in conjunction with IEEE supported conferences.

Salvador Petit received his Ph.D. degree in computer engineering from the Universitat Politècnica de València, Spain. Currently, he is an associate professor in the Department of Computer Engineering at UPV where he teaches several courses on computer organization. His research topics include multithreaded and multicore processors, memory hierarchy design, as well as real-time systems.