A Research-Oriented Course on Advanced Multicore Architecture: Contents and Active Learning Methodologies

Salvador Petit, Julio Sahuquillo, María E. Gómez, Vicent Selfa

Department of Computer Engineering (DISCA)

Universitat Politècnica de València

Cmno. de Vera s/n, 46022, SPAIN

spetit@disca.upv.es

Abstract

The fast evolution of multicore processors makes it difficult for professors to offer computer architecture courses with updated contents. To deal with this shortcoming that could discourage students, the most appropriate solution is a research-oriented course based on current microprocessor industry trends. Additionally, we also seek to improve the students' skills by applying active learning methodologies, where teachers act as guiders and resource providers while students take the responsibility for their learning. In this paper, we present the Advanced Multicore Architecture (AMA) course, which follows a research-oriented approach to introduce students in architectural breakthroughs and uses active learning methodologies to enable students to develop practical research skills such as critical analysis of research papers or communication abilities. To this end five main activities are used: i) lectures dealing with key theoretical concepts, ii) paper review & discussion, iii) research-oriented practical exercises, iv) lab sessions with a state-of-the-art multicore simulator, and v) paper presentation. An important part of all these activities is driven by active learning methodologies. Special emphasis is put on the practical side by allocating 40% of the time to labs and exercises. This work also includes an assessment study that analyzes both the course contents and the used methodology (both of them compared to other courses).

Keywords: Advanced computer architecture courses, teaching methods,

☆Fully documented templates are available in the elsarticle package on CTAN.

research-oriented method, lab sessions

2010 MSC: 00-01, 99-00

1. Introduction

Many universities all over the world organize Computer Architecture topics at least in two courses: an introductory course and an advanced course. Usually, several advanced courses covering different computer architecture topics (e.g. parallel computer architectures or memory subsystems) are offered. From our point of view, designing courses with updated contents is a key issue to capture the student's interest. However, the vertiginous technological and architectural advances in these topics impose a serious limitation for instructors to offer courses addressing up-to-date topics. Because of this reason, many instructors opt to follow an advanced computer architecture book for their courses. Despite this fact, some interesting courses are already being offered following a *research-oriented* perspective. In general, instructors of these courses present a solid background in research like Professor Onur Mutlu at Carnegie Mellon or Professor Christos Kozyrakis at Stanford. Examples of recent courses offered by Professor Mutlu are *18–742 Research in Parallel Computer Architecture* (available at `http://www.ece.cmu.edu/~ece742/f14/doku.php`) and *18–740 Computer Architecture* (available at `http://www.ece.cmu.edu/~ece740/f13/doku.php`). These courses cover a wide spectrum of computer architecture topics, mainly hot topics of top top-ranked conferences like ISCA or MICRO. This approach, based on recent hot topics is useful to capture the students' interest. Our approach, complements this one and, apart from stimulate the students' motivation, we also pursue to improve the students' skills by applying active learning methodologies.

Active learning methodologies have been recently applied on engineering courses [1, 2]. Under these methodologies, the student takes responsibility in his learning process while instructors act as resource providers and directors who guide the learning process. These methodologies have been shown to improve the understanding of engineering principles with respect to methodologies dominated by traditional lectures in which students are passive recipients. Additionally, these methodologies develop cross-curricular skills that students should acquire such as oral communication and critical analysis skills.

This work presents the undergraduate *Advanced Multicore Architectures* (AMA) course, offered in winter 2014, 2015 and 2016 at Universitat Universitat Politècnica de València. The course consists of 16 sessions of $2\frac{1}{2}$ hours each, which are taught over a period of four months.

The key difference of the proposed course over typical *advanced courses* is the use of active learning methodologies to teach hot research topics. For this purpose, the AMA course does not try to cover a wide range of architectural concepts (e.g. transactional memory or dataflow architectures are not studied). Instead, the course focuses on a *few key aspects* of recent and future multicores to place the stress on the practical side. The AMA course is organized in four main modules. Three of them study three major components (cores, caches, and memories) of a typical multicore, paying attention to both architectural and performance aspects; and one of them devoted to the study of performance methodologies recently proposed for multicores. Instructors highlight the hot research topics for each studied component, that is, where is the current academia and industry research interest. Additionally, students participate in the learning process by using active learning methodologies, which help them to develop skills required in the professional or research activity, such as writing and oral communication, apart from helping in the understanding of the studied concepts.

The main contribution of this paper is to provide guidelines to combine a research-oriented approach with active learning methodologies, which allows instructors to achieve an *advanced research-oriented* course. For this purpose, two main directions have been followed. First, the number of studied topics has been selectively reduced. Second, a wide set of activities based on active learning methodologies have been designed. Five main types of learning activities are performed: lectures, paper review & discussion, research-based exercises, lab sessions, and paper presentation.

In the AMA course, traditional lectures, where the instructors act as the only communicator, are used to explain some theoretical concepts, but in other lessons, the student plays an active role, by reviewing research papers and participating in their discussion in the classroom, while the instructor chairs and moderates the students' discussion. Research-based exercises refer to typical research problems that students will likely face after graduation. Lab sessions are performed with a state-of-the-art multicore simulator used at academia and the industry. For illustrative purposes, we present some examples for each type of activity discussing the roles of the student and the teacher, as well as some excerpt of the provided material. The time devoted by instructors to lectures, exercises, and labs is about 60%, 15%, and 25%, respectively; apart from the paper presentation sessions.

The remainder of this paper is organized as follows: Section 2 introduces the course main goals and methods employed to achieve them. Section 3 describes the course contents in detail. Section 4 explains in depth the methodology used to teach the course. Section 5 presents a survey which assesses the course contents and applied learning methods. Finally, Section 6 presents some concluding remarks.

2. AMA Course Learning Goals

This section summarizes the learning goals of the undergraduate AMA course. Before this course, students have attended to the *Computer Organization* course that studies the pipeline of a simple processor and the *Computer Architecture and Engineering* course, which describes extensions to the previous pipeline to support speculative execution. The focus in both courses is mainly educational and their aim is to introduce the basic concepts and to describe how the distinct computer components (e.g. superscalar processors, caches, etc.) work.

The AMA course covers the study of three main components of a typical multicore: the cores, the cache hierarchy and the main memory. It is aimed at providing students with the knowledge about industry and academia trends on multicores, as well as with the skills that enable them to initiate research in these topics. Due to time constraints, we have focused the course on industry-ready multicores with computing cores that implement conventional instruction set architectures (ISAs). For instance, graphic processing units (GPUs) and computational accelerators have not been included in the course.

The AMA course is organized in four main modules as shown in Table 1 apart from

Table 1: Learning Goals: concepts & skills (skills are shown with italic font). Legend. *Lect*: Lectures. *Prd*: Paper review & discussion. *Ex*: Research-oriented exercises. *Labs*: Lab sessions. *Pp*: Paper presentation.

Mod.	Learning Goals: concepts & skills	Method.
Mod. 0	Course Presentation	
	Course organization (contents, methodology and grading)	
	How to review papers	
Mod. 1	Core review & multicores	
	Sound knowledge of microarchitectural concepts	Lect
	Understand why multicores	Lect
	Get familiarized with multicore evolution	Lect
	Understand the benefits of heterogeneous multicores	Lect & Ex
	Acquire a sound understanding of how multicore simulators work	Labs
	Experiment Design and Data Analysis skills	Ex & Labs
	Acquire oral communication (discussion) and critical analysis skills	Prd
Mod. 2	Performance	
	Understand performance metrics for multicores	Lect
	Acquire skills analyzing performance metrics	Ex & Labs
	Confidence intervals and practical skills to obtain them	Lect & Ex
	Solid understanding about where performance can drop and why	Lect & Labs
	Understanding interferences in multicores	Lect & Labs
	Acquire a sound understanding of how multicore simulators work	Labs
	Acquire oral communication (discussion) and critical analysis skills	Prd
Mod. 3	Advanced caching	
	Shared cache partitioning for performance	Lect & Lab
	Dealing with fairness in shared caches	Lect & Lab
	Shared caches: scheduling and allocation	Lect
	Acquire a sound understanding of how multicore simulators work	Lab
	Acquire oral communication (presentation) skills	Pp
Mod. 4	Main memory	
	Understand how current DDR memory modules are organized	Lect & Ex
	Understand the current DRAM limitations and trends	Lect & Prd
	Acquire communication (discussion) and critical analysis skills	Prd
	Acquire oral communication (presentation) skills	Pp

Module 0, where instructors present the course contents, organization and grading, as well as introducing the guidelines followed in the course for reading research papers [3]. Module 2 is aimed at providing students with multicore evaluation fundamentals, while modules 1, 3 and 4 are devoted to the study of the main system components. An example of a recent multicore illustrating these components is presented in Figure 1, which depicts the layout of the IBM Power8 with twelve cores, on-chip L2 and L3 caches, and main memory controllers (located at the edges of the chip).

Each module pursues two main goals: i) review architectural and structural concepts in order to homogenize students' knowledge, ii) provide students with research

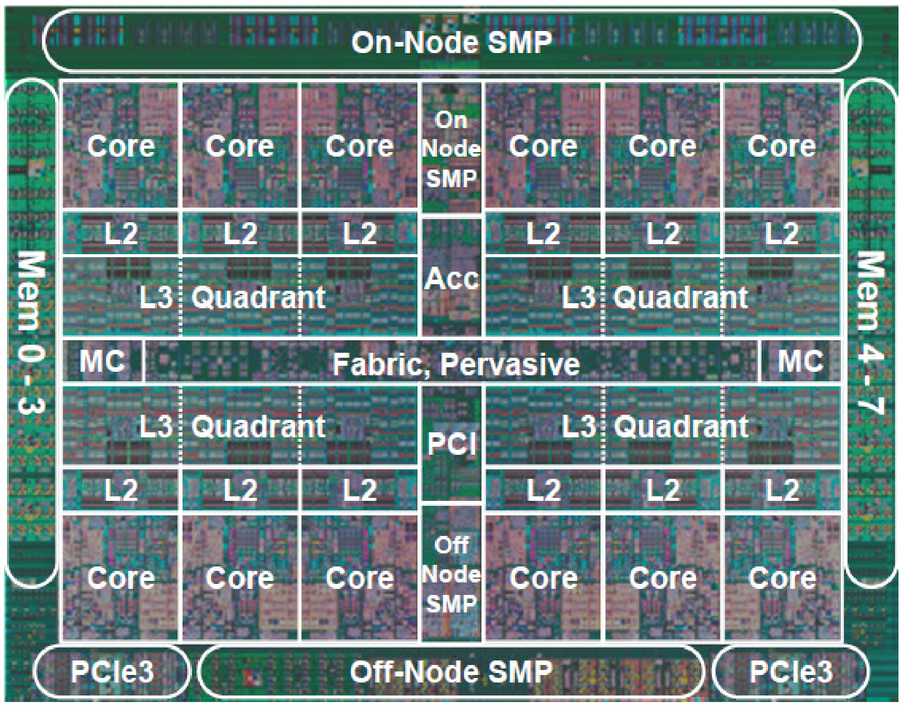

Figure 1: The Power IBM Power 8 die and major components. Source: IBM.

skills by means of research oriented exercises, labs, as well as paper review and discussion activities. The central column of the table summarizes the main learning goals and skills pursued across all the main modules (*skills are emphasized in italic characters*). A broad range of skills is pursued ranging from research oriented skills like simulator usage and development or data analysis, to interdisciplinary skills like paper review, critical analysis, paper presentation or paper discussion.

Finally, the rightmost column refers to the teaching methods employed to achieve the pursued learning goals.

3. AMA Course Contents

This section presents the contents of the AMA course. As mentioned above, the course is organized in four main modules that take a total of 16 sessions of $2\frac{1}{2}$ hours

Table 2: Course contents.

```
MODULE 0. Course Presentation
Topic 0.0. Course Description
           Grading policies
           How to do paper reviews

MODULE 1. Core review and multicores
Topic 1.1. Advanced Microarchitectural Concepts (review)
Topic 1.2. Reasons for moving to multicore
Topic 1.3. Multicore Evolution and Design

MODULE 2. Performance
Topic 2.1. Performance Evaluation Metrics
Topic 2.2. Performance Accounting Architectures

MODULE 3. Caching
Topic 3.1. Advanced Caching: Concepts and Problems
Topic 3.2. Advanced Caching: Papers

MODULE 4. Main Memory
Topic 4.1. Main Memory Organization
Topic 4.2. Main Memory Scheduling
```

each of them. The topics included in each module are presented in Table 2. Below, we describe each of these topics.

3.1. Module 1: Core review and multicores

3.1.1. Topic 1.1: Advanced microarchitectural concepts

In Topic 1.1, microarchitectural concepts are reviewed in order to homogenize the students' knowledge concerning core details. These working details are widely and deeply studied in another elective course called *Advanced Computer Architectures*. The focus of this module is to review and highlight microarchitectural details of typical commercial processors. The studied microarchitecture closely resembles to the Alpha 21264 [4] and most commercial microprocessors. The pipeline consists of a physical register file, a single instruction queue, the reorder buffer (ROB), and a load/store unit. The microarchitecture is reviewed detailing what is done at each stage. Emphasis is given to renaming, dispatching, and issue stages, paying special attention to why pipeline stalls can appear. In addition to this architecture, multithreaded processors are also studied, focusing on simultaneous multithreaded processors that are dominating an important segment of the market.

The key goal of this topic is twofold. On the one hand, to summarize the key characteristics of the distinct types of cores implemented in current multicores. On the other hand, to enable students to understand (in subsequent lectures) where performance can be lost during the program execution. At the end of this topic two papers [5, 6] are assigned to students to be discussed before introducing the next topic. Students must deliver at the beginning of the next lecture a brief (less than one page) review of the paper to the instructor.

3.1.2. Topic 1.2: Reasons for moving to multicore

Advances in transistor technology have allowed cramming more components onto integrated circuits as predicted by Moore's law [7]. This fact brings new opportunities for computer architects. In Topic 1.2, we discuss alternative architectures to multicores like *bigger* cores, larger caches, clustered processors, etc. Instructors present and discuss the pros and cons of each alternative to provide the students with a wide perspective on multicore design. Attention is paid to the analysis of the benefits each alternative provides.

Before starting this topic it is highly recommended that all the students read the paper *The Case for a Single-chip Multiprocessor* by Olukotun et al. [5]. This paper explains in detail the multiple reasons why industry moved to multicores (e.g. the power wall, wire delay, microarchitecture complexity, etc.) This reading is important since a widely extended misconception is that there is only one single cause of the industry trends. Based on our experience, the discussion of this paper in the classroom really encourages students to study multicore topics.

3.1.3. Topic 1.3: Multicore evolution and design

The last topic of Module 1 is devoted to multicore evolution and design. We present a representative set of commercial multicores, ranging from very simple in-order execution cores (e.g. the Piranha Chip Multiprocessor [8]) to complex multithreaded out-

Table 3: Multicore processors presented in the AMA course.

Multicore	#cores	Characteristics				#threads	Year
		Issue width	Threads/Core	Multithreading type	Issue type		
Piranha	8	1	1	–	in-order	8	2000
Niagara	8	2	4	fine-grained	in-order	32	2005
Niagara II	8	2	8	fine-grained	in-order	64	2007
IBM Power4	2	8	1	–	o-o-o	2	2002
IBM Power5	2	8	2	SMT	o-o-o	4	2004
IBM Power6	2	8	2	SMT	in-order	4	2007
IBM Power7	8	8	4	SMT	o-o-o	32	2010
IBM Power8	12	10	8	SMT	o-o-o	96	2014

of-order (e.g. IBM Power8) cores. The discussion on these multicores is always done emphasizing the design objectives and use case of each machine. For instance, if the goal is to support the execution of many threads in specific workloads (e.g. web workloads) a good design choice might be to implement many but simpler cores.

Table 3 summarizes the main characteristics of the studied multicores. We select a representative subset to illustrate the industry trends. During the earliest phases of the multicore evolution (2000-2007), some high-performance systems were built using processors with simple cores, which allowed supporting a high number of threads relatively early (e.g. the Niagara II supported 64 threads by 2007). During the same period, companies such as IBM or Intel introduce multicore designs with fewer complex cores (e.g. in 2007, the IBM Power6 only included 2 cores). Later, the number of cores and supported threads scaled with technology advances, reaching by one hundred of threads in 2014 (IBM Power8).

The second part of this topic focuses on the Amdahl's Law for multicores. This part is entirely based on the talk by Mark Hill entitled *Amdahl's Law in the Multicore Era* [9]. We use the Amdahl's Law to analyze both asymmetric and symmetric multicores.

3.2. Module 2: Performance

Both the industry and the academia have sharply moved from single core processors to multicores. The nature of multicores, different from their single core counterparts, has lead researchers to define specific performance metrics to evaluate multicore performance.

Several recent works [10, 11, 12] survey multicore performance metrics that have been used in the literature. These works also analyze and discuss which metrics should be used in order to get a more complete performance evaluation study. Due to these reasons, the mentioned papers are discussed in this module.

3.2.1. Topic 2.1: Performance evaluation metrics

In Topic 2.1, we discuss the key performance engineering steps: measurement, analysis, and improvements. Regarding measurement, this module covers both monitoring/profiling tools, as well as simulation tools. Special attention is paid to multicore metrics mainly based on the discussion presented in [10].

An important set of current research is being done on real machines (e.g. thread scheduling policies). In this regard, an interesting reading can be the work by Feliu et al. [13] where performance counters are used to assist a thread-to-core allocation policy on the Intel Xeon. We also present distinct profiling tools related to performance counters (e.g. Perf, PAPI, Libpfm, etc.).

Finally, practical stats for architects are studied. We present the basic principles and how to use stats in real systems to interpret the results. We study confidence intervals as a statistical tool that is useful to analyze the values of a given performance metric when they are not deterministic, which is the case of measurements performed on real systems.

3.2.2. Topic 2.2: Performance accounting architectures

Accounting architectures [14, 15, 16] allow researchers to achieve a sound understanding about where performance can be lost. Thus, we strongly recommend to include the study of these architectures in advanced architecture courses. We start Topic

2.2 with the concept of CPI stacks [14] for single-threaded processors. These stacks represent the contribution of the major processor components to the system's performance. After that, different approaches to construct CPI (cycles per instruction) stacks are analyzed, mainly focusing on that of the IBM Power5 and on the interval analysis approach. Interval analysis is studied in detail; the performance penalty is analyzed for both frontend miss events (e.g. I-Cache misses) and backend miss events (e.g. L2 data cache). The implementation of the accounting architecture is also discussed in detail in order to enable students to implement this architecture in a detailed multicore simulator.

After the study of accounting architectures in single core processors, we proceed with Topic 2.2 by explaining the accounting architecture for multicores [15]. The first step in this study is to understand the sources of interferences, which depend on the shared resources. The base system presents two main shared resources, a shared L2 cache which acts as the LLC (last level cache) and the main memory resources.Two types of interference at the LLC are studied and estimated, inter-thread cache misses and intra-thread cache misses. Interferences at the main memory are estimated assuming an open page policy and FR-FCFS (first ready, first come first served) scheduling policy. Students are provided with the equations to calculate all (inter- and intra-thread) interferences at run time.

These architectures allow estimating the execution time that each benchmark would experience in isolated execution. Therefore, they are of paramount importance to estimate the individual progress of each benchmark, which can be used as a powerful tool to investigate on fairness-aware policies for shared resources.

Finally, Topic 2.2 could be extended by applying interval analysis to processors including other type of cores like simultaneous multithreading (SMT) [16] cores or graphic processing units (GPUs). Nevertheless, these studies are relatively more complex so we leave them as optional readings for those interested students.

3.3. Module 3: Caching

Advanced cache design is of paramount importance for multicore performance due two main reasons. First, the miss latency introduces a serious performance penalty when the accessed data is retrieved from the off-chip main memory. Second, shared caches can become contention points that increase the average memory access time. Solutions to both problems require advanced techniques beyond classic cache performance enhancements. Module 3 deals with the most successful techniques proposed in the literature, as discussed below.

3.3.1. Topic 3.1: Advanced caching: concepts and problems

In Topic 3.1, basic concepts related to cache performance such as working set, associativity and miss ratio are revised. Special emphasis is given to the fact that simply reducing the miss ratio may not improve the performance, since miss latency depends on where the block is located and latency-hiding mechanisms must be taken into account.

After introducing basic caching concepts, several techniques to reduce miss rates are overviewed. These techniques go beyond increasing associativity and cache size, since blindly doing that will significantly increase access latency while only providing incremental benefits on the hit ratio. Instead, some successful proposals are presented, such as victim caches [17] or skewed associative caches [18]. The goal of these proposals is to reduce conflict misses without significantly impacting the access time.

Next, the topic deals with cache enhancements to reduce miss latencies. Basic approaches, such multi-level cache hierarchies, critical word first, or subblocking are reviewed, but special attention is paid to techniques aware of memory level parallelism (MLP). In this regard, non-blocking caches are used to allow multiple outstanding miss requests. First, the implementation of non-blocking caches [19] is explained in detail as well as the role of the miss status handling registers (MSHRs). Then, to demonstrate the importance of MLP-aware microarchitectural techniques, an example is presented where the optimal (regarding miss ratio) Belady's replacement algorithm [20] obtains lower performance than a basic MLP-aware replacement policy.

The last part of Topic 3.1 studies the multicore memory hierarchy as a shared resource. This part analyzes benefits and disadvantages of cache sharing. Disadvantages are mainly caused by uncontrolled sharing that can cause *unfairness* and even starvation of individual threads. This is an ongoing research area that links with the next topic.

3.3.2. Topic 3.2: Advanced caching: papers

Topic 3.2 studies recent papers dealing with caching problems already introduced in Topic 3.1. In particular, we focus on cache partitioning and insertion/replacement policies.

Regarding cache partitioning, the *Utility-based partitioning* approach [21] is discussed. This scheme partitions a shared cache among multiple applications depending on the reduction in the number of cache misses that each application is likely to experience. We selected this approach because it includes the *auxiliary tag directory*, a useful mechanism that has been used in other papers since it helps estimate the cache behavior is stand-alone execution.

With respect to insertion policies, the work by Seshadri *et al.* [22], which presents the *evicted-address filter*, is studied. This approach implements a hardware structure that holds the address of the most recently replaced blocks. This proposal decides the position of the LRU (least recently used) queue in which the block should be inserted to mitigate cache pollution and trashing. We selected this approach to illustrate insertion policies other than placing the incoming block at the top of the queue. With the same aim, the work [23] by Qureshi *et al.* that proposes a MLP-aware cache replacement policy is also studied.

3.4. Module 4: Main memory

The last module covers main memory issues in modern multicores. This module focuses on two main system components: the DRAM (dynamic random-access memory) organization and the memory controller. We start the module describing the main memory subsystem as a set of off-chip DRAM modules connected to one or more on-chip memory controllers. Then, we describe the major concerns affecting main memory design: i) capacity, bandwidth and quality of service (QoS) requirements; ii) energy consumption and iii) DRAM technology scaling.

3.4.1. Topic 4.1: Main memory organization

In the first topic, the DRAM organization is deeply reviewed using a bottom-up approach, starting from the DRAM memory cell. Once the basic cell is introduced, cell arrays and banks are straightforward presented. The concept of bank is introduced as a mean to reduce the access time and to increase memory level parallelism. This abstract concept then is placed in context by explaining how DRAM memory banks expand across several chips with a narrower data path in order to reduce the manufacturing costs of DRAM memory chips, and how they work jointly and synchronously to compound the wider data path of the banks. The internal organization of a memory chip is deeply analyzed with the students explaining the *row buffer* concept and how it acts as a basic prefetcher. Once the bank and chip structures have been studied, instructors introduce the basic DRAM commands that the memory controller issues to control DRAM memory access.

After the study of the chip organization, instructors define the concept of rank as a set of chips with their respective banks working in lockstep. Then, dual in-line memory module (DIMMs) are described as a set of ranks and memory channels are introduced. Finally, instructors present different DRAM address mapping schemes varying the physical address bits used to select the distinct components (banks, ranks, and channels) of the multidimensional DRAM organization. This is an interesting topic to discuss since the optimal mapping scheme depends on the memory access patterns of the executed workload.

3.4.2. Topic 4.2: Main memory scheduling

Finally, Module 4 covers the memory controller and memory request scheduling topics. Instructors first explain how refresh is done in current DRAM memories and its implications in performance and energy consumption nowadays and in the near future.

We then devote some time to the memory controller, describing all its functions, alternative locations (on-chip versus off-chip) and its components. Special attention is paid to memory request queues and scheduling policies. Two main policies are introduced and compared: FCFS and FR-FCFS. Finally, instructors review the two main ways of operation in current DRAM modules: open page and closed page, analyzing how they handle the row buffers, as well as their implications on performance and energy consumption.

4. AMA Learning Methodology

This section describes the proposed learning methodology to achieve the learning goals presented in Section 2, covering the skills and theoretical concepts of the course. For this purpose, the proposed course applies five main teaching methods: i) research-oriented lectures, ii) paper review & discussion, iii) practical exercises, iv) realistic lab sessions, and v) paper presentation. Below, each of them is discussed.

4.1. Lectures

Lectures are used with different purposes. On the one hand, the professor reviews and presents the basic theoretical concepts required to enable students to follow all the remaining activities planned for the course. On the other hand, lectures also serve to motivate students. For this purpose, the professor introduces current research and industry trends. In addition, the professor presents himself some research papers; either focusing on hot topics or fundamental papers that have had an important contribution in the past (e.g., [5]).

4.2. Paper review and discussion

In the first class of the course (Module 0), the professor states the importance of reviewing papers and introduces the guidelines to review papers. The professor explains that reviewing papers consists not only on understanding the proposed approach, but on analyzing the manuscript from a critical perspective, identifying the weaknesses and strengths. To provide feedback in the reviewing skills, the papers are discussed in the classroom.

This teaching method is implemented as follows. First, instructors assign one or two research papers to the students, that must be reviewed as homework. Students must deliver a short review report (around half to one page) to the instructor at the beginning of the paper discussion session, and before starting the core lecture. Basically, the report contains four main points about the paper proposal: summary, strengths, weakness, and ideas about how to improve the approach.

Then, the professor chairs a short (10 to 20 minutes) session, where the key aspects –pros and cons– of the studied paper are discussed. Usually, these students' discussions are co-scheduled with lectures related with the topics of the paper. Overall, each student reviews around four papers during the course. All the students review the same papers because it facilitates the students' evaluation and makes students more proactive in the discussion.

The *paper review and discussion* teaching method allow the instructors to work with active learning methodologies and improve skills such as oral communication and critical analysis.

4.3. Exercises

Unlike typical *paper and pencil* exercises, the proposed exercises are designed to train students to deal with common research problems. Most of the exercises are individually performed using a spreadsheet like *excel or libre office*. This can be done thanks to the classroom is equipped with enough desktop computers. The time taken to solve a typical exercise is relatively low (e.g. from half an hour to one hour). Because of this short time, exercises are intermingled with lectures. This way allows students to reinforce theoretical concepts. Below three exercises are discussed for illustrative purposes.

4.3.1. Exercise example 1. Amdahl's Law for multicores

This exercise focuses on the work *Amdahl's Law in the Multicore Era* [24] by Mark Hill. To perform the exercise, the instructor first explains basic concepts on symmetric and asymmetric multicores. The study focuses on a bounded chip area that can fit N simple cores (i.e., a symmetric multicore). Alternatively, this area can also fit an asymmetric multicore consisting of: i) an enhanced (big) core that occupies an area equivalent to R simple cores (base core equivalents or BCEs), and ii) $N - R$ simple cores. In the exercise, students must determine the speedup of an asymmetric multicore with respect to the baseline symmetric multicore while varying the size of the enhanced

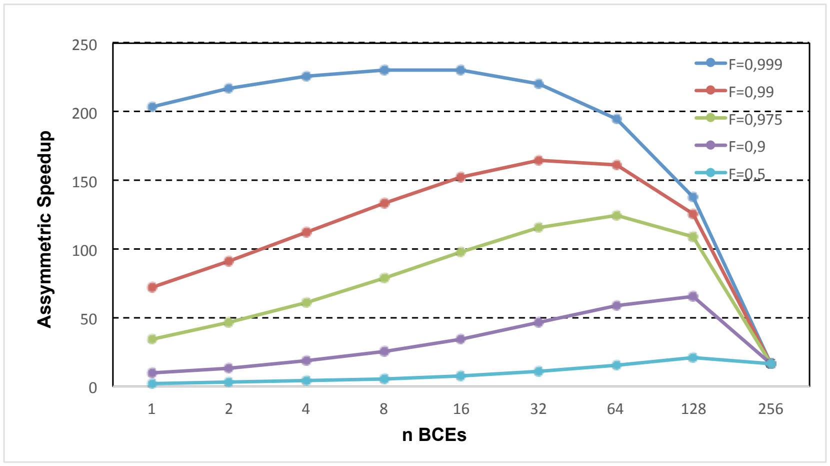

Figure 2: Resulting figure of the Amdahl law exercise for asymmetric multicores with a big core (area equivalent to n cores –X axis–) and $256 - n$ simple cores (BCEs).

core. Figure 2 presents the results for $N = 256$. Different curves are presented varying the parallel fraction F of the workload.

This exercise is really interesting and illustrates the need of heterogenous cores. In addition it is instructive since students fail to represent correctly the curve in their first two or three attempts corroborating Mark Hill's words: *"Everyone knows Amdahl's Law but quickly forgets it"*.

4.3.2. Exercise example 2: Metrics for measuring the performance of multiprogrammed workloads

Multicores typically run multiprogram workloads most of the time. These workloads consist of multiple independent applications, where each one runs on a specific core. The evaluation of such workloads is not straightforward mainly due to multicores implement shared resources. That is, applications interfere among them when accessing to such resources. Because of the interference, the execution time of a given application becomes unpredictable. Moreover, the multicore can favor the execution time of a given application at the cost of the others. To deal with this fact, recent work has claimed that there is a need to evaluate both performance and fairness in these systems. In this sense, new metrics have been recently proposed to deal with performance and fairness.

In this exercise students are provided with performance values (e.g. instructions per cycle in isolation and in multicore execution) of different applications across several machines. Students must use the provided values in a spreadsheet, where they must implement the equations and obtain different global and per application performance metrics. Once they have the results, students must argue which is the best machine considering the tradeoff between performance and fairness.

4.3.3. Exercise example 3. Energy consumption estimation in caches

A main design concern in current microprocessors is static and dynamic energy consumption. In this exercise students are asked to estimate the energy consumption of a L1 data cache. Dynamic energy can be obtained as the energy consumed by a single cache access multiplied by the number of cache accesses, while static energy grows linearly with the execution time.

To estimate both types of energy, the students combine the results of two tools: Multi2Sim [25] and CACTI [26]. Multi2Sim is used to gather the execution time and number of cache accesses while CACTI provides the dynamic energy consumed by a single access and the leakage (static energy) incurred by a given cache geometry and technology node.

Students must obtain the required inputs to compute the energy consumption with the aforementioned tools for a given set of benchmarks. To make the exercise length reasonable, we provide students the script to execute the benchmarks for a limited number of instructions. After that, they must deduce how to calculate, with the help of an spreadsheet, the energy consumption broken down in static and dynamic for a given processor clock. Figure 3 presents the resulting figure of this exercise.

4.4. Lab sessions

Lab sessions are designed pursuing four main goals: i) familiarize students with a detailed state-of-the-art multicore simulation framework, ii) enable students to accurately model multicores, iii) obtain results with representative workloads, iv) train stu-

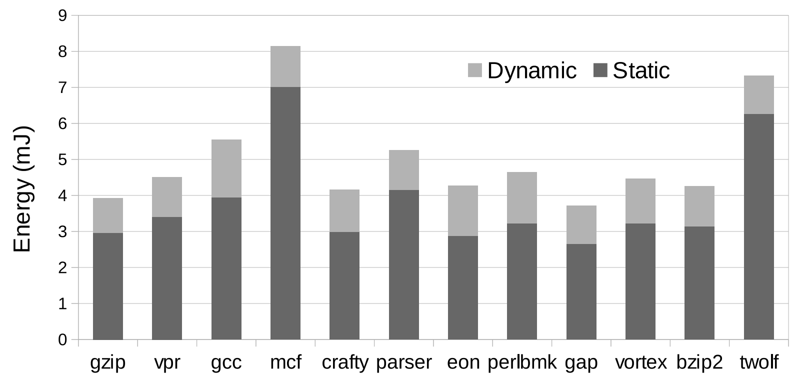

Figure 3: Resulting figure of the energy estimation exercise. Legend. *Dynamic*, *Static*: Dynamic and static energy consumption, respectively.

dents to analyze performance results and understand how multicore processors work. To achieve these goals, we choose Multi2Sim as simulation framework, used for research both in the academia and the industry.

To perform the experiments, we provide the students several scripts that run a short simulation with a given configuration. The obtained results must be plotted and analyzed with the help of a spreadsheet. As the exercises, the lab sessions are also carried out with the computers available at the classroom and take a $2\frac{1}{2}$-hour session. The main differences between both teaching methods lie on the time length and the complexity of the job to be done.

Students must prepare a report for each lab session where they must plot and discuss the obtained results. The report is revised by the instructor considering whether the student relates the discussed results with theoretical lectures, exercises or other lab sessions. Then, the revised report is used to provide feedback to the students. Below, three lab sessions are discussed for illustrative purposes.

4.4.1. Lab example 1. Single-threaded monocore processor

In this lab, students analyze the microarchitecture of a single-threaded processor. The purpose of this lab is to provide a solid understanding of the processor *stalls* that occur during the execution of applications. With this lab, students realize that most of

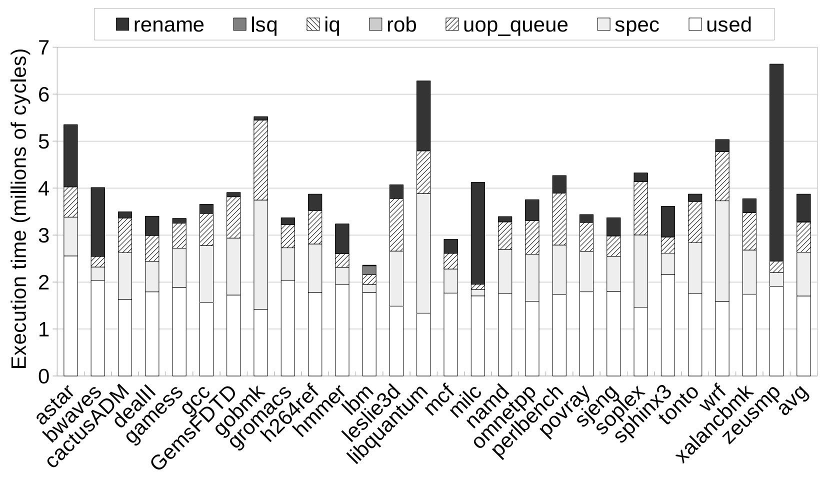

Figure 4: Resulting figure of laboratory 1. Legend. *rename*, *lsq*, *iq*, *rob*: stall time due to lack of free physical registers, entries in the load-store queue, entries in the instruction queue, entries in the reorder buffer, respectively. *uop_queue*: stall time due to empty fetch queue. *spec*: time lost dispatching misspeculated instructions. *used*: time used dispatching *useful* instructions.

these stalls are mainly related to long memory latency miss events.

Once students download and compile the Multi2Sim source code, the instructor explains the code to detect the multiple events that cause the dispatch to *stall*. During the lab, it is discussed if the default implementation takes into account all the possible causes or some of them are missing. Students are encouraged to back their claims with the information provided during regular lectures. After some discussion, the instructor indicates how to update the simulator before running the experiments for all the SPEC CPU 2006 benchmarks [27].

The lab explores the following dispatch stall causes: i) branch mispredictions, ii) lack of free physical registers, iii) lack of free entries in the load-store queue (LSQ), iv) lack of entries in the instruction queue (IQ), and v) lack of entries in the ROB. We start with a baseline configuration, where branch misprediction events is the main cause that *stalls* the processor. Figure 4 shows the resulting figure. It can be appreciated that the effects of branch mispredictions broken down in misspeculated instructions (*spec* label) and empty fetch queue (*uop_queue* label), take all together more execution time

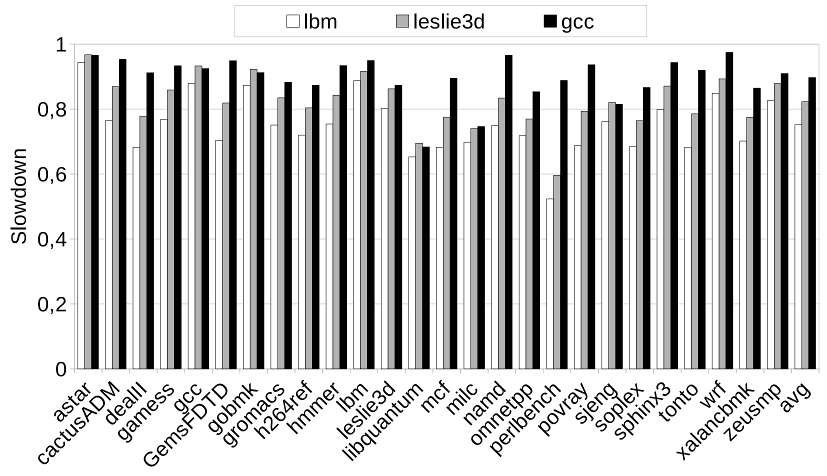

Figure 5: Resulting figure of laboratory 2. Legend. *lbm*, *leslie3d*, *gcc*: slowdown due to concurrent execution of 3 instances of *lbm*, *leslie3d*, or *gcc*, respectively.

in some benchmarks than *useful* instructions (*used*) do. As it can be observed, these stalls completely hide in the figure other possible stalls (i.e. those caused by a lack of entries in the LSQ, the IQ, and the ROB).

After this observation, students are asked to modify the baseline configuration to reduce the misprediction penalty. As a consequence, other stall causes or performance bottlenecks rise. The students should iteratively improve the baseline configuration step by step until performance stabilizes.

4.4.2. Lab example 2. Multicore evaluation with multiprogram workloads

In this laboratory, students explore the impact of multicore execution on the individual per-application performance.

First, students characterize the behavior of each application in stand-alone execution. They measure metrics such as the instructions per cycle (IPC), the L1 cache misses per kilo-instruction committed the ($MPKI_{L1}$) and the $MPKI_{L2}$.

Second, the study concentrates on analyzing the impact of the co-runners interference in multicore execution. Each application is executed with 1 and 3 instances of the same co-runner in a 2- or 4-core processor, respectively. Different co-runners are analyzed presenting different requirements of the L2 shared cache. Figure 5 depicts the resulting figure of the 4-core experiment where each bar shows the impact of a specific co-runner (*lbm*, *leslie3d*, and *gcc*). Students realize that the execution time of a given application is unpredictable since it can widely vary depending on the co-runner.

4.4.3. Lab example 3. Impact of cache partitioning on performance and fairness

This lab session examines different attempts to improve multicore fairness and performance by acting on the shared L2 cache. Three 8-way L2 cache schemes are evaluated: i) *default configuration* where no sharing policy is considered, ii) *static way partitioning* in which the 8 ways of each set are equally distributed among the 4 cores (2 ways per core), and iii) *static set partitioning* where the same amount of cache sets are assigned to each core.

To evaluate these schemes, students measure different performance and fairness indicators across multiple 4-application workloads. After that, students analyze the tradeoff between performance and fairness.

4.5. Paper presentation

At the end of the course, each student is asked to present a conference paper in the classroom. The aim of this activity is to develop the oral communication skills by doing a *real presentation* similarly as done in a conference. For this purpose, the instructor provides a list of papers belonging to any of the studied topics, e.g. advanced caching (module 3) or main memory organization (module 4). All the papers in the list have been presented in a recent conference. A key point of all these papers is that the slides used by the presenting author in the conference are available on the internet. This way eases the student's job, which mainly consists in studying the paper and its associated slides. Of course, students can modify and adapt available slides.

5. Course Assessment

The success of the course relies on two main pillars, the studied contents and the employed learning methodologies. To evaluate and provide feedback about these pil-

Table 4: Course contents assessment.

Statement	Totally disagree	Disagree	Average	Agree	Totally Agree
1. The theoretical contents cover in a wide extend recent processors and state-of-the-art research	0%	0%	0	14.3%	85.7%
2. The course presents current problems in current processors and analyzes the solutions to overcome them	0%	0%	0	57.1%	42.9%
3. Studying existing mechanisms to measure multicore performance helps me to understand better how the system works and where performance is lost	0%	0%	0 %	42.9%	57.1
4. Studying updated contents motivates me to the study of the course topics	0%	0%	14.2%	42.9%	42.9

lars, students were asked to complete two surveys, consisting of a set of statements, where students mark their level of satisfaction ranging from totally disagree to totally agree.

The results of the survey of the *Course Content Assessment* are presented in Table 4. The marks of this questionnaire highlight the importance of studying existing mechanisms and recent processors. Results show that students `agree` and `totally agree` that these aspects help them to achieve a better understanding about how the real systems work. Regarding motivation (fourth question), most of them (by 86%) state that these updated contents motivate them to study the course topics. Thus, the last question shows a clear sign of the success of the studied contents because increasing students' motivation was one of the aim that we chased with this new course organization.

The survey of the *Course Methodology Assessment* is presented in Table 5. As observed in the first three questions, students have had a good acceptance about paper reading, discussion and paper presentation methods. Most of the students consider that using active teaching methods such as exercises and lab sessions, in which students play an active role, helps them to better understand theoretical concepts. This is demonstrated by the results of statements 4 and 5, which are positively scored by more than 85% of students and did not receive any negative score. Regarding labs, notice that students agree in the importance of writing lab reports after laboratories. An interesting observation is that students perceive that the teaching methodologies contribute to students' cross-curricula skills (e.g. communication, analysis and writing skills). In

Table 5: Course methodology assessment.

Statement	Totally disagree	Disagree	Average	Agree	Totally Agree
1. Reading papers develops the critical analysis that helps identifying the most important issues of each proposal	0%	0%	14.3%	28.6	57.1%
2. Paper discussion at class help me to develop communication skills (public speaking and ideas discussion)	0%	0%	14.2%	42.9%	42.9%
3. Presenting a top conference paper in class helps me to improve communication skills	0%	0%	0 %	42.9%	57.1
4. Research-oriented exercises help me to understand theoretical aspects	0%	0%	14.2%	42.9%	42.9%
5. Lab sessions help me to better understand the processor	0%	0%	14.3%	57.1%	28.6
6. Writing the *lab report* after the Laboratory helps me to improve analysis and writing skills	0%	0%	0	57.1%	42.9%
7. In general, I find the followed methodology (exercises, paper reading, Lab sessions) very complete in comparison with other courses	0%	0%	0	57.1%	42.9%

this context, the work related with papers' review has contributed specially to develop these skills. Special attention must be paid to the last statement where all the students found the methodology very complete compared to other existing courses.

In short, the results of the surveys show that the AMA methodology motivates the study and helps them to achieve a better understanding of the topics addressed in the course. Moreover, the second survey shows that the teaching methods develop students' analysis and communication abilities, which are fundamental for the professional activity.

6. Conclusions, lessons learned and future directions

This paper has presented the contents of the course *Advanced Multicore Architectures* offered at Universitat Politècnica de València. The course is organized in four modules, three of them devoted to the study of the three main components of a current multicore (core, caches, and main memory) and the other tackling multicore performance evaluation. The course has been designed to motivate students on the study of advanced computer architecture topics and to enable them to research on these topics.

For this purpose, the course includes cutting-edge contents at lectures, highlighting current research trends on the academia and the industry, and makes use of active learning methodologies.

In addition, this paper presents an overview of the main teaching methods where the course relies on in order to fulfill its objectives. These teaching methods are lectures, paper reviews & discussion, exercises, labs, and paper presentation; all of them with the aim of providing students the skills to enable them to research on the studied computer architecture topics.

We would like to remark the main lessons learned that could help other colleagues to adapt the proposed approach to other contexts:

- A key issue is the selection of an appropriate set of papers for revision and discussion. It is important that this set contains both classic papers (e.g. germinal/visionary papers, widely referenced, that have had a great impact both in the academia and the industry) and recent papers, showing the current research/industry trends. In our opinion, both kind of papers motivate the students.
- Another issue is if the course focus should be either wide (more topics with few details) or deep (less topics but more details and lab training). Our experience tells us that the first approach, in general, demotivates most of the students. In contrast, the second option allows students to master some topics. Thus, we feel that the latter approach should be followed. The mastery of a subject makes the student gain confidence and motivation to go on.
- It is highly recommended that both presentation of papers and laboratories are related to taught lectures and reviewed documents. Working on the same direction helps students get more performance and benefits of their work.

As for future AMA course directions, we plan to include contents related to many-core and GPU processors due to their growing importance in the high-performance computing domain. In this regard, an interesting processor is the Xeon Phi "Knights Landing", an x86-compatible many-core recently launched by Intel that includes 72 computing cores.

Acknowledgments

This work was supported in part by the Spanish Ministerio de Economía y Competitividad (MINECO) and by Plan E funds under Grant TIN2014–62246–EXP and Grant TIN2015–66972–C5–1–R, and by Generalitat Valenciana under grant AICO/2016/059. Authors also would like to thank Onur Mutlu for making available online his valuable teaching material.

References

[1] I. Estévez-Ayres, C. Alario-Hoyos, M. Pérez-Sanagustín, A. Pardo, R. M. Crespo-García, D. Leony, H. A. P. G., 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, International Journal of Technology and Design Education 25 (3) (2014) 387–408.

[2] O. Arbelaitz, J. I. Martín, J. Muguerza, Analysis of introducing active learning methodologies in a basic computer architecture course, IEEE Transactions on Education 58 (2).

[3] A. J. Smith, The task of the referee, Computer 23 (4) (1990) 65–71. `doi:10.1109/2.55470`.

[4] R. E. Kessler, The alpha 21264 microprocessor, IEEE Micro 19 (2) (1999) 24–36. `doi:10.1109/40.755465`.

[5] K. Olukotun, B. A. Nayfeh, L. Hammond, K. Wilson, K. Chang, The case for a single-chip multiprocessor, in: ASPLOS, 1996, pp. 2–11. `doi:10.1145/237090.237140`.

[6] S. Palacharla, N. P. Jouppi, J. E. Smith, Complexity-effective superscalar processors, in: ISCA, 1997, pp. 206–218. `doi:10.1145/264107.264201`.

[7] R. R. Schaller, Moore's law: Past, present, and future, IEEE Spectr. 34 (6) (1997) 52–59. `doi:10.1109/6.591665`.

[8] L. A. Barroso, K. Gharachorloo, R. McNamara, A. Nowatzyk, S. Qadeer, B. Sano, S. Smith, R. Stets, B. Verghese, Piranha: A scalable architecture based on single-chip multiprocessing, in: ISCA, 2000, pp. 282–293. doi:10.1145/339647.339696.

[9] https://www.youtube.com/watch?v=KfgWmQpzD74.

[10] V. Selfa, J. Sahuquillo, C. Gómez, M. E. Gómez, Methodologies and performance metrics to evaluate multiprogram workloads, in: PDP, 2015.

[11] S. Eyerman, L. Eeckhout, Restating the case for weighted-ipc metrics to evaluate multiprogram workload performance, IEEE Comput. Archit. Lett. 99 (2013) 1. doi:http://doi.ieeecomputersociety.org/10.1109/L-CA.2013.9.

[12] P. Michaud, Demystifying multicore throughput metrics, IEEE Comput. Archit. Lett. 12 (2) (2013) 63–66. doi:http://doi.ieeecomputersociety.org/10.1109/L-CA.2012.25.

[13] J. Feliu, J. Sahuquillo, S. Petit, J. Duato, L1-bandwidth aware thread allocation in multicore SMT processors, in: PACT, 2013, pp. 123–132. doi:10.1109/PACT.2013.6618810.

[14] S. Eyerman, L. Eeckhout, T. Karkhanis, J. E. Smith, A performance counter architecture for computing accurate cpi components, in: ASPLOS, 2006, pp. 175–184. doi:10.1145/1168857.1168880.

[15] K. Du Bois, S. Eyerman, L. Eeckhout, Per-thread cycle accounting in multicore processors, ACM Trans. Archit. Code Optim. 9 (4) (2013) 29:1–29:22. doi:10.1145/2400682.2400688.

[16] S. Eyerman, L. Eeckhout, Per-thread cycle accounting in smt processors, in: ASPLOS, 2009, pp. 133–144. doi:10.1145/1508244.1508260.

[17] N. P. Jouppi, Improving direct-mapped cache performance by the addition of a small fully-associative cache and prefetch buffers, in: ISCA, 1990, pp. 364–373. doi:10.1145/325164.325162.

[18] A. Seznec, A case for two-way skewed-associative caches, in: ISCA, 1993, pp. 169–178. doi:10.1145/165123.165152.

[19] D. Kroft, Lockup-free instruction fetch/prefetch cache organization, in: ISCA, 1981, pp. 81–87.

[20] L. A. Belady, A study of replacement algorithms for a virtual-storage computer, IBM Syst. J. 5 (2) (1966) 78–101. doi:10.1147/sj.52.0078.

[21] M. K. Qureshi, Y. N. Patt, Utility-based cache partitioning: A low-overhead, high-performance, runtime mechanism to partition shared caches, in: MICRO, 2006, pp. 423–432. doi:10.1109/MICRO.2006.49.

[22] V. Seshadri, O. Mutlu, M. A. Kozuch, T. C. Mowry, The evicted-address filter: A unified mechanism to address both cache pollution and thrashing, in: PACT, 2012, pp. 355–366. doi:10.1145/2370816.2370868.

[23] M. K. Qureshi, D. N. Lynch, O. Mutlu, Y. N. Patt, A case for mlp-aware cache replacement, in: ISCA, 2006, pp. 167–178. doi:10.1109/ISCA.2006.5.

[24] M. D. Hill, M. R. Marty, Amdahl's law in the multicore era, Computer 41 (7) (2008) 33–38.

[25] R. Ubal, J. Sahuquillo, S. Petit, P. López, Multi2sim: A simulation framework to evaluate multicore-multithread processors, in: SBAC-PAD, 2007, pp. 62–68.

[26] T. Thozhiyoor, N. Muralimanohar, J. Ahn, N. Jouppi, Cacti 5.1, Tech. rep., HP-2008-20, HP Labs (2008).

[27] C. D. Spradling, Spec cpu2006 benchmark tools, SIGARCH Comput. Archit. News 35 (1) (2007) 130–134.