A Research-Oriented Course on Advanced Multicore Architecture

J. Sahuquillo, S. Petit, V. Selfa, M.E. Gómez
Depto. de Informática de Sistemas y Computadores (DISCA)
Universitat Politècnica de València
Valencia (SPAIN)
{jsahuqui,spetit,viselol,megomez}@disca.upv.es

***Abstract*—Multicore processors have become ubiquitous in our real life in devices like smartphones, tablets, etc. In fact, they are present in almost all segments of the computing market, from supercomputers to embedded devices. The huge market competence have lead industry and academia to develop vertiginous technological and architectural advances.**

The fast evolution that are still experiencing current multicores makes difficult for instructors to offer computer architecture courses with updated contents, preferably showing the industry and academia research trends. To deal with this shortcoming, authors consider that a research-oriented course is the most appropriate solution.

This paper presents an advanced computer architecture course called *Advanced Multicore Architectures*, offered in 2015. The course covers the basic topics of multicore architecture and has been organized in four main modules regarding multicore basis, performance evaluation, advanced caching, and main memory organization.

The course follows a research-oriented approach that covers theoretical concepts at lectures in which recent research papers are analyzed to provide students a wide view of current trends. Moreover, additional teaching methods like lab sessions with a state-of-the-art multicore simulator or research-oriented exercises have been used with the aim of introducing students to research in these topics. To achieve this *fully research-oriented* methodology, about 40% of the time is devoted to labs and exercises.

***Index Terms*—Advanced computer architecture courses; teaching methods; research-oriented method; lab sessions.**

I. Introduction

Computer architecture topics are organized in many universities all over the world at least in two courses: an introductory course and an advanced course. Usually, there are several advanced courses covering different computer architecture topics (e.g. parallel computer architectures or memory subsystems).

The fast pace of technological and architectural advances in computer architecture suppose 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. From our point of view, offering courses with updated contents is a key issue to capture the student's interest. In this regard, some interesting courses are already being offered. 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* and *18–740 Computer Architecture*. These courses cover an wide spectrum of computer architecture topics, mainly hot topics of top top-ranked conferences like ISCA or MICRO. With the aim of offering updated contents Professor Mutlu is continuously changing the course's topics or offering new courses.

This work presents the contents of the undergraduate course called *Advanced Multicore Architectures* (AMA) offered in 2015 at the Universitat Politècnica de València. Multicores have experienced an evolution without precedent as demonstrated by the fact that multicores are present in the market with a wide variety of design choices: in-order versus out-of-order execution cores, many simple cores versus few but powerful cores, heterogeneous cores, etc.

The offered AMA course does not try to cover a wide range of architectural concepts, e.g. transactional memory or dataflow architectures are not covered. Instead, the course focuses on key aspects of multicores and is organized in four main modules. Three of them aimed at studying the working details of the major components (cores, caches, and memories) of a typical multicore processor, paying attention to both architectural and performance aspects; and one of them to study general performance methodologies used in multicore design. Instructors highlight for each studied component the hot research topics, that is, where is the current academia and industry research interest.

Regarding lectures, this course follows a research-oriented approach with the aim of providing updated contents and capture the interest of students. However, unlike the previous courses, the AMA course pursues a *fully research-oriented* approach. With this aim, instructors selectively reduce the number of the studied topics, which allows them to devote an important amount of time to train students to research. With this aim, teaching methods such as research-based exercises, lab sessions, and a course work have been also used. 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. The course work refers to a relatively complex implementation in the multicore simulator. The time devoted by instructors to lectures, exercises, and labs is about 60%, 15%, and 25%, respectively. In addition, extra time is

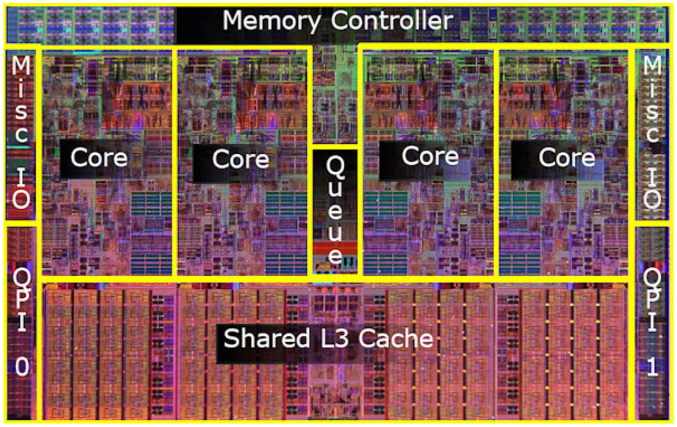

Fig. 1. Core i7 die and major components. Source: Intel.

required at home to perform the course work.

The remainder of this paper is organized as follows: Section II describes the course contents in detail. Section III explains the methodology used to taught the course. Finally, Section IV presents some concluding remarks.

II. Proposed Research-Oriented Advanced Multicore Architecture Course

This section describes the contents of the 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 contents of the AMA course are aimed to provide the students with the knowledge about how current multicores work as well as the industry and academia trends, giving the students a broader and more insightful view of modern computers. Moreover, we provide also the skills that allow students to initiate research in these topics. For this purpose, the proposed lectures are organized in four main modules. The course is organized in 16 lectures of $2\frac{1}{2}$ hours each of them.

The course studies the three main components of a typical multicore processor: the cores, cache structures, and main memory. Figure 1 illustrates the layout of the Intel Core i7 as example. The course is organized in four main modules as listed in Figure 2. Module 0 is the typical one where instructors present the course contents, organization, and grading. In addition, in this course instructors include a few slides to explain the key rules to review research papers [1].

```
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. Multicore Processors Why
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
```

Fig. 2. Course contents.

Module 1: Core review and multicores

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 Alpha21264 [2] and most commercial microprocessors. The pipeline consists of a physical register file, a single instruction queue, the 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 [3], [4] are assigned to students to be discussed before introducing he 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.

Topic 1.2: Multicore processors why

Advances in transistor technology have allowed cramming more components onto integrated circuits as predicted by Moore's law [5]. 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 payed 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. [3]. Based on our experience, the discussion of this paper at classroom really encourages students to the study of multicore topics.

Topic 1.3: Multicore evolution and design

The last topic of Module 1 is devoted to multicore evolution and design. We present a representative subset of commercial multicores, ranging from very simple in-order execution cores (e.g. the Piranha Chip Multiprocessor [6]) to complex multithreaded out-of-order (e.g. IBM Power 8) 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.

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* [7] that can be found on the Internet at https://www.youtube.com/watch?v=KfgWmQpzD74. We use the Amdahl's Law to analyze both asymmetric and symmetric multicores.

Module 2: Performance

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

In [8], Selfa *et al.* present a survey on multicore performance evaluation metrics that have been defined and used in recent top computer architecture conferences. Some interesting readings on this topic are the work by Eyerman and Eeckhout in [9] and the work by Michaud in [10].

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 payed to multicore metrics mainly based on the discussion presented in [8].

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. [11] 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

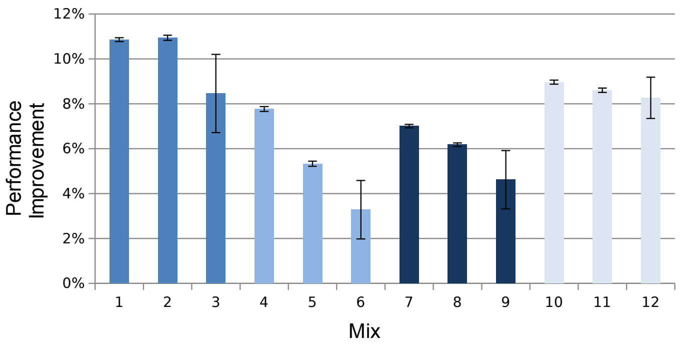

Fig. 3. Resulting figure from a confidence interval exercise.

the case of measurements performed on real systems. As an example, Figure 3 shows the confidence intervals that students obtain for distinct performance metric (e.g. IPC). For this purpose, we first provide the students with a wide set of values of that metric, which are gathered on a real machine. To plot each bar and it associated confidence interval, the results of 20 executions of the same mix (1–12) are used.

Topic 2.2: Performance accounting architectures

Accounting architectures [12]–[14] represent an important advance that allow researchers to achieve a sound understanding about where performance can be lost. Instructors strongly recommend their colleagues to include the study of these architectures on their courses. We start Topic 2.2 with the concept of CPI stacks [12] 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 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 and I-TLB misses) and backend miss events (e.g. L2 data cache). Implementation of the accounting architecture are 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 [13]. 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 which acts as the LLC (last level cache) and the main memory resources (memory controller, memory bus, and memory modules). Two types of interference at the LLC are studied and estimated, inter-thread cache misses and intra-thread cache misses. The former represents extra conflict misses due to threads evicting each other's data. The latter refers to misses that also occur in single core execution but they present longer latency in multicore execution. 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 formulas to calculate all (inter- and intra-thread) interferences at run time.

These architectures allow to estimate the execution time that each benchmark would have experienced in isolation. 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 SMT [14] cores or GPUs. Nevertheless, these studies are relatively more complex so we leave them as optional readings for those interested students.

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 off-chip 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. In Topic 3.1, advanced caching concepts are explained highlighting possible shortcomings, while in Topic 3.2, a group of selected papers tackling several of the studied shortcomings is reviewed.

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 [15] or skewed associative caches [16]. 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 payed 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 [17] 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 [18] 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 sharing the cache. Disadvantages are mainly caused by uncontrolled sharing that can produce unfairness and even starvation of individual threads. This fact difficults complying with QoS and real-time constraints. Static partitioning of resources is presented as a naive solution to solve these disadvantages; however, it lowers resource utilization. Therefore, the best solution should enable resource sharing but addressing QoS and fairness. This is an ongoing research area that links with the next topic.

Topic 3.2: Advanced caching: papers

Topic 3.2 presents several recent papers dealing with caching problems already introduced in Topic 3.1. In particular, a proposal regarding cache partitioning is explained, as well as two others addressing insertion and replacement policies.

Regarding cache partitioning, the *Utility-based partitioning* paper [19] by Qureshi and Patt is discussed. This scheme partitions a shared cache between multiple applications depending on the reduction in the number of cache misses that each application is likely to experience for a given partition. For this purpose, the proposal implements an auxiliary tag directory, a useful mechanism that has been used in some other papers, and that helps estimate the cache behavior is stand-alone execution.

With respect to insertion policies, the work by Seshadri *et al.* [20], which presents the evicted-address filter mechanism, is studied. This approach implements a hardware structure that holds the address of the most recently replaced blocks. This structure is used to check if a given block belongs to the actual working set of the workload. The result of this lookup decides in what position of the LRU queue the block should be inserted to avoid cache pollution and trashing.

Finally, the work [21] by Qureshi *et al.* that proposes an interesting dynamic MLP-Aware cache replacement approach is discussed. This paper claims that misses that occur in isolation are more costly on performance than those that occur in parallel, since the latency of the latter ones can be hidden. Based on this claim, authors classify misses depending on a cost metric that takes into account the number of parallel outstanding memory requests. Simple logic to compute this metric is implemented in the MSHRs. When the block is finally retrieved, its associated computed cost is stored in the cache to assist the replacement policy, with the aim of replacing those blocks whose miss cost is predicted to be higher. The proposal also explores a hybrid replacement policy that dynamically moves to the LRU algorithm when the estimation accuracy is not good enough.

Module 4: Main memory

The last module covers main memory issues in modern multicores. This module focuses on two main components of the system: the DRAM memory organization and the memory controller. We start the module describing the main memory

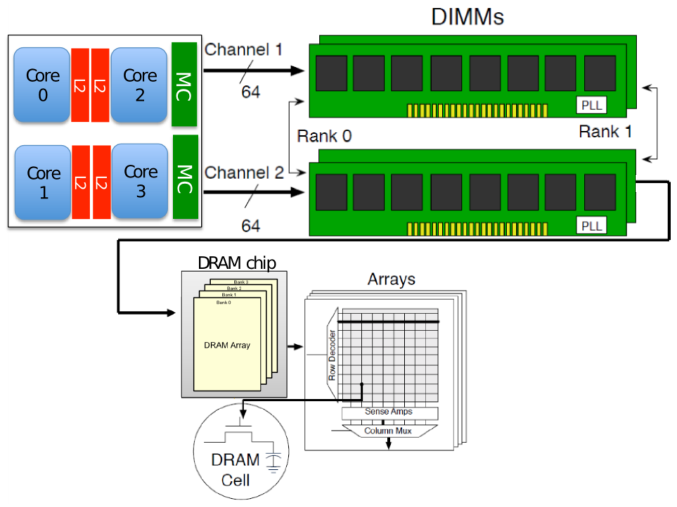

Fig. 4. Multidimensional organization of the DRAM memory.

subsystem as a set of off-chip DRAM memory modules connected to one or more on-chip memory controllers. Then we describe the major concerns affecting main memory: i) need for capacity, bandwidth and QoS requirements; ii) energy consumption; and iii) DRAM technology scaling. This helps students to know which are the main problems that threaten nowadays the performance of the main memory subsystem.

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. The concept of bank is presented 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 cost of the 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 students explaining the concept of row buffer 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, DIMMs are described as a set of ranks, and memory channels are introduced. An example of a hierarchical DRAM organization is depicted in Figure 4.

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 main memory access patterns of the executed worload.

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 (first come first served) and FR-FCFS (first ready, first come first served). 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.

III. Methodology

The offered course pursues three main goals: i) to make more attractive for students the study and research on computer architecture topics, ii) to provide the students a sound understanding of the studied topics, and iii) to enable the students in the research of the studied computer architecture topics.

The methodology devised to achieve these goals and to make the course successful combines four main teaching methods: research-oriented lectures, practical exercises, realistic lab sessions, and course work. Below the focus of each one is discussed.

A. *Lectures*

Lectures are used to review and present basic theoretical concepts to enable students to follow the discussion. After that, current research and industry trends are introduced by instructors, highlighting the hot topics. For this purpose and to motivate the students as well, research papers are discussed. Instructors select papers to be explained at classroom according to two main criteria: papers that have had a strong impact on the industry or research in the past (e.g. [3]), and papers addressing current hot topics.

B. *Exercises*

As mentioned above, exercises are designed to train students to deal with common research problems. Different types of exercises are proposed (e.g. estimating energy consumption –static and dynamic– in caches or obtaining confidence intervals for a given set of performance metrics). The time taken to solve each of these problems is relatively low (e.g. from half an hour to one hour), and most of them are solved with the help of an spreadsheet available at the computer assigned to each student in the classroom. Because of this short time, exercises are intermingled with lectures. This way allows the student to reinforce theoretical concepts.

C. *Lab sessions*

The main goal of lab sessions is to provide the students with the skills to work on a typical simulation framework used for research both in the academia and the industry.

In the lab sessions students are trained in the use of the Multi2Sim multicore simulation framework [22]. This tool is a state-of-the-art simulator used for research in the academia and the industry. Multi2Sim allows modeling the distinct system components (memory hierarchy, cores, and main memory) and study their impact on the multicore performance. Results obtained with the simulator are plotted and analyzed with the help of a spreadsheet. Just as with exercises, lab sessions are also carried out using the computers available at the classroom. The main differences between labs and exercises lie on the time needed to complete them and the used tools. A lab session requires relatively more work so it takes an entire 2.5-hour session.

D. Course work

Students must do a final course work when the course is approaching to its end. The course work consists on performing an implementation in the multicore simulator that is more complex than those done at lab sessions. Moreover, unlike lab sessions, the course work is not guided by instructors.

The main goal of this teaching method is that students demonstrate that they have acquired a certain degree of autonomy to work alone with a relatively complex simulation framework. This way will make students more self confident to work on research. For this purpose, the course work is carried out individually by each student at home.

As an example, one of the offered works is the extension of a lab session where students estimate hit ratio decrease in mixed workloads to implement an accounting architecture [13].

IV. Conclusions

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.

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

Acknowledgments

This work was supported by the Spanish Ministerio de Economía y Competitividad (MINECO) and by FEDER funds under Grant TIN2012–38341–C04–01, and by the Intel Early Career Faculty Honor Program Award. Authors also would like to thank Onur Mutlu for making available online his valuable teaching material.

References

[1] A. J. Smith, “The task of the referee,” *Computer*, vol. 23, no. 4, pp. 65–71, Apr. 1990.
[2] R. E. Kessler, “The alpha 21264 microprocessor,” *IEEE Micro*, vol. 19, no. 2, pp. 24–36, Mar. 1999.
[3] K. Olukotun, B. A. Nayfeh, L. Hammond, K. Wilson, and K. Chang, “The case for a single-chip multiprocessor,” in *ASPLOS*, 1996, pp. 2–11.
[4] S. Palacharla, N. P. Jouppi, and J. E. Smith, “Complexity-effective superscalar processors,” in *ISCA*, 1997, pp. 206–218.
[5] R. R. Schaller, “Moore’s law: Past, present, and future,” *IEEE Spectr.*, vol. 34, no. 6, pp. 52–59, Jun. 1997.
[6] L. A. Barroso, K. Gharachorloo, R. McNamara, A. Nowatzyk, S. Qadeer, B. Sano, S. Smith, R. Stets, and B. Verghese, “Piranha: A scalable architecture based on single-chip multiprocessing,” in *ISCA*, 2000, pp. 282–293.
[7] M. D. Hill and M. R. Marty, “Amdahl’s law in the multicore era,” *Computer*, vol. 41, no. 7, pp. 33–38, Jul. 2008.
[8] V. Selfa, J. Sahuquillo, C. Gómez, and M. E. Gómez, “Methodologies and performance metrics to evaluate multiprogram workloads,” in *PDP*, 2015.
[9] S. Eyerman and L. Eeckhout, “Restating the case for weighted-ipc metrics to evaluate multiprogram workload performance,” *IEEE Comput. Archit. Lett.*, vol. 99, p. 1, 2013.
[10] P. Michaud, “Demystifying multicore throughput metrics,” *IEEE Comput. Archit. Lett.*, vol. 12, no. 2, pp. 63–66, 2013.
[11] J. Feliu, J. Sahuquillo, S. Petit, and J. Duato, “L1-bandwidth aware thread allocation in multicore SMT processors,” in *PACT*, 2013, pp. 123–132.
[12] S. Eyerman, L. Eeckhout, T. Karkhanis, and J. E. Smith, “A performance counter architecture for computing accurate cpi components,” in *ASPLOS*, 2006, pp. 175–184.
[13] K. Du Bois, S. Eyerman, and L. Eeckhout, “Per-thread cycle accounting in multicore processors,” *ACM Trans. Archit. Code Optim.*, vol. 9, no. 4, pp. 29:1–29:22, Jan. 2013.
[14] S. Eyerman and L. Eeckhout, “Per-thread cycle accounting in smt processors,” in *ASPLOS*, 2009, pp. 133–144.
[15] 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.
[16] A. Seznec, “A case for two-way skewed-associative caches,” in *ISCA*, 1993, pp. 169–178.
[17] D. Kroft, “Lockup-free instruction fetch/prefetch cache organization,” in *ISCA*, 1981, pp. 81–87.
[18] L. A. Belady, “A study of replacement algorithms for a virtual-storage computer,” *IBM Syst. J.*, vol. 5, no. 2, pp. 78–101, Jun. 1966.
[19] M. K. Qureshi and Y. N. Patt, “Utility-based cache partitioning: A low-overhead, high-performance, runtime mechanism to partition shared caches,” in *MICRO*, 2006, pp. 423–432.
[20] V. Seshadri, O. Mutlu, M. A. Kozuch, and T. C. Mowry, “The evicted-address filter: A unified mechanism to address both cache pollution and thrashing,” in *PACT*, 2012, pp. 355–366.
[21] M. K. Qureshi, D. N. Lynch, O. Mutlu, and Y. N. Patt, “A case for mlp-aware cache replacement,” in *ISCA*, 2006, pp. 167–178.
[22] R. Ubal, J. Sahuquillo, S. Petit, and P. López, “Multi2sim: A simulation framework to evaluate multicore-multithread processors,” in *SBAC-PAD*, 2007, pp. 62–68.