Multicore Education Through Simulation

Ozcan Ozturk, *Member, IEEE*

Abstract—**A project-oriented course for advanced undergraduate and graduate students is described for simulating multiple processor cores. Simics, a free simulator for academia, was utilized to enable students to explore computer architecture, operating systems, and hardware/software cosimulation. Motivation for including this course in the curriculum is provided along with a detailed syllabus and an assessment demonstrating its successful impact on the students.**

Index Terms—**Architecture, chip, experiential learning, multicore, Simics, simulation.**

I. Introduction

As COMMONLY accepted, due to power limitations present in processor design, the current performance trajectory of doubling chip performance every 24 to 36 months can only be achieved by integrating multiple processors on a chip rather than through increasing the clock rate of single processors. Multicore architectures have already made their way into the industry [1]–[6], with more aggressive configurations being prototyped, such as Intel's 80-core TeraFlop [7]. Since future technologies offer the promise of being able to integrate billions of transistors on a chip, the prospect of having hundreds of processors on a single chip along with an underlying memory hierarchy and an interconnection system is entirely feasible.

One of the most important benefits of multicore architecture over a traditional single-processor design is the power consumption reduction achieved via reduced clock frequency. Other benefits of multicore architectures include: 1) scalability provided through many dimensions of parallelism, such as thread-level, loop-level, and instruction-level; 2) simpler verification, which in turn reduces the time-to-market and lowers chip costs; 3) better use of the available silicon area since the cores share common logic; and 4) faster and cheaper on-chip communication.

Despite the many advantages of multicore architectures over uniprocessor architectures, one of the key questions raised by many researchers concerns their effectiveness [8], [9]. One aspect of this problem is due to the infancy of the software solutions targeting such architectures. Current programs, compilers, and software architecture techniques in general rely on the fact that there is only one core running in the background [10], and hence it becomes very difficult to effectively use the underlying processing power. There are some initial attempts to target this problem, yet these techniques are relatively new [10].

Manuscript received August 28, 2009; revised September 14, 2010; accepted December 07, 2010. Date of publication January 13, 2011; date of current version May 04, 2011. This work was supported in part by EC FP7-PEOPLE Marie-Curie IRG under Grant 239446.

The author is with the Computer Engineering Department, Bilkent University, Ankara 06800, Turkey (e-mail: ozturk@cs.bilkent.edu.tr).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TE.2010.2102027

Clearly, teaching multicore architectures to today's computer engineers is a desirable goal in every curriculum. In general, computer engineering curricula aim to provide a balanced education in the design and analysis of computer software and computer hardware. According to the IEEE Computer Society and ACM Joint Task Force on Computer Engineering Curricula [11]: "Computer engineers are solidly grounded in the theories and principles of computing, mathematics and engineering, and apply these theoretical principles to design hardware, software, networks, and computerized equipment and instruments to solve technical problems in diverse application domains." In practice, however, this is not always the case. Hence, many computer engineering schools concentrating on software topics lack hardware knowledge, and vice versa [12]–[14].

On the other hand, current language extensions or class libraries are not sufficient to use the available processing power. Software designers need to understand the nature of multicore architectures and learn how to enforce code sequencing on multiple cores. Therefore, it is critical to include multicore education in both undergraduate and graduate computer engineering curricula.

In order to equip students with a better understanding of multicore architecture and its programming, a chip multiprocessors course can be offered with parallel programming concepts on these architectures. This pairing will enable students to learn about state-of-the-art multicore architectures while giving them the opportunity to write parallel programs on these architectures.

Of the multicore-related courses that exist, many [15], [16] use actual hardware to provide a hands-on experience, but this may not always be desirable or possible. First, target architectures may not be available to students due to financial reasons. This is especially true if multiple architectures are being used. Second, maintaining and setting up various architectures may not be feasible. Third, the time required to learn each architecture may not fit within the scope of a course. Hence, it may be preferable to use a software simulator [17] to enable multicore education.

Using a Simics-like simulator in teaching such a course has a number of benefits. First, it is especially valuable for universities with limited financial resources. Second, students are able to run a script to switch from one architecture to another without locating or configuring hardware. Third, debugging and testing is easier in the simulation environment. Fourth, simulators provide system state saving, fault injection, and forward or reverse execution, features that are not available in a hardware environment. Clearly, a simulator is not as accurate or fast as hardware and does not provide the hands-on experience, but it provides a feasible platform for multicore education purposes. It is especially valuable when learning different architectures with different properties in a limited time frame. It also enables tackling the same problem in different architectures, thereby providing "experiential learning."

This paper demonstrates how the Simics platform can be used in teaching different multicore architecture concepts. Specifically, the Simics toolkit was used in the Chip Multiprocessors course at Bilkent University in Ankara, Turkey, to help graduate and senior undergraduate students understand the hardware/software issues related to chip multiprocessors. This particular course enables student participation through the implementation of a semester-long project and continuous discussions.

The remainder of this paper is structured as follows. Section II describes related work. Section III gives details about the Simics platform and how it is used in teaching the above-mentioned chip multiprocessors course. A course overview, including student backgrounds, objectives, teaching methodology, and course plan, is given in Section IV. Section V gives the details of various projects implemented by the students. The course assessment is given in Section VI, while future work and conclusions are presented in Section VII.

II. Related Work

While it is critical to equip students with sufficient multicore architecture knowledge, the important question is how this should be done. To tackle this problem, many computer engineering education conferences and workshops [18]–[21] have recently addressed the importance of multicore education. For example, during the 40th ACM Technical Symposium on Computer Science Education [20], Intel organized a session named "Merging onto the Parallel Programming Highways," where teaching parallel programming in multicore architectures was the main theme of the session. As a result of such efforts, there have been initial attempts to develop an overall computer engineering curriculum incorporating multicore education [22], [23].

As stated earlier, multicore courses [15], [16] are conducted in various universities. However, there are no published papers specifically addressing a multicore course. To the best of the author's knowledge, this paper is the first to present a complete multicore course description and teaching strategy, with detailed discussion on student background, curriculum, and assessment.

III. Simics Platform

Simics [17] is a full-system simulator designed by Virtutech to strike a balance between accuracy and speed in simulation. Specifically, Simics simulates systems containing binary compiled code on several different industry standard architectures (ISAs). Simics can be considered a virtual hardware system that runs the same binary software as the physical target system, including firmware, drivers, the operating system, and the software. It is able to simulate contemporary microprocessors such as PowerPC, x86, ARM, and MIPS.

A. System Simulation

There are both advantages and disadvantages to using simulators. First, with only slight modifications to the configuration file, it is very easy to modify the target architecture. Similarly, it is very convenient to switch to a different architecture. Setting up a particular target system is a matter of running a script,

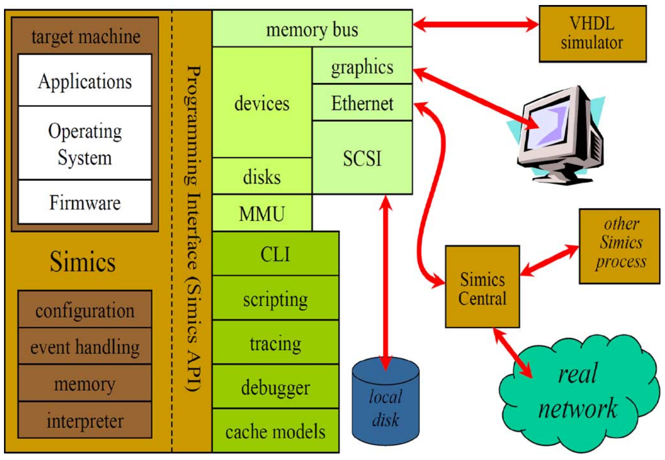

Fig. 1. Simics architecture [17].

without any need to locate or configure hardware boards. Also, debugging and testing is simplified through various debug options and tracing, which are normally not available in physical hardware. Moreover, Simics enables saving system states, injecting faults, forward and reverse execution, replay, and many other features. Being able to test different architectures with minor modifications is one of the key benefits of Simics for this course.

As with any fast simulator, Simics lacks accuracy and timing. A basic Simics setup does not include a cache system since modeling a hardware cache model would slow down the simulation. Instead, it uses the memory system to obtain high-speed simulation. Another limitation in Simics is that it assumes all transactions are executed coherently. That is, memory is always up to date with the latest CPU. Moreover, it assumes device transactions and memory accesses are atomic. On the other hand, in a real processor, transactions have already gone through the L1 and L2 caches before coming to the processor, which may involve cache misses.

However, these are not major concerns for the purposes of this class, as the main objective is to enable students to explore and experience multicore architectures, rather than to perform cycle-accurate simulations. Within this context, Simics has been used to model and evaluate different designs.

B. Simics Architecture

Simics is based on run-time loadable modules that are typically either an extension such as a multilevel cache hierarchy [24] or a device such as an application specified integrated circuit (ASIC) or a field programmable gate array (FPGA). Fig. 1 shows the general Simics architecture [17], where the core Simics platform is composed of the target machine and other components, such as configuration, event handling, memory, and an interpreter. When a configuration file is loaded, the necessary modules are automatically loaded by Simics, whereas the Simics API is responsible for facilitating the external interface. Using Simics, a user can run arbitrary configurations with multiple processors, address spaces, device models, networking models, and clusters.

As shown in Fig. 2, Simics provides a range of accuracy/speed options. Specifically, the simulation platform provides functional accuracy and sufficient abstraction while achieving tolerable performance levels. It is fast enough to run realistic

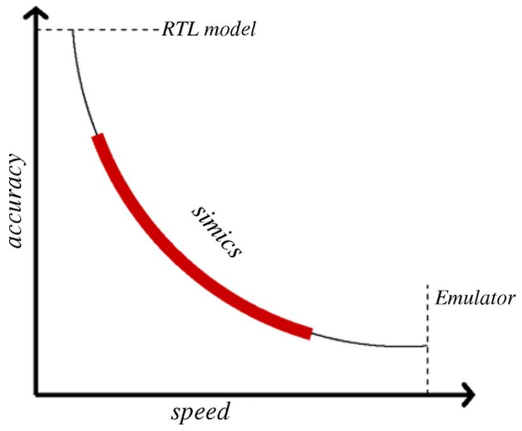

Fig. 2. Correlation between simulation accuracy and speed.

benchmarks on a wide set of unmodified operating systems, including Solaris, Linux, and Windows XP. Moreover, systems ranging from a basic embedded system to a complex multiprocessor environment can be modeled in Simics.

Various statistics can be collected about the system performance, ranging from total execution time, to core utilization, to cache statistics. For example, it is possible to profile the cache accesses by using the timing-model interface of Simics.

Simics provides modeled machines that can be modified to test different multicore characteristics. Publicly available Simics targets include AlphaPC (Alpha 21164, also known as EV5), ARM SA1110 (Intel StrongARMv5), Ebony (PPC440GP 32-bits processor), Fiesta (Sun Blade 1500), IA-64 460GX (Itanium2 processors), Malta/MIPS4kc (32 bit MIPS-4Kc processor), PM/PPC (32-bit PowerPC 750 processor), Simple PPC64 (64-bit PowerPC 970FX processor), Serengeti (Sun Fire 3800-6800 server), SunFire (Sun Enterprise 3000-6500 server), and x86 440BX (x86 compatible processors). Each of these targets comes with an underlying operating system and standard PC devices, such as graphics devices, north and south bridges, and floppy and hard disks. A student can pick a certain target machine and use its preconfigured machine description provided in the target's subdirectory of the Simics tool set.

For example, to run the Tango system (a preconfigured x86 processor with a Linux OS), the following command must be executed: *./simics targets/tango/tango-common.simics*. In this example, "tango-common.simics" is a configuration file that describes the machine properties. In Tango, a Fedora Core 5 is installed, and the base configuration has a single 20-MHz Pentium 4 processor, 256-MB memory, one 19-GB IDE disk, and one IDE CD-ROM.

When the target machine boots up, the corresponding OS will be executed. To access the files in the target machine, the user can employ SimicsFS utility, a Linux kernel file system module that talks to the simulated device. Moreover, it is possible to trace what is going on with the memory accesses, I/O accesses, control register writes, and exceptions during the simulation. Additionally, Simics enables Checkpoints, where the complete state of a simulation can be saved.

As stated, each of the targets comes with an underlying operating system. Therefore, Simics can run the entire software stack, thereby eliminating the need for stubbing or recompiling the target software. More specifically, Simics runs the exact same binaries that run on the physical hardware.

Simics's configuration system represents a simulated machine as a set of objects. Each object (processors, memories, and devices) is defined by a number of attributes. A processor object, as in Fig. 2, will have an attribute, called freq_rm mhz, to define its clock frequency. The following example shows a x86-440bx target configuration description:

$$\text{freq\_mhz} = 200$$

$$\text{cpi} = 1$$

$$\text{disk\_size} = 20496236544$$

$$\text{rtc\_time} = \text{“}2002 - 09 - 1810 : 00 : 00 \text{ UTC"}$$

$$\text{num\_cpus} = 4$$

$$\text{memory\_megs} = 512$$

$$\text{cpu\_class} = \text{“pentium} - 4\text{"}$$

Students can easily modify different parts of the system and test various effects.

C. Simics in Education

Computer architecture-related courses can benefit from such a platform since it is flexible enough to support a wide range of microprocessor types, instruction sets, operating systems, and memory hierarchies. In addition to these attributes, Simics is offered through the Academic Licensing Program [25] to universities at no cost. Simics can be used in many different types of courses, from low-level computer architecture to parallel programming on different systems. At the computer architecture level, Simics can be used to teach the effects of system parameters on performance and energy. System properties can easily be modified through a configuration file, thereby opening up possibilities to experiment with machines of different configurations. Another important use is to teach students assembly language programming. Programs can be executed instruction by instruction, providing the contents of registers and memory changes.

It is also possible to simulate a multicore environment using Simics, even if the host is a single processor. This method provides greater flexibility in testing arbitrary configurations, including multiple processors, multiple nodes, and multiple device configurations. A sample configuration for two processors and a shared memory is given in Fig. 3. In this configuration, there are two x86 processors, namely cpu0 and cpu1, both running at 3500 MHz and sharing the memory space designated as mem0. In addition to these two processors (defined as the first two objects), there is also a memory object that includes a map of the address space for different devices in the system. This specification can be modified to generate various systems ranging from single-processor architectures to heterogeneous chip multiprocessors with different memory hierarchies.

IV. Course Overview

The Chip Multiprocessors course at Bilkent University was first offered in Spring 2008, with an initial enrollment of eight students who were graduate and senior undergraduate students and all computer engineering majors. This course was the first advanced computer architecture course to be offered in the Computer Engineering Department. As such, initial enrollment was not high. However, the number of students doubled in

```
OBJECT cpu0 TYPE x86-hammer
{
    freq_mhz: 3500
    physical_memory: phys_mem0
}
OBJECT cpu1 TYPE x86-hammer
{
    freq_mhz: 3500
    physical_memory: phys_mem0
}
OBJECT phys_mem0 TYPE memory-space
{
    map: ((0xa0000, vga0, 1, 0,0x20000),
      (0x100000, mem0, 0, 0x100000,
      0xff00000),
    ...
}
```

Fig. 3. Sample configuration for two processors with a shared memory.

Spring 2009, making the course among the most popular graduate courses in the department.

A. Objectives

The course is designed to provide a deep understanding of multicore architectures. Although the computer engineering curriculum is designed to provide a balanced education in both software and hardware, in practice, many students have a significant lack of hardware knowledge. This course aims to fill this gap for average students while providing deeper knowledge to the more interested ones.

Instead of students passively listening to an instructor's lecture, Chip Multiprocessors aims to achieve continuous in-class discussions on Simics-based multicore projects, thereby providing students with an experiential learning environment. Students are able to react to lecture material from their personal experiences and apply the course material to real-life situations and/or to new problems. In addition, cooperative learning is achieved through project groups, where structured groups of students are assigned certain research projects.

B. Student Background

Students in the class had no previous exposure to advanced computer architecture topics, except for what had been covered in a sophomore computer organization class. However, students were expected to know general processor design and the basic concepts of pipelining and caches. Computer engineering students had already been exposed to such experience through a Verilog-based processor design project required in the CS224 Computer Organization course. Students also needed to be proficient in programming in C/C++, which is necessary for Simics modifications. Since this course is offered to senior undergraduate and graduate students, it is assumed that students have the required background.

C. Course Organization

Experiential learning is a term used to describe the sort of learning undertaken by students who are able to acquire and apply knowledge, skills, and feelings in an immediate and relevant setting [26]. This course aimed to provide such an environment through simulation.

To achieve this goal, the first part of the course covers multicore evolution, starting from Moore's law, with examples from state-of-the-art multicore architectures. Next, issues such as communication, memory hierarchy, cache coherency, data distribution, task assignment, and operating system involvement are covered. The third part of the course deals with programming on such architectures, using OpenMP and MPI. In the fourth part, advanced topics such as heterogeneous multicore architectures, cache design, network-on-chip architectures, and the like are discussed. The last part is solely dedicated to project development and demonstration.

TABLE I
CHIP MULTIPROCESSORS COURSE SYLLABUS

Week	Topic
1	Introduction to Chip Multiprocessing
2	The March to Multicore and Manycore
3	Simics: A Full System Simulation Platform
4	State-of-the-art CMP Architectures
5	Parallel Programming Concepts: Coverage, granularity, locality
6	Performance Monitoring and Optimizations: Parallelism, Communication, Load Balancing
7	Programming with Message Passing: MPI
8	Multicore programming: OpenMP
9	Compiler Optimizations
10	Cache – Heterogeneous Multicore Architectures
11	Network-on-Chip Architectures
12	Project Requirements-Development
13	Project Development
14	Project Development
15	Project Development-Demonstration

The overall organization of the course for a 15-week semester is shown in Table I. There are three lecture sessions per week where technical discussions and project meetings are held. Note that project discussions start right after the second week to provide sufficient time for students to work on their projects. Simics is introduced in the third week to allow the students to start using the simulator.

There is no textbook that covers all the course topics. As a result, *Multiprocessor Systems-on-Chips* [27], which covers the majority of the topics, was adopted as the course textbook. Students were also referred to other sources of information for topics, such as OpenMP and MPI.

Student learning and instructional effectiveness are measured by student assessment. Assessment instruments used in the Chip Multiprocessors course are exams, homework, and a semester-long project. Midterm/final exams as well as the homework and programming assignments are spread throughout the semester. While all of these instruments comprise the assessment, the project is the most important parameter as it provides hands-on experience.

V. SIMICS PROJECTS

Simics projects in the Chip Multiprocessors course can be classified into two groups. The first of these deals with the architectural properties of multicores, whereas the second tries to optimize/parallelize a certain application onto different multicore architectures. Examples of the first type of project include the following.

- One student team studied the cache protocols on distributed caches, namely snoop protocols and directory-based protocols. For this project, students used the General Execution-driven Multiprocessor Simulator (GEMS) [24], a module built on top of Simics that simulates the memory system in

```
## Simics configuration script.

$num_cpus = 2
## Transaction staller for memory
@staller =
pre_conf_object('stall','trans-stall')
@staller.stall_time = 256

## L2 Shared Unified Cache
@l2c = pre_conf_object('l2c','g-cache')
@l2c.cpus = [conf.cpu0, conf.cpu1 ]
@l2c.config_line_number = 144688
@l2c.config_line_size = 64
@l2c.config_assoc = 4
...
## L1 instruction cache for processor 0
@ic0 = pre_conf_object('ic0','g-cache')
@ic0.cpus = conf.cpu0
@ic0.config_line_number = 512
@ic0.config_line_size = 32
@ic0.config_assoc = 4
...
## L1 data cache for processor 0
@dc0 = pre_conf_object('dc0','g-cache')
@dc0.cpus = conf.cpu0
@dc0.config_line_number = 512
@dc0.config_line_size = 32
@dc0.config_assoc = 4
...
```

Fig. 4. Sample Simics (gcache) cache configuration file.

greater detail. More specifically, they wrote the aforementioned cache coherency protocols within the GEMS framework. They experimented with different numbers of CPUs, ranging from 2 to 64, keeping the interconnection network bandwidth same. Results collected from Splash2 benchmarks [28] (PARSEC benchmarks are now available [29]) express the pros and cons of the two protocols.

- Another team performed a tradeoff analysis between shared versus private caches at levels L1 and L2 on several benchmarks. A sample benchmark tested is dense matrix-matrix multiplication, where a naive algorithm has an $O(n^3)$ complexity. This application is parallelized using OpenMP and tested with different variables such as the cache hierarchy, line size, associativity, cache replacement policy, and number of cache lines. A sample of this cache configuration (gcache) file is shown in Fig. 4.

Examples of the second type of project, which tried to parallelize real-life applications on multicore architectures using Simics, include the following.

- One group of students parallelized Sparse Matrix Vector Multiplication using OpenMP. After the parallelization step, programs were executed on different multicore configurations to measure the scaling of these algorithms with OpenMP. They analyzed different data partitioning heuristics on different architectures, as shown in Fig. 5. In this figure, a sparse matrix populated primarily with zeros is shown, where empty boxes indicate zero entries and nonzero entries are indicated with an "X."
- One student team used Simics to compare a hybrid OpenMP-MPI approach with pure-MPI and pure-OpenMP scenarios. For the specific application in question, students found out that the hybrid OpenMP-MPI performs better than the pure-MPI and pure-OpenMP schemes.
- Other examples of application parallelization projects included parallelization of a 3-D Voronoi Diagram Generation Algorithm and an optimization for a facesim—face recognition—algorithm.

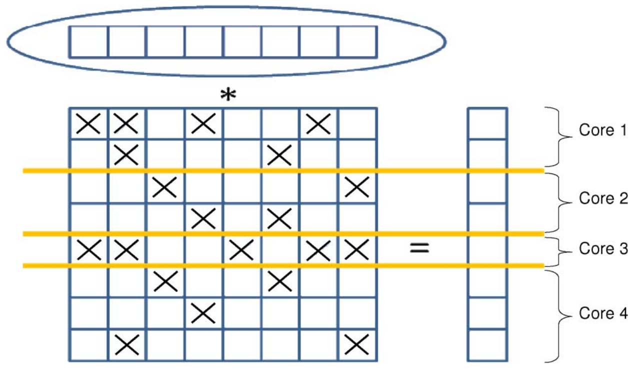

Fig. 5. Data partitioning for matrix multiplication.

A. Sample Project

Most of the student projects tried to modify the architectural properties of a given multicore architecture and measure the effects of various parameters. In this context, a team of students considered die-area optimization in multicore architectures. Specifically, they performed a tradeoff analysis between the cache size and the number of cores. They designed different multicore architectures using Simics, where the number of cores ranged from 1 to 32. For example, they used a 65-nm 145-mm^2 Intel Core 2 Duo as a reference design, where the size of a core is equal to the size of 3-MB L2 cache.

A Pentium 4 architecture with 200-MHz clock is selected for each of the multicore platforms tested. The Simics g-cache mechanism is used to handle the cache configuration of these multicore architectures. While a unified L2 cache is shared by all the processors, each core has a private L1 instruction cache and a private L1 data cache. The L1 instruction cache and L1 data cache are both 16 kB, whereas the L2 cache size configuration varies depending on the number of cores. Assuming the total die area is equal to 13 MB worth of cache area, students tested three different cases: 1) one core with 10-MB L2 cache; 2) two cores with 7-MB L2 cache; and 3) four cores with 1-MB L2 cache. The Simics configuration script for the two-core case is shown in Fig. 4. As can be seen from this example, the number of cores can be modified by the $\mathbf{num\_cpus = 2}$ parameter. In this example, **l2c** is the shared L2 cache, where sharing is indicated by @$\mathbf{l2c.cpus = [conf.cpu0, conf.cpu1]}$. On the other hand, **ic0**, **ic1**, **dc0**, and **dc1** are private L1 instruction and data caches for cores 0 and 1. For each cache in the system, the user can specify the size of the cache, number of cache lines, associativity, and many other characteristics of the cache configuration. Note that some of the important parts of the configuration file, such as the snooping configuration, transaction splitter, timing model, and so on, have been omitted due to space limitations.

Die-area tradeoff analysis was also performed for the high-end cluster processors with a higher number of cores. To test different processor designs, students parallelized the

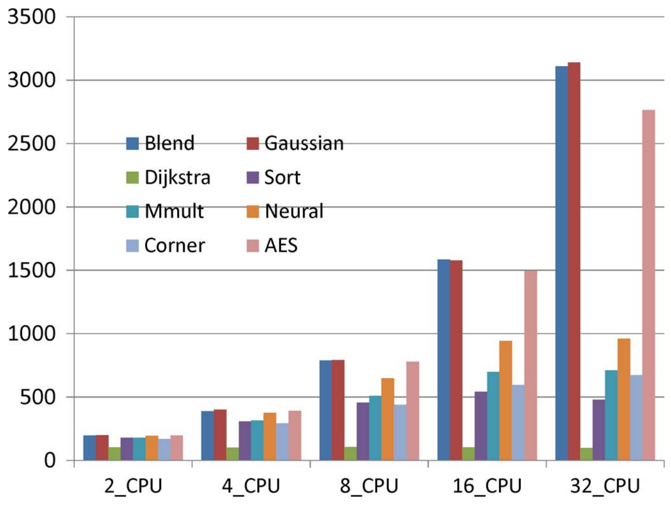

Fig. 6. Experimental results collected in the tested system.

MiBench benchmarks using OpenMP and collected results. Specific benchmarks used in this framework were Image Blending, Gaussian Filter, Digit Recognition, Corner Detection, Merge Sort, Matrix Multiplication, the Dijkstra SSSP (single source shortest path) algorithm, and AES Encryption. Benchmarks were mostly chosen from application domains that can benefit from parallel processing.

Students observed that input sizes and dataset properties significantly affect the run-time performance. To see this effect, they tuned the input data sizes with various cache sizes. While data-bound applications favor an increase in the cache size, simple applications with few data accesses favor an increased number of cores. As can be seen in Fig. 6, 32 core architectures suffer due to the small cache size in Dijkstra SSSP or Merge Sort, whereas applications such as Blending and Gaussian Filter improve with a higher number of cores.

Another student observation was the speed of the simulation in Simics. When the configuration of the target system gets complicated, simulation time is drastically affected, although it is still manageable with high-end systems.

VI. Course Assessment

Course assessment is performed using both student feedback and industry feedback. Student feedback was collected in the 2008 and 2009 Spring semesters, while industry feedback was only available for the 2008 graduates since it is collected toward the end of the year following the course.

A. Student Feedback

Students provided anonymous feedback through the standard Bilkent University course evaluation surveys given at the end of every semester. Table II shows the average responses to six questions about the course in the university surveys for Spring 2008 and Spring 2009. In this table, 1 indicates "strongly disagree," whereas 5 indicates "strongly agree." The responses to questions 1–3 indicate that students found the course interesting and valuable. Moreover, the results for questions 4–6 indicate that students valued the course's contribution to their career and feel that they have gained competency in the area.

Students' written responses reflected their reactions to the following statements: "Provided real world context with the emerging multicore architectures"; "Helped me to understand how multicores work"; "I discovered a new career path."

TABLE II
Anonymous Student Feedback Collected in the Course Evaluation Surveys. 1 = strongly disagree and 5 = strongly agree

Question	Spring 2008	Spring 2009
• Stimulates interest in the subject	5.00	4.56
• Stimulates in-class student participation effectively	4.83	4.78
• Develops students' analytical, creative, critical, and independent thinking abilities	4.67	4.44
• I learned a lot in this course	4.83	4.44
• Exams, assignments and projects required analytical, creative, and critical thinking	4.67	4.33
• Rate the overall teaching effectiveness	4.83	4.38

When the Spring 2008 and Spring 2009 responses are compared, it can be seen that almost all the averages dropped, mainly due to the difference in the class compositions in 2008 and 2009. While most of the students in Spring 2008 were graduate students focused in the subject area, students enrolled in Spring 2009 were a mix of undergraduate students and graduate students.

In addition to the anonymous course evaluation surveys, the course was discussed with students in an informal environment after the semester had finished. This discussion showed that students are very keen to learn more about multicore architectures, programming, and application development on such architectures. The majority of them suggested that it would be a good idea to introduce this topic earlier and integrate it as a part of the computer engineering undergraduate curriculum.

B. Industry Feedback

The Bilkent University Computer Engineering Department meets with its External Advisory Board (EAB) toward the end of every year. The EAB is composed of the department faculty members and industry executive partners from Turkey's leading companies in the field. These companies include both local and global leaders—among them, for example, Microsoft.

Most graduates begin their employment (if local) in one of these companies. After one year of employment, the former students' performances are evaluated in the discussions at this meeting. This valuable feedback promotes university–industry alignment by keeping the curriculum current with global industry needs.

In the Spring 2009 EAB meeting, industry partners provided feedback on the students who graduated in Spring 2008, which included students who took the Chip Multiprocessors course. Table III shows the responses to the relevant questions from the survey, where 1 = strongly disagree and 5 = strongly agree. While the second column in Table III shows the overall average for all graduates, the third column shows the average for those who took the Chip Multiprocessors course.

When the averages in the second and third columns in Table III are considered, students who took the course performed better than the rest of the graduates. Although multicore education cannot be claimed as the only reason for these higher scores, it definitely contributed, especially when questions 1 and 2 in Table III are considered.

TABLE III
INDUSTRY FEEDBACK COLLECTED IN THE EAB MEETING

Question	Overall Average	Course Average
• Demonstrates knowledge of contemporary issues related to computer engineering in general	4.17	4.78
• Able to apply knowledge and skills learned in school to real-world problems	4.13	4.65
• Able to get up to speed on a new concept	4.25	4.45
• Succeeds in difficult design projects	4.05	4.52
• Rate the overall employee effectiveness	4.11	4.60

VII. FUTURE WORK AND CONCLUSION

In future work, it is planned to evaluate the course continuously through student and industry feedback. Simultaneously, an appropriate undergraduate-only version of this course will be developed as a third/fourth-year course. This will be valuable for computer engineering graduates in the multicore computing era.

This paper presents multicore education using a simulator. Instead of universities buying a big and expensive server for students to try and run parallel programs with real workloads, a full-system simulation platform can be used. Simics was used to implement different multicore architectures and for writing parallel programs on these architectures. This simulation framework provides great advantages for teaching and learning multicore architectures and parallel programming when the hardware is not available. From student feedback, industry feedback, and final project results, this simulation-based approach was successful.

REFERENCES

[1] Intel, "Intel Xeon processor 5600 series (quad-core)," Santa Clara, CA, 2010 [Online]. Available: http://www.intel.com/Assets/PDF/prodbrief/323501.pdf

[2] J. A. Kahle, M. N. Day, H. P. Hofstee, C. R. Johns, T. R. Maeurer, and D. Shippy, "Introduction to the cell multiprocessor," *IBM J. Res. Dev.*, vol. 49, no. 4–5, pp. 589–604, 2005.

[3] P. Kongetira, K. Aingaran, and K. Olukotun, "Niagara: A 32-way multithreaded SPARC processor," *IEEE Micro*, vol. 25, no. 2, p. 21, Mar./Apr. 2005.

[4] R. McGowen, "Adaptive designs for power and thermal optimization," in *Proc. 2005 IEEE/ACM Int. Conf. Comput.-Aided Design*, San Jose, CA, 2005, pp. 118–121.

[5] D. Pham, H. Anderson, E. Behnen, M. Bolliger, S. Gupta, P. Hofstee, P. Harvey, C. Johns, J. Kahle, A. Kameyama, J. Keaty, B. Le, S. Lee, T. Nguyen, J. Petrovick, M. Pham, J. Pille, S. Posluszny, M. Riley, J. Verock, J. Warnock, S. Weitzel, and D. Wendel, "The design and implementation of a first-generation cell processor," in *Proc. Solid-State Circuits Conf.*, 2005, vol. 1, pp. 184–592.

[6] M. B. Taylor, J. Kim, J. Miller, D. Wentzlaff, F. Ghodrat, B. Greenwald, H. Hoffman, P. Johnson, J.-W. Lee, W. Lee, A. Ma, A. Saraf, M. Seneski, N. Shnidman, V. Strumpen, M. Frank, S. Amarasinghe, and A. Agarwal, "The RAW microprocessor: A computational fabric for software circuits and general purpose programs," *IEEE Micro*, vol. 22, no. 2, pp. 25–35, 2002.

[7] Intel, "Intel's Teraflops research chip," Santa Clara, CA, Last accessed Dec. 9, 2010 [Online]. Available: http://download.intel.com/pressroom/kits/Teraflops/Teraflops_Research_Chip_Overview.pdf

[8] R. Kumar, D. M. Tullsen, N. P. Jouppi, and P. Ranganathan, "Heterogeneous chip multiprocessors," *Computer*, vol. 38, no. 11, pp. 32–38, Nov. 2005.

[9] R. Kumar, D. M. Tullsen, P. Ranganathan, N. P. Jouppi, and K. I. Farkas, "Single-ISA heterogeneous multicore architectures for multithreaded workload performance," in *Proc. 31st Annu. ISCA*, 2004.

[10] L. Sarno, W. W. Hwu, C. Lund, M. Levy, J. R. Larus, J. Reinders, G. Cameron, C. Lennard, and T. Yoshimori, "Corezilla: Build and tame the multicore beast," in *Proc. Design Autom. Conf.*, Jun. 4–8, 2007, pp. 632–633.

[11] IEEE Computer Society and ACM Joint Task Force on Computer Engineering Curricula, "Curriculum guidelines for undergraduate degree programs in computer engineering," 2004 [Online]. Available: http://www.acm.org/education/curricula-recommendations

[12] J. N. Amaral, P. Berube, and P. Mehta, "Teaching digital design to computing science students in a single academic term," *IEEE Trans. Educ.*, vol. 48, no. 1, pp. 127–132, Feb. 2005.

[13] N. L. V. Calazans and F. G. Moraes, "Integrating the teaching of computer organization and architecture with digital hardware design early in undergraduate courses," *IEEE Trans. Educ.*, vol. 44, no. 2, pp. 109–119, May 2001.

[14] G. Puvvada and M. A. Breuer, "Teaching computer hardware design using commercial CAD tools," *IEEE Trans. Educ.*, vol. 36, no. 1, pp. 158–163, Feb. 1993.

[15] J. Cavazos, "CISC 879—Machine learning for solving systems problems," Last accessed Dec. 9, 2010 [Online]. Available: http://www.cis.udel.edu/~cavazos/cisc879/

[16] A. Lanterma, "Multicore and GPU programming for video games," Last accessed Sep. 4, 2010 [Online]. Available: http://users.ece.gatech.edu/lanterma/mpg08/related_classes.html

[17] Wind River, "Virtutech Simics," Alameda, CA, Last accessed Dec. 9, 2010 [Online]. Available: http://www.virtutech.com/

[18] "International Computing Education Research Workshop (ICER 2009)," Berkeley, CA, Aug. 2009.

[19] "International Congress on Engineering and Computer Education (ICECE 2009)," Buenos Aires, Argentina, Mar. 2009.

[20] "The 40th ACM Technical Symposium on Computer Science Education (SIGCSE 2009)," Chattanooga, TN, Mar. 2009.

[21] "The 14th Annual Conference on Innovation and Technology in Computer Science Education (ITiCSE 2009)," Paris, France, Jul. 2009.

[22] W. Hu, T. Chen, and Q. Shi, "Exploring multicore computing education in China by model curriculum construction," in *Proc. 1st ACM SCE*, Beijing, China, 2008, Article no. 1.

[23] B. Mike, "How the multicore module will integrate into the existing CS programme at Trinity College Dublin," in *Proc. Intel MCCC*, 2006.

[24] M. M. K. Martin, D. J. Sorin, B. M. Beckmann, M. R. Marty, M. Xu, A. R. Alameldeen, K. E. Moore, M. D. Hill, and D. A. Wood, "Multifacet's general execution-driven multiprocessor simulator (GEMS) toolset," *Comput. Archit. News*, vol. 33, no. 4, pp. 92–99, Nov. 2005.

[25] Wind River, "Wind River University Program," Alameda, CA, Last accessed Dec. 9, 2010 [Online]. Available: http://www.windriver.com/universities

[26] D. A. Kolb and R. Fry, "Toward an applied theory of experiential learning," in *Theories of Group Process*, C. Cooper, Ed. London, U.K.: Wiley.

[27] W. Wolf and A. Jerraya*, Multiprocessor Systems-On-Chips (Systems on Silicon)*. San Mateo, CA: Morgan Kaufmann, 2004.

[28] W. Heirman, "SPLASH-2 for Solaris on SPARC on Simics," Last accessed Dec. 9, 2010 [Online]. Available: http://trappist.elis.ugent.be/~wheirman/simics/splash2/

[29] Princeton University, "The PARSEC benchmark suite," Princeton, NJ, Last accessed Dec. 9, 2010 [Online]. Available: http://parsec.cs.princeton.edu/

Ozcan Ozturk (M'07) was born in Istanbul, Turkey, in 1978. He received the Bachelor's degree from Bogazici University, Istanbul, Turkey, in 2000, the M.Sc. degree from the University of Florida, Gainesville, in 2002, and the Ph.D. degree from Pennsylvania State University, University Park, in 2007, all in computer engineering.

He is currently an Assistant Professor with the Department of Computer Engineering, Bilkent University, Ankara, Turkey. Prior to joining Bilkent University, he was a Software Optimization Engineer with the Cellular and Handheld Group, Intel (Marvell), Chandler, AZ. He also held Visiting Researcher positions with the ALCHEMY Group of INRIA, Paris, France, and with the Processor Architecture Laboratory (LAP), Swiss Federal Institute of Technology of Lausanne (EPFL), Lausanne, Switzerland. His research interests are in the areas of multicore and manycore architectures, power-aware architectures, and compiler optimizations.

Dr. Ozturk is a Member of the Association for Computing Machinery (ACM), the Gigascale Systems Research Center (GSRC), and the European Network of Excellence on High Performance and Embedded Architecture and Compilation (HiPEAC). He is currently serving as an Editor and a reviewer on leading IEEE, ACM, and other journals. He is a recipient of the 2006 International Conference on Parallel and Distributed Systems (ICPADS) Best Paper Award, a 2009 IBM Faculty Award, and a 2009 Marie Curie Fellowship from the European Commission.