EDUCacheIC: Interactive and Collaborative Successor of the EDUCache Simulator

Sasko Ristov, Blagoj Atanasovski, Marjan Gusev, and Nenad Anchev
Ss. Cyril and Methodius University
Faculty of Information Sciences and Computer Engineering,
Rugjer Boshkovik 16, PO Box 393,
1000 Skopje, Macedonia
Email: sashko.ristov@finki.ukim.mk, blagoj.atanasovski@gmail.com,
marjan.gushev@finki.ukim.mk, nenad_ancev@hotmail.com

***Abstract*—Introducing our EDUCache simulator [1] in the Computer Architecture and Organization course increased the students willingness to learn more details about CPU cache memory. A positive experience of students interest to work on laboratory exercises with our EDUCache simulator lead us to develop an EDUCacheIC, a new interactive and collaborative version of EDUCache simulator. EDUCacheIC offers the students to work in a group with one student member of the group having a role as a teacher. The student - teacher creates a CPU architecture and tasks for other students in the group. This interactive and collaborative simulator will improve the learning and teaching process of the Computer Architecture and Organization course reducing the teacher's efforts during the laboratory exercises.**

***Index Terms*—*Cache*; *Computer Architecture and Organization*; *CPU*; *Education*; *Multiprocessor*.**

I. Introduction

The Computer Architecture and Organization course is usually in the first or second study year and is an important area in the undergraduate computer science (CS) curricula [2]. Teaching the computer architecture fundamentals is a very difficult process which requires a lot of effort from both teachers and students. Introducing an appropriate visual simulator can significantly lighten the teaching process and increase the students' interest in hardware courses. There are a lot of visual simulators to support the learning of the basic concepts in computer organization. However, none of them covers all topics in computer architecture and organization [3]. Some of the simulators are focused on education, while the others on data profiling.

Since we have not found any available simulator in the literature and on Internet, designed to teach the students about concepts of the cache memory, hierarchy and organization, cache size, cache line, cache associativity, etc, we have recently developed the EDUCache simulator [1] which visually presents cache hits and misses, cache line fulfillment, and cache associativity problem. It allows the students to design their own CPU with different cache hierarchy defined with appropriate levels (L1 to L3), various configuration of cache owners, i.e. either private cache per core or shared among several or all CPU cores, various cache size, realization of the n-way cache set associativity, cache line sizes, and cache replacement policy, with ability for applying different cache replacement policies per different cache levels.

We have also developed hands-on exercises to support the students using EDUCache simulator [4], including several case studies with examples how to use the EDUCache simulator in the learning process.

Despite the positive experience of introducing the EDUCache simulator in the laboratory exercises, we have observed that EDUCache simulator is not collaborative. That is, each student works with its own desktop application, generates its own CPU and executes the examples. This was the motivation to develop a new interactive and collaborative **EDUCacheIC** simulator. It is a client-server based simulator which allows the students to learn the CPU architecture and organization in a team. One student of a team works as a teacher (student - teacher) and creates particular CPU (multi-core multiprocessor) in the simulator. After that, the other members of the team execute different trace files according to their particular tasks scheduled by the student - teacher. Now the students' trace files compete for the core and they can analyze the parallel execution of trace files, thus simulating the concurrent execution of several processes.

Nowadays, the teachers should give more attention to the multi-core processors architecture and organization [5]. The most important additional feature of the EDUCacheIC simulator is the simulation of CPU cache memory while multi tasking memory trace is executed with different CPU affinity, i.e., binding or unbinding a particular process with one or several CPU cores. Each student can execute one or more processes, each with particular number of threads and particular $CPU\ AFFINITY$.

This feature of the EDUCacheIC simulator will not only increase the student knowledge about the multi-core multiprocessor's architecture and organization, but it will also increase their knowledge in software performance engineering, that is, they can easily improve and optimize their software programs for other software courses.

The rest of the paper is organized as follows. Section II explains main features of the predecessor, i.e., the EDUCache simulator and the architecture and features of its successor, the interactive and collaborative EDUCacheIC simulator. Related

work about the simulators used in education, especially in computer architecture and organization is presented in Section III. The final Section IV concludes our work and present our plans for future work.

II. EDUCacheIC Simulator Architecture

This section describes the EDUCacheIC Simulator, the inherited features of its predecessor, the EDUCache simulator, as well as the new features, roles and user interface.

A. Inherited Features from the Predecessor (EDUCache)

The EDUCacheIC simulator was built as an extension of the predecessor, the EDUCache simulator. It uses all the functionalities of EDUCache simulator and the underlying simulation architecture has not been changed. The implementation is also in the Java programming language. The simulation architecture represents a simple mapping of every CPU cache memory parameter to a specific object described by a class. The students create the CPU cache memory architecture by specifying the required parameters (the number of cores, cache levels, cache set associativity, cache replacement policies, cache size, cache line etc) and combining the elements together.

The construction begins with creating separate cache levels. The students can configure various cache level parameters: cache memory size, cache line size, cache associativity and replacement policy. Then the cache levels are combined in a CPU core specifying which instance of the created cache levels belongs to which CPU core, providing the ability to create private or shared cache levels. Each core can have private or shared cache on different cache levels (generally L1 to L3). Most of today's modern multiprocessors have private L1 and L2 caches, while L3 cache is shared. The simulation is executed by creating address trace files. These files contain the sequence of main memory addresses being accessed by each CPU core and are subsequently read from the cache. A memory address is either found in the cache memory generating cache hit, or it is not found in the cache and loaded from the lower cache level or main memory, thus generating cache miss.

While executing the EDUCache simulator collects the number of cache hits and misses for each CPU cache level and the number of cache hits and misses per core. The data are used in the students analysis of different cache level behavior in each core. The configuration of the simulator can be read from previously saved files.

B. Description of New Roles

EDUCacheIC does not change the functionalities of its predecessor; it represents the added features that allow the students to collaborate among each other in groups during the laboratory exercises also improving the interactivity of the exercises. Before we go into explaining workflow of the new functionalities, we will introduce four new roles of the EDUCacheIC simulator:

- *Student-teacher*: a student responsible for creating a CPU architecture with a specified number of cores and cache levels with specified parameters;

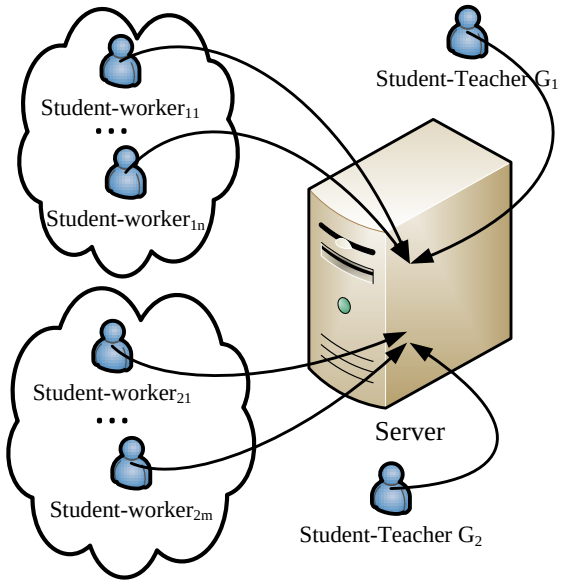

Fig. 1. EDUCacheIC architecture

- *Student-worker*: a student responsible for creating a trace file of memory addresses to be read by the CPU architecture created by the student-teacher;
- *Master-program*: an instance of EDUCacheIC started on the student-teacher's computer (server). It is responsible to schedule the commands given by the student-workers on the architecture created by the student-teacher of a student's group. It also broadcasts the commands to other worker-programs in order to simulate the execution to the students; and
- *Worker-program*: an instance of EDUCacheIC started on the student-worker's computer. It logs in to a master-program and sends the commands to be executed on the architecture that is created by the student-teacher. It also listens for broadcasts that are sent by the master-program of the same student group.

C. Description of New Features

Apart of the predecessor functionalities, EDUCacheIC has new additional features and allows the students to collaborate among each other in groups during the laboratory exercises.

Figure 1 depicts the new main feature "collaboration" of EDUCacheIC compared to its predecessor, i.e., a group of students creates a team. A student-teacher (one student of a group) creates a CPU with one or multiple cores together with their cache memory architecture. Each student-worker of the group is required to create its own trace file. The trace file represents the commands (memory addresses) that will be executed on the already created CPU by the student-teacher. The student-workers connect their worker-programs to the master-program of the student-teacher.

The master-program and all connected worker-programs represent a single computer and several processes, and not a distributed system. The traces executed by the worker-programs are programs stored in a single continuous memory space executed on the CPU created by the student-teacher in the master-program.

At the beginning, the worker-programs register themselves at the master-program and wait for the student-teacher to create the CPU cache architecture. When all the parameters are set, the student-workers can create and set their trace files. The trace files can contain an arbitrary number of commands. The commands contain information about which main memory address should be accessed and the $CPUAFFINITY$, that is, which core of the CPU should execute them. Depending of the settings of the student-worker, the commands in their trace file can be executed with an absolute or relative affinity. In absolute affinity, the student-worker must specify explicitly which core (or cores) of the CPU must execute the commands with the ID of the core. In relative affinity, the student-worker specifies only an arbitrary core ID that needs to execute the commands. In the case of relative affinity, the student-teacher chooses which core is to execute the commands. In a single trace file, all commands must have only one type of affinity specified.

When all the trace files are submitted, the student-teacher assigns CPU cores to all the trace files with relative affinity and assigns a position to all workers in the command-queue. The top of the command-queue shows the next command in line that will be executed by the CPU. One by one command from each of the registered student-workers is executed in a round-robin fashion. Master program broadcasts a notification to all registered worker-programs containing which command is executed (which main memory address was read and which core of the CPU executes the command). The execution ends when all commands from all trace files are finished.

Although this type of scheduling is not practically implemented in any real computer systems (operating systems), we have implemented it in this way since the focus is on learning the CPU and its cache memory, rather than the process scheduling. We believe that it will be easier for the students to understand the area of computer architecture and organization. The scheduler removes a command at the top of the queue and assigns the command to a core. Since the primary target users of our EDUCacheIC simulator are the undergraduate students of the first study year, the concurrency is not implemented in the real meaning of the word.

Figure 2 depicts an example of the collaborative EDUCacheIC simulation scenario. The CPU consists of four cores C_0, C_1, C_2 and C_3, each with dedicated L1 and L2 caches. All four cores share the last L3 cache. Additionally, the student-teacher should configure the cache size, set associativity, cache line and cache replacement policy of each cache memory.

The presented scenario consists of three students A, B and C. Student A executes only one process (memory reads) on core C_0. Student B executes two processes such that the first process is executed also on core C_0 and the second process, which consists of two threads is executed with $CPU\_AFFINITY$ = "2 3", i.e., by one thread on cores C_2 and C_3. Finally, student C executes one process with memory reads executed in parallel with 4 threads with default $CPU\_AFFINITY$, i.e., on all four cores.

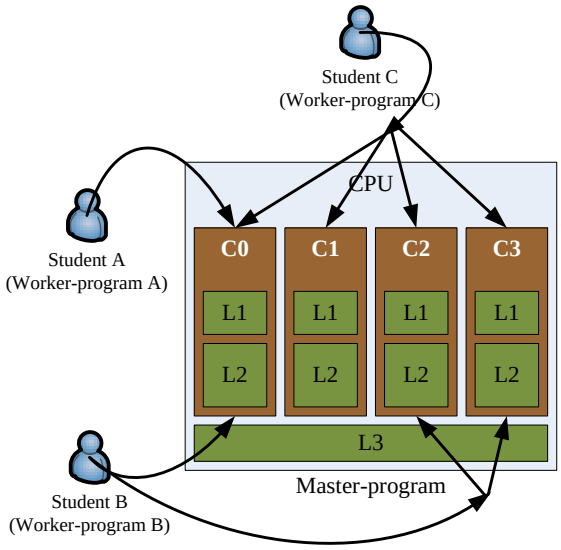

Fig. 2. An example of collaborative EDUCacheIC simulation scenario

Let's analyze the processes (threads) on each core. C_0 executes three processes that compete for L1 and L2 cache memories of core C_0. Core C_1 executes only the process of student C, while cores C_2 and C_3 execute by one thread of students B and C, correspondingly.

D. Protocol

The communication between the master-program and the worker-programs is implemented via a simple custom protocol over TCP sockets. The Java implementation of TCP sockets creates a full-duplex channel between the two parties. At the beginning, this channel is used for connection establishment and to transfer the information about the architecture and state of the cache memory. Once the connection is established, the master-program sends a broadcast message telling the other worker-programs about the status of the simulator. The possible states are:

- NOT_CONFIGURED - the master-program has opened its connection port. It can accept connections by the worker-programs, but a CPU cache architecture has not been defined by the student-teacher yet;
- CONFIGURED_WAITING - the student-teacher has already created the CPU cache architecture. The master-program is waiting for the student-workers to submit their trace files and new connections can also be established to other worker-programs;
- READY - all student-workers have submitted their trace files. The student-teacher has to assign CPU's and create

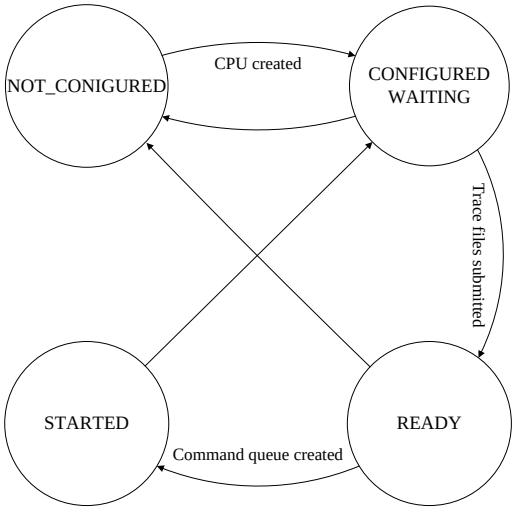

Fig. 3. States and transitions of EDUCacheIC simulator

the command queue, new connections can not be established in this state; and
- STARTED - the command queue is created and simulation is started; new connections can not be established.

Figure 3 depicts the transitions between the states that the simulator can make. The simulator starts in the NOT_CONFIGURED state and after all the trace files have been executed, or the simulation has been stopped, it goes in the CONFIGURED_WAITING state.

If the master-program has been configured with a CPU, the architecture details are sent to the connected worker-programs. Transitions between states are broadcast to all connected worker-programs. When the trace files have been submitted, the simulator is configured by the student-teacher to create the command queue. These settings are also broadcast to all worker-programs. Then the simulator is ready to transition to the STARTED state, where it executes a command from the top of the command-queue and broadcasts the command to all worker-programs. The worker-programs listen to these broadcasts and update the displays for their local worker-students. The protocol is shown as a sequence diagram in more detail in Figure 4.

E. EduCacheIC User Interface

As the EDUCacheIC is an extension of the original simulator, the steps for creating the CPU architecture remain the same. The student-teacher must put the simulator in construction mode where the required cache levels must be created first and combined in CPU cores. After that, the CPU cores can be saved into a Cache Configuration File.

Figure 5 gives an example of creating a cache configuration with two cores. The cache elements have already been created and they are displayed in the table on the right side of the form. The left side of the form shows the first core with an UID "CO", which has not yet been created. This form is inherited of the predecessor and is adapted with the successor's requirements.

Figure 6 is a screen-shot of the EDUCacheIC when it is in the CONFIGURED_WAITING state. The top left panel displays the loaded configuration, a dual-core CPU where the first two cache levels are private for each core and the third level is shared. The level one cache of core C0 is 2-way associative with 32 bytes in size, the cache line is 8 bytes and the replacement policy is FIFO.

The top right panel shows a list of connected worker-programs with their specified identifier names and their IPs. Because the simulator is in the second state (CONFIGURED_WAITING) it can accept more connections. The bottom left panel displays the list of the users that has submitted a trace file. The bottom right panel is disabled until all the student-workers submit a trace file. The student-teacher uses this panel to create the command-queue by specifying each worker-program's starting position in the round-robin command scheduling to the CPU.

III. Related Work

We have found two main approaches to teach the hardware-based courses:

- Using *visual simulators* - this includes all benefits of distance learning, open education resources (OER), on-line laboratories, etc; and
- Working on *real hardware* - this includes all benefits of FPGA-based configurable processors.

Both approaches offer benefits to teachers, students and universities. The former approach is usually cheaper since by using the simulators and on-line tools, the universities can share their laboratory equipment, thus removing the obstacles of cost, time-inefficient use of facilities, inadequate technical support and limited access to design and laboratory resources [6]. It can awake the students' interest in hardware [3]. Using OER, the teachers and students can share their experiences.

The latter approach is also very important [7], [8], [9]. The newest FPGAs are with considerable price. They can be used by the students in laboratory exercises [10], [11] in order to develop, implement and monitor both hardware and software of multi-core processor systems on real hardware.

We prefer the visual simulators to lighten the teaching and learning processes of hardware-based courses in the CS curriculum, leaving the working on real hardware for better students to work on more complex projects and research:

- HADES simulator [12] and our EDUCache simulator in the Computer Architecture and Organization course;
- Visual OOO simulator [13] for ILP dynamic out-of-order executions in the Advanced Computer Architectures course; and
- Several visual simulators [14] for the Microprocessors and Micro-controllers course.

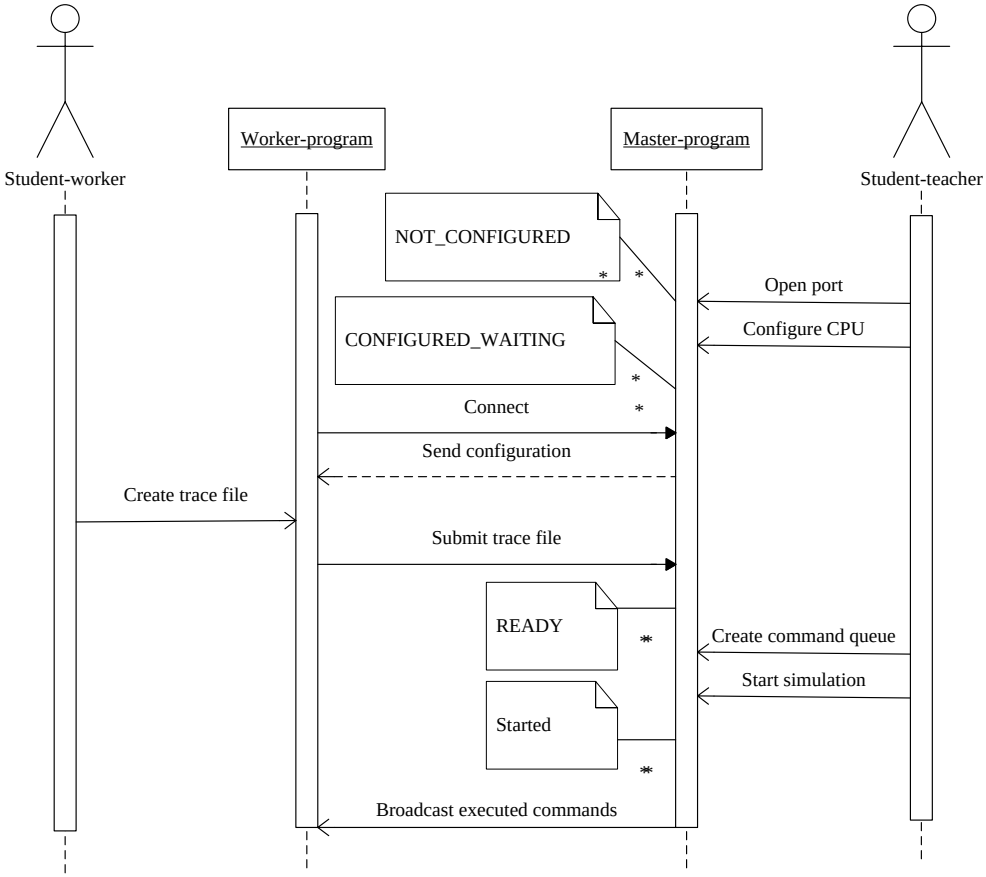

Fig. 4. Sequence diagram of EDUCacheIC simulator

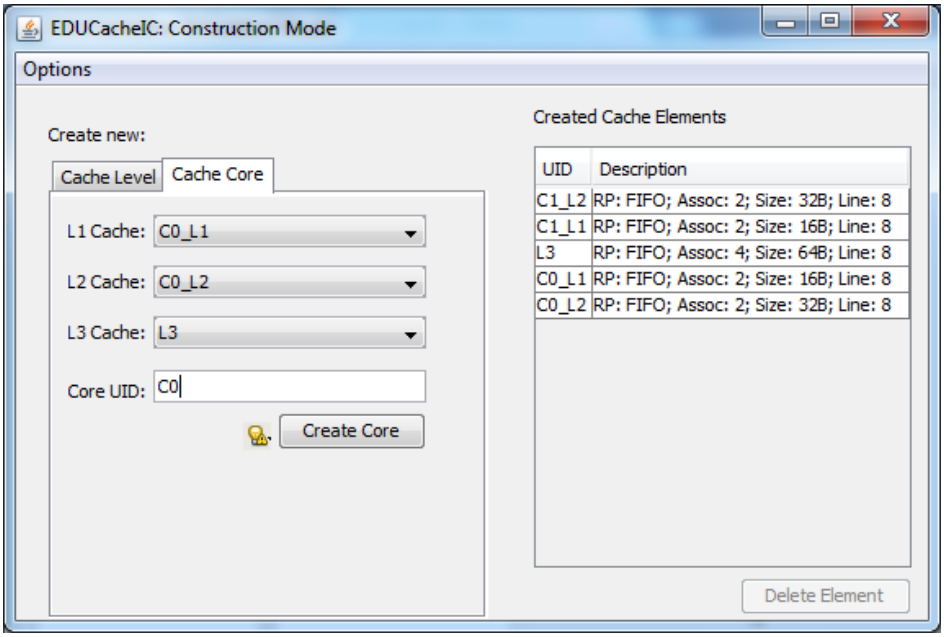

Fig. 5. Creating a cache configuration with two cores

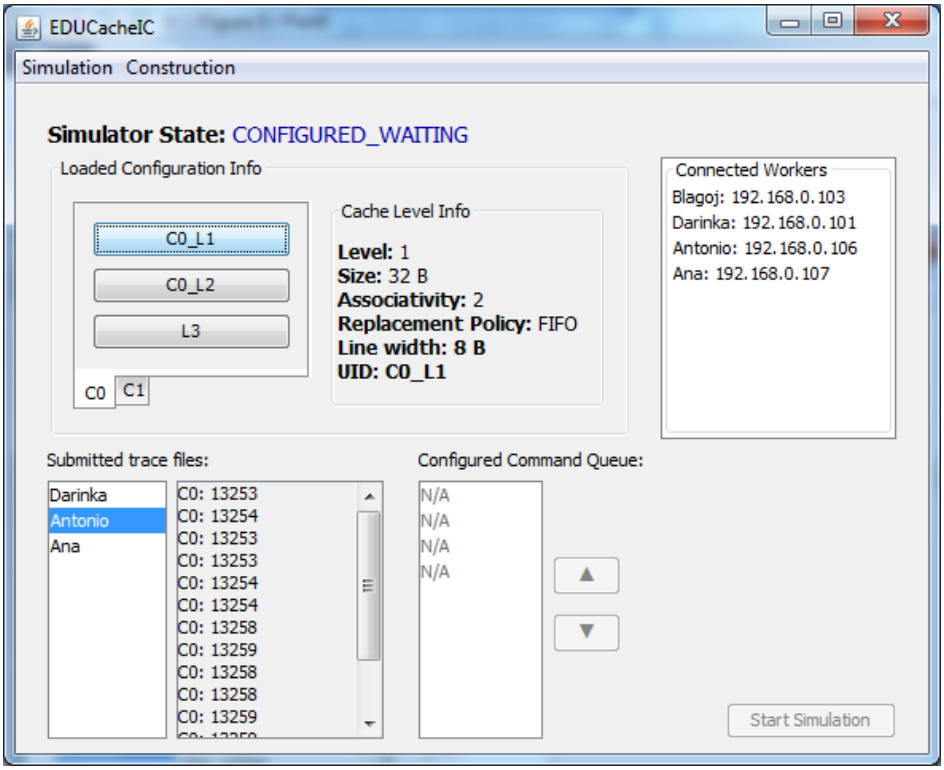

Fig. 6. The EDUCacheIC interactive and collaborative simulation

It is very difficult for the teachers of the Computer Architecture and Organization course to decide which simulator to be used since many visual simulators exist with different features: EduMIPS64 [15], Dinero IV [16], CMP$im [17], HC-Sim [18], Herruzo's [19], Valgrind [20] (with its module Cachegrind), SimpleScalar [21] and SMPCache [22]. Brief explanation of all these simulators is given in our previous research [4]. We concluded that all these simulators have one or several deficiencies:

- Not primarily developed for teaching, but for data profiling;
- A lack of educational features;
- Do not cover all topics in computer architecture and organization area;
- Do not have prepared hands-on exercises to cover all learning objectives;
- Do not include collaborative learning features; and
- Cover more advanced topics.

Our EDUCache simulator [1] covers all these deficiencies and can be used in the Computer Architecture course to lighten the teaching and learning [23], [4]. However, we have detected that the EDUCache simulator lacks collaborative and interactive features.

Bruffee [24] gives an overview of history of collaborative learning. He discusses that it is a way to engage students more deeply with the context and enable a more efficient learning environment. Several authors [25], [26], [27], [28], [29] have already introduced interactive course materials in engineering courses, which can be very useful in students' learning. Hamada [30] introduced an integrated environment for active learning of computer engineering, as support for collaborative learning. Bachour et al. [31] show that using an interactive tool increases the students' motivation and leads to more balanced collaboration. Weng et al. [32] proposed a technique for learning the Boolean logic through the interaction of the students and the interactive computer games, i.e., the Pac-Man game rule tuning.

Therefore, we have developed a successor of our EDUCache simulator, i.e., the interactive and collaborative EDUCacheIC simulator, described in this paper.

IV. Conclusion and Future Work

This paper describes the EDUCacheIC visual simulator, the interactive and collaborative successor of our recently developed EDUCache simulator for teaching and learning the Computer Architecture and Organization course. Beside the inherited features from the predecessor, such as designing CPU with different hierarchies in cache levels (L1 to L3), various cache owner configurations, visual simulation of cache hits and misses, cache line fulfillment, cache associativity problem and different cache replacement policies, the EDUCacheIC simulator has a lot of additional features like CPU affinity and multi-core processing.

The EDUCacheIC allows the CS students to learn the Computer Architecture and Organization course easily through team work on the laboratory exercises. Since it is a web

application, the students can work remotely from home.

Both EDUCache and EDUCacheIC simulators simulate only memory reads, regardless of wether a memory read generates cache hits or cache misses. We will continue to upgrade the EDUCacheIC simulator to simulate the memory writes to the students, both write invalidate and write through.

Our plan is to adapt and implement the EDUCacheIC simulator into several other courses in CS curricula, such as Operating Systems and Parallel and Distributed Computing. That is, we would like to include simulation of real concurrent execution with an overview on cache coherency algorithms. Also, executing parallel threads or processes can be used for analyzing shared memory threading.

REFERENCES

[1] B. Atanasovski, S. Ristov, M. Gusev, and N. Anchev, "EDUCache simulator for teaching computer architecture and organization," in *Global Engineering Education Conference (EDUCON), 2013 IEEE*, March 2013, pp. 1015–1022.

[2] R. Shackelford, A. McGettrick, R. Sloan, H. Topi, G. Davies, R. Kamali, J. Cross, J. Impagliazzo, R. LeBlanc, and B. Lunt, "Computing curricula 2005: The overview report," *SIGCSE Bull.*, vol. 38, no. 1, pp. 456–457, Mar. 2006.

[3] B. Nikolic, Z. Radivojevic, J. Djordjevic, and V. Milutinovic, "A survey and evaluation of simulators suitable for teaching courses in computer architecture and organization," *Education, IEEE Transactions on*, vol. 52, no. 4, pp. 449 –458, nov. 2009.

[4] S. Ristov, B. Atanasovski, M. Gusev, and N. Anchev, "Hands-on exercises to support computer architecture students using EDUCache simulator," in *2013 Federated Conference on Computer Science and Information Systems FEDCSIS'13*, Krakow, Poland, Sep 2013, p. in press.

[5] ACM/IEEE-CS Joint Interim Review Task Force, "Computer science curriculum 2008: An interim revision of CS 2001, report from the interim review task force," 2008, [retrieved: Aug., 2013]. [Online]. Available: http://www.acm.org/education/curricula/ComputerScience2008.pdf

[6] D. Pop, D. G. Zutin, M. E. Auer, K. Henke, and H.-D. Wuttke, "An online lab to support a master program in remote engineering," in *Proceedings of the 2011 Frontiers in Education Conference*, ser. FIE '11. USA: IEEE Computer Society, 2011, pp. GOLC2–1–1–GOLC2–6.

[7] I. Kastelan, D. Majstorovic, M. Nikolic, J. Eremic, and M. Katona, "Laboratory exercises for embedded engineering learning platform," in *MIPRO, 2012 Proc. of the 35th Int. Conv.*, 2012, pp. 1113–1117.

[8] J. Qian, R. Wang, S. Shi, Y. Zhu, and Z. Xie, "Simplifying and integrating experiments of hardware curriculums," in *Computer Science and Information Technology (ICCSIT), 2010 3rd IEEE International Conference on*, vol. 9, 2010, pp. 610–614.

[9] D. Kehagias and M. Grivas, "Software-oriented approaches for teaching computer architecture to computer science students," *Journal of Communication and Computer*, vol. 6, no. 12, pp. 1–9, Dec. 2009.

[10] X. Wang, "Multi-core system education through a hands-on project on FPGAs," in *Frontiers in Education Conference (FIE), 2011*, 2011, pp. F2G–1–F2G–6.

[11] J. H. Lee, S. E. Lee, H.-C. Yu, and T. Suh, "Pipelined CPU design with FPGA in teaching computer architecture," *Education, IEEE Transactions on*, vol. 55, no. 3, pp. 341–348, 2012.

[12] University of Hamburg, "HADES - Hamburg design system," 2002, [retrieved: Aug., 2013]. [Online]. Available: http://tams-www.informatik.uni-hamburg.de/applets/hades/html/

[13] A. Misev and M. Gusev, "Visual simulator for ilp dynamic ooo processor," in *Proceedings of the 2004 workshop on Computer architecture education: held in conjunction with the 31st International Symposium on Computer Architecture*. ACM, 2004, p. 18.

[14] N. Ackovska and S. Ristov, "Hands-on improvements for efficient teaching computer science students about hardware," in *Global Engineering Education Conference (EDUCON), 2013 IEEE*, March 2013, pp. 295–302.

[15] D. Patti, A. Spadaccini, M. Palesi, F. Fazzino, and V. Catania, "Supporting undergraduate computer architecture students using a visual MIPS64 CPU simulator," *Education, IEEE Transactions on*, vol. 55, no. 3, pp. 406 –411, aug. 2012.

[16] J. Edler and M. D. Hill, "Dinero iv trace-driven uniprocessor cache simulator," 2012, [retrieved: Aug., 2013]. [Online]. Available: http://pages.cs.wisc.edu/~markhill/DineroIV/

[17] A. Jaleel, R. S. Cohn, C.-K. Luk, and B. Jacob, "CMPSim: A pin-based on-the-fly multi-core cache simulator," in *The Fourth Annual Workshop MoBS, co-located with ISCA '08*, 2008.

[18] Y.-T. Chen, J. Cong, and G. Reinman, "HC-Sim: a fast and exact L1 cache simulator with scratchpad memory co-simulation support," in *Proc. of the 7-th IEEE/ACM/IFIP Int. conf. on HW/SW codesign and system synthesis (CODES+ISSS '11)*. USA: ACM, 2011, pp. 295–304.

[19] E. Herruzo, J. Benavides, R. Quislant, E. Zapata, and O. Plata, "Simulating a reconfigurable cache system for teaching purposes," in *Microelectronic Systems Education (MSE '07). IEEE International Conference on*, 2007, pp. 37 –38.

[20] Valgrind, "System for debugging and profiling Linux programs," [retrieved: March, 2013]. [Online]. Available: http://valgrind.org/

[21] SimpleScalar LLC, "SimpleScalar tool set," [retrieved: Aug., 2013]. [Online]. Available: http://www.simplescalar.com/

[22] University of Extremadura, "SMPCache - simulator for cache memory systems on symmetric multiprocessors," [retrieved: Aug., 2013]. [Online]. Available: http://arco.unex.es/smpcache/

[23] S. Ristov, M. Gusev, B. Atanasovski, and N. Anchev, "Using EDUCache simulator for the computer architecture and organization course," *International Journal of Engineering Pedagogy (iJEP)*, vol. 3, no. 3, pp. 47–56, 2013.

[24] K. A. Bruffee, "Collaborative learning and the "conversation of mankind"," *College English*, vol. 46, no. 7, pp. 635–652, 1984.

[25] S. Hadjerrouit, "Learner-centered web-based instruction in software engineering," *IEEE Trans. on Educ.*, vol. 48, no. 1, pp. 99–104, Feb. 2005.

[26] M. Hamada, "Visual tools and examples to support active e-learning and motivation with performance evaluation," in *Technologies for E-Learning and Digital Entertainment*, ser. Lecture Notes in Computer Science, Z. Pan, R. Aylett, H. Diener, X. Jin, S. Gbel, and L. Li, Eds. Springer Berlin Heidelberg, 2006, vol. 3942, pp. 147–155. [Online]. Available: http://dx.doi.org/10.1007/11736639_20

[27] S. Li and R. Challoo, "Restructuring an electric machinery course with an integrative approach and computer-assisted teaching methodology," *Education, IEEE Transactions on*, vol. 49, no. 1, pp. 16–28, 2006.

[28] J. Masters, T. M. Madhyastha, and A. Shakouri, "Educational applets for active learning in properties of electronic materials," *IEEE Trans. on Educ.*, vol. 48, no. 1, pp. 29–36, Feb. 2005.

[29] R. Nelson and A. Islam, "Mes - a web-based design tool for microwave engineering," *Education, IEEE Transactions on*, vol. 49, no. 1, pp. 67–73, 2006.

[30] M. Hamada, "An integrated virtual environment for active and collaborative e-learning in theory of computation," *Learning Technologies, IEEE Transactions on*, vol. 1, no. 2, pp. 117–130, 2008.

[31] K. Bachour, F. Kaplan, and P. Dillenbourg, "An interactive table for supporting participation balance in face-to-face collaborative learning," *Learning Technologies, IEEE Transactions on*, vol. 3, no. 3, pp. 203–213, 2010.

[32] J.-F. Weng, S.-S. Tseng, and T.-J. Lee, "Teaching boolean logic through game rule tuning," *Learning Technologies, IEEE Transactions on*, vol. 3, no. 4, pp. 319–328, 2010.