Using EDUCache Simulator for the Computer Architecture and Organization Course

http://dx.doi.org/10.3991/ijep.v3i3.2777

Sasko Ristov, Marjan Gusev, Blagoj Atanasovski, and Nenad Anchev
Ss. Cyril and Methodius University, Skopje, Macedonia

***Abstract*—The computer architecture and organization course is essential in all computer science and engineering programs, and the most selected and liked elective course for related engineering disciplines. However, the attractiveness brings a new challenge, it requires a lot of effort by the instructor, to explain rather complicated concepts to beginners or to those who study related disciplines. The usage of visual simulators can improve both the teaching and learning processes. The overall goal is twofold: 1) to enable a visual environment to explain the basic concepts and 2) to increase the student's willingness and ability to learn the material.**

A lot of visual simulators have been used for the computer architecture and organization course. However, due to the lack of visual simulators for simulation of the cache memory concepts, we have developed a new visual simulator EDUCache simulator. In this paper we present that it can be effectively and efficiently used as a supporting tool in the learning process of modern multi-layer, multi-cache and multi-core multi-processors.

EDUCache's features enable an environment for performance evaluation and engineering of software systems, i.e. the students will also understand the importance of computer architecture building parts and hopefully, will increase their curiosity for hardware courses in general.

***Index Terms*—Cache Memory; CPU; Education; Multiprocessor; Performance.**

I. INTRODUCTION

The Computer Architecture and Organization course is acknowledged as a significant part of the body of knowledge and an important area in undergraduate computer science (CS) curricula [1], [2]. Nevertheless, the problem arises since the high-level programming languages do not provide a clear picture of how the program is executed by the computer. As a consequence, learning of computer architecture and organiza-tion decreases the student's interest and deeper understanding.

Teaching computer architecture is a very difficult process and requires a lot of effort from both instructors and students [3]. It is usually scheduled in the first study year and it is almost always a completely new course for the students. Visual simulators lighten the teaching process and significant-ly improve CS student interest in hardware generally [4]. Teachers must select appropriate hands-on exercises, assignments and projects. Liang [5] realizes a nice survey of hands-on assignments and projects.

Modern multi-processors use multilayer cache memory system [6] to balance the gap between CPU and main memory and to speedup data access. Thus students must understand not only the architecture, but the organization inside the multi-processor. We have not found appropriate educational simulator that will help the students to understand easily all cache parameters and their impact to program execution. In this paper we present the EDUCache simulator that visually presents cache hit and miss, cache line fulfillment, cache associativity problem [7], for both sequential and parallel algorithm execution.

The rest of the paper is organized as follows. In Section II we discuss the related work in the literature about other simulators similar to our EDUCache simulator. In Section III, we present the course "Computer Architecture and Organization" organization and challenges that motivated us to develop the EDUCache simulator. Section IV describes the architecture of EDUCache simulator. EDUCache user interface and different working modes are described and depicted in Section V. Several demo case studies are presented in Section VI-B. The final Section VII is devoted to conclusion and future work.

II. RELATED WORK

We found many visual simulators that help students to surmount particular fundamental parts of computer architecture and organization. However, there is no single simulator which covers all topics in computer architecture [8]. Most visual simulators are not designed to teach the students neither about cache memory hierarchy and organization, nor it's internal parameters such as cache size, cache line, cache associativity, cache inclusivity etc.

Visual EduMIPS64 is a learning aid for instruction pipelining, hazard detection and resolution, exception handling, interrupts, and memory hierarchies [9]. It is a very powerful learning tool and it simulates the complete pipeline architecture of the MIPS64 processor, but it does not offer a thorough overview on how the cache memory hierarchy works or affects execution. Also the simulator requires the students to be already familiar with the MIPS64 ISA before they are able to use the simulator which is impossible for the first year computer science students.

Dinero IV is the cache simulator that simulates a memory hierarchy with various caches [10]. It is a powerful and accurate tool but it can simulate only single core systems. However, it is only a command line tool that offers neither visualizations nor explanations when the students use it. Fraguela et al. [11] define a fully parameterizable models applicable to n-way associative caches, but only for LRU (Least Recently Used) replacement policy. Our EDUCache simulator simulates both FIFO (First In First Out) and LRU cache replacement policies for all cache levels.

CMP$im is a simulator based on the Pin binary instrumentation tool [12]. It is a better simulator offering multi-core support and data gathering for all levels of the cache. However, the capturing of results is more complex than our EDUCache simulator. HC-Sim simulator is also based on Pin that generates traces during runtime and simulates multiple cache configurations in one run [13].

Herruzo et al. [14] designed a configurable cache system simulator that helps students in understanding the process of cache look up and writing elements in the cache memory. They made it possible to configure the block size, the number of blocks in a set and cache capacity, but only with a few predefined values. Also, their implementation does not have the support for multi-core cache systems. Misev and Gusev have developed visual simulator for ILP dynamic out-oforder executions [15], covering aspects on shelving, register renaming, issuing, dispatch and other elements of out-of-order executions.

Valgrind [16] with its module Cachegrind is the most used profiler for cache behavior. Although Valgrind goes deep into code and provides information about each function of the program, it provides the information only for the first and the last level cache. Modern multi-processors possess three level caches and our EDUCache simulator can simulate middle level L2 cache behavior. Another important advantage of our simulator is its platform independence. Valgrind also does not support shared memory parallel system when using threads.

All these simulators were not primarily developed for teaching the cache memory although most of them are visual. They lack educational features since they are built to complete the simulation as fast as possible rather than to present the architecture and organization of cache memory system in a modern multi-processor. EduCache simulator offers step by step simulation allowing the students to pause the simulation and analyze the cache hits and misses in each cache level.

III. THE COURSE COMPUTER ARCHITECTURE AND ORGANIZATION

This section briefly describes the course Computer Architecture and Organization.

The course's main objective is to offer the students a clear understanding of the main computer architectures, performance of the computer parts and the whole computer system. It also covers the topics of today's modern multi-chip and multi-core multi-processors, as well as the digital logic circuits.

A. *Course Organization*

The course is organized in three parts: theoretical lectures with 2 classes per week, tutorials with 2 classes per week and practical lab tutorials with 1 class per week. Lectures and theoretical tutorials are organized in larger groups of less than 100 students, while practical lab tutorials are carried out in computer labs in groups of up to 20 students, with each student working on its own workstation. Prerequisites for enrolling in the course are previously completed courses in Discrete Mathematics.

Theoretical lectures cover the computer abstractions and technology, the computer language (MIPS), computer arithmetic, the processor, memory, storage, and multi-chip multicore multi-processors [17].

Tutorials are divided in two parts. The first midterm covers the topics: computer arithmetic, codes and performance parameters, while the second part is devoted to digital logic circuits. Hands-on lab assignments follow the topics of theoretical tutorials. A comprehensive overview of the core concepts and organization of the Computer Architecture and Organization course can be found in [2].

B. *Challenges and Motivation*

The previous section briefly described the course organization. We have analyzed the student's results and determined that they have more problems with the topics of theoretical lectures compared to the exercises, and more precisely, the topics planned for the second midterm, i.e., the processor, memory, I/O and parallelization.

Our analysis shows that although these topics are covered during the theoretical lectures, neither theoretical nor practical tutorials are provided for these topics, since the exercises are devoted to the design of digital logic circuits. Even more, IEEE Computer Society and ACM stated that more attention should be given to the multi-core processors architecture and organization, instead of the hardware logic design level [18].

Therefore, we realized a small survey of the students with two simple questions:

- Q1: Should we remove the hands-on lab tutorials with topics for hardware logic design level and introduce topics for multi-processor architecture and organization?; and
- Q2: Should we introduce visual simulator in lab tutorials for multi-processor architecture and organization?

We offered two simple answers: Yes or No for both questions.

61 Students participated in the survey.

Figure 1 presents the results of the survey for the first question Q1. Almost 64% of the students answered affirmatively to change the lab tutorials of the second part of the course, i.e. replace topics for hardware logic design with multi-processor architecture and organization lab tutorials.

The results of the survey for the second question Q2 are depicted in Figure 2. Although we expected that the students will answer both questions with the same answer, the results of the second question are more affirmatively. Almost 92% of the students answered Yes, i.e., they would like visual simulator to learn the multi-processor architecture and organization.

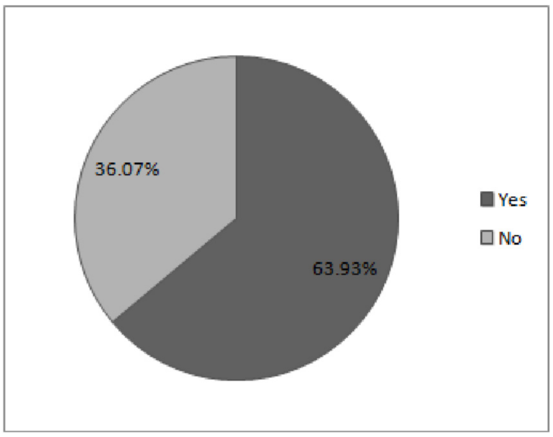


Figure 1. Results of the survey for Q1

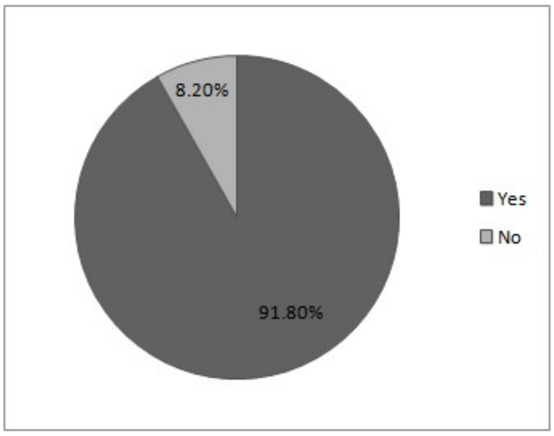


Figure 2. Results of the survey for Q2

What have we learned from the survey? There is a substantial group of students (one third) that want to have detailed lab tutorial on hardware logic design. It seems that they need extensive help to understand these concepts. We also conclude that visual simulators will substantially improve the learning process, as a conclusion of the second question.

Overall conclusions, approved our commitment to build a new visual simulator for multi-processor's architecture and organization, as necessity for realization of the Computer Architecture and Organization course.

IV. EDUCACHE SIMULATOR FEATURES AND INTERFACES

This section describes the main features and interfaces of new proposed EDUCache simulator. It is a platform independent simulator developed in JAVA which main simulation is described by a set of Java classes, each for a different CPU cache parameter. EDUCache simulator allows the students to design a multi-layer cache system with different multi-core multi-cache hierarchy and to analyze sequential and parallel execution of particular algorithm.

A. *Single-core or Multi-core Multi-processor*

Students can create homogeneous multi-processor with one or several cores with particular processor speed.

B. *Chip Cores and Cache Owners*

Each chip can have one or several homogeneous cores. Each core has access to some cache of different cache level (generally L1 to L3). Particular cache can be owned by one, several or all cores of the chip. In general, L1 and L2 caches are dedicated per core in modern multi-processors, while L3 cache is shared among several or all cores in the chip.

C. *Cache Parameters per Cache Level*

Particular cache level parameters can vary. Cache is determined by cache memory size, cache line size, cache associativity and replacement policy. Even more, the "inclusivity" among cache levels is also defined. EDUCache simulator allows the students to configure all these cache parameters.

D. *Data Statistics*

EDUCache simulator collects various data about the configured multi-processor system during its simulation. The most important parameters are:

- The number of hits and misses for each CPU cache level regardless of shared or dedicated cache per core; and
- The number of hits and misses per core.
- The logged data helps the students in their analysis of different cache level behavior in each core.

V. EDUCACHE USER INTERFACE

Each visual simulator devoted to teaching must have user friendly graphical user interface. Our main focus was to create an easy-to-use and easy-to-learn visual simulator. In this section we describe the EDUCache user interface and its working modes in details.

The EDUCache interface is visual and user friendly. It uses the Multiple Document Inteface (MDI) paradigm. EDUCache simulator works in two modes: *Design* and *Simulation*.

The students can design particular multi-processor in design mode and save the multi-processor's cache hierarchy and parameters to use it in the simulation mode. The simulation mode offers the students to load a set of memory addresses and run the simulation either with automated execution on intervals or step by step on user input.

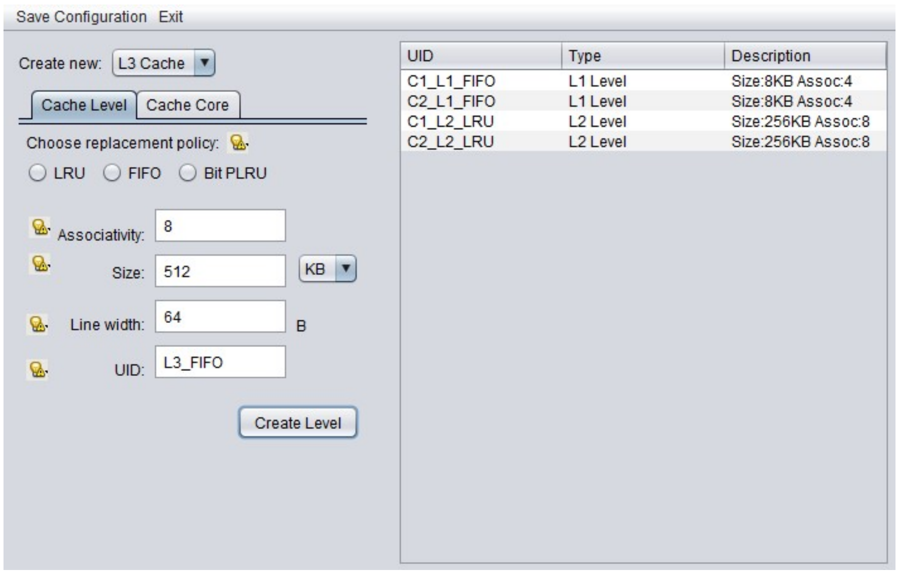
A. *Design Mode*

The students can configure various cache parameters and levels to create instances of cache levels and share them among chip cores. Figure 3 depicts an overview of EDUCache user interface in design mode.

The main window contains 2 main frames. The panel on the left side offers the students to select what kind of a cache level instance they would like to create. The students should create the cache levels with fulfilling the parameters such as cache replacement policy, cache associativity, cache size (in KB or MB), cache line size and Unique ID (UID) for that particular cache level. The students use UIDs to create a chip core instance configuring which previously created cache level instances will be incorporated as L1, L2 or L3 caches for particular CPU core. Figure 3 also depicts an example how a student can create very easily L3 cache level instance with a FIFO replacement policy, 8 way set associativity, 64 byte cache line, 512 KB L3 cache size and UID L3 FIFO.

The right frame of the window shows the previously created instances with their type, UID and description. The grid shows that 4 other instances have been previously created, two L1 and two L2 instances, as depicted in Figure 3.

After creating cache level instances, the students can create a core, selecting cache instances from the list of previously created ones (visible in the table in the right frame) for each cache level. Figure 4 depicts a design of a core with UID C1 that has L1 and L3 caches with FIFO replacement policy and L2 cache with LRU replacement policy. The EDUCache simulator offers the students to configure the "inclusivity" among the cache levels, as well.



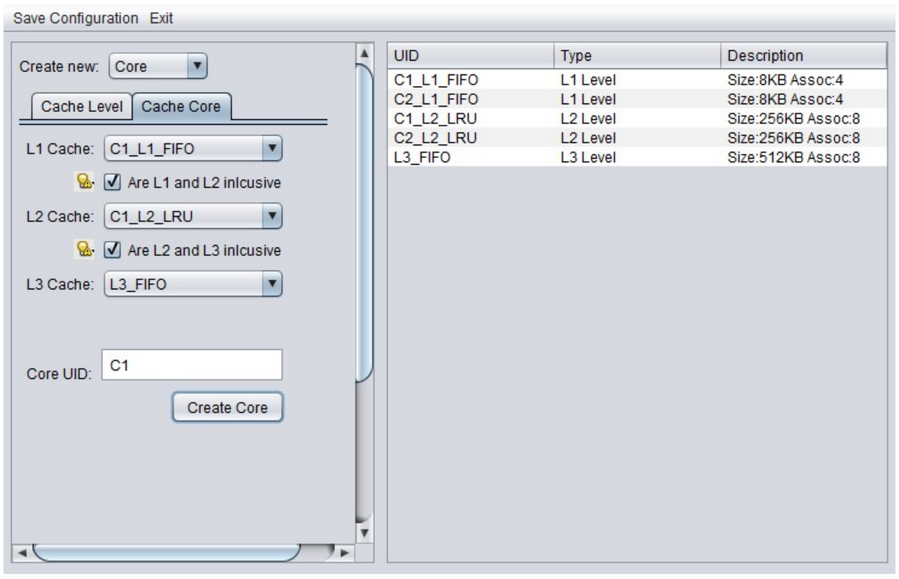
Create new: L3 Cache

Choose replacement policy:
LRU FIFO Bit PLRU

Associativity: 8
Size: 512 KB
Line width: 64 B
UID: L3_FIFO

UID	Type	Description
C1_L1_FIFO	L1 Level	Size:8KB Assoc:4
C2_L1_FIFO	L1 Level	Size:8KB Assoc:4
C1_L2_LRU	L2 Level	Size:256KB Assoc:8
C2_L2_LRU	L2 Level	Size:256KB Assoc:8

Figure 3. Overview of design mode of EDUCache simulator - Creating L3 cache instance



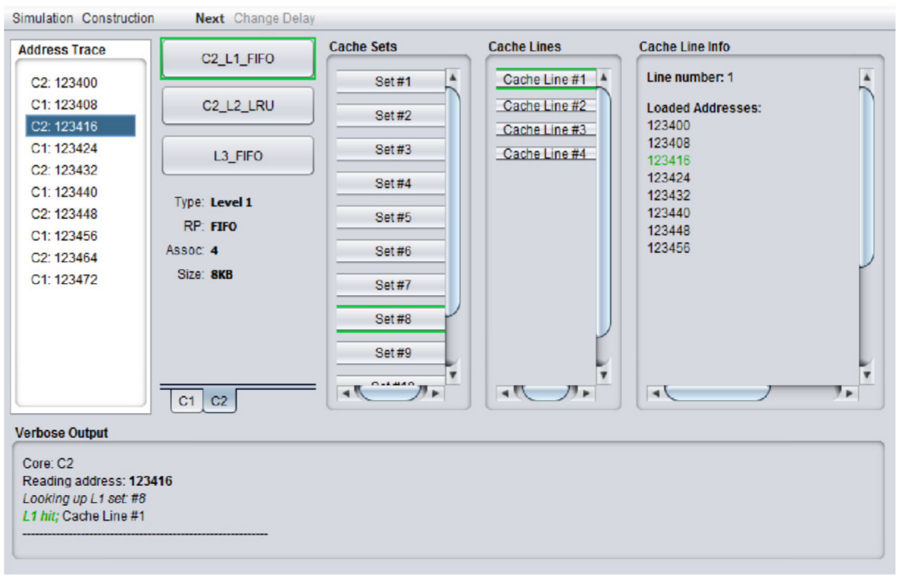
Create new: Core

L1 Cache: C1_L1_FIFO
☑ Are L1 and L2 inlcusive
L2 Cache: C1_L2_LRU
☑ Are L2 and L3 inlcusive
L3 Cache: L3_FIFO

Core UID: C1

UID	Type	Description
C1_L1_FIFO	L1 Level	Size:8KB Assoc:4
C2_L1_FIFO	L1 Level	Size:8KB Assoc:4
C1_L2_LRU	L2 Level	Size:256KB Assoc:8
C2_L2_LRU	L2 Level	Size:256KB Assoc:8
L3_FIFO	L3 Level	Size:512KB Assoc:8

Figure 4. Overview of design mode of EDUCache simulator - Creating a CPU core



```
C2: 123400
C1: 123408
C2: 123416
C1: 123424
C2: 123432
C1: 123440
C2: 123448
C1: 123456
C2: 123464
C1: 123472
```

Type: **Level 1**
RP: **FIFO**
Assoc: **4**
Size: **8KB**

Line number: 1

Loaded Addresses:
```
123400
123408
123416
123424
123432
123440
123448
123456
```

```
Core: C2
Reading address: 123416
Looking up L1 set: #8
L1 hit; Cache Line #1
--------------------------------------------------------
```

Figure 5. Overview of Simulation mode - hit in L1 level of core C2, set #8, line #1, address 123416

The main advantage of our EDUCache simulator compared to others is in allowing the design of multi-processor with two or more cores (for example C1, C2, C3 and C4) which can share the same cache level instance. For example, choosing the same instance (for example L3 FIFO) for all cores as L3 cache will design a shared last level L3 cache. The students can configure two by two cores to share last level cache, and can share L1 or L2 caches among more cores, as well.

Finally the students can save the created configuration that represents a CPU chip. They configure which core instances they would like to include on the chip and they are prompted where to save the configuration file. The configuration file format is discussed in Appendix A.

Additional information icons are positioned next to each label, to alleviate the computer architecture learning process and the usage of the simulator. Figure 6 depicts these icons which give short explanations and directions to the students of the purpose of the field they are configuring, but also allows the students to learn and understand the features and the purpose of the cache parameter represented with that field. The explanations are shown as tool tip boxes when the students click the icons or hold the mouse pointer over a particular icon. Completing the design mode successfully, the students have designed a multi-core chip with different caches as described in this section. Now they can move to the simulation mode in order to simulate some memory accesses and analyze which of them will generate hit or miss in a particular cache level of particular core.

B. Simulation Mode

EduCache's Design mode is used for configuration and the Simulation mode for execution, simulating and analyzing. After a configuration of the CPU chip with multiple cores per chip and multiple cache levels per each core, the students should load the memory addresses and then run the simulation of accessing for those addresses. Figure 5 depicts the Simulation mode. Its main window consists of:

- Simulation Control Menu Bar;
- Loaded Address Trace Frame;
- Verbose Output Frame; and
- Visual Representation Frame.

Let's explain their purpose and layout in more details.

1) Simulation Control Menu Bar:

The menu bar is the central control hub for the simulation process. It contains 2 menus, *Simulation* and *Construction* as depicted in Fig 7.

Construction menu has two options, i.e. "Create New Configuration" and "Load Configuration". The former opens Design Mode, while the latter prompts for a location of a configuration file. The Simulation menu has options to load a study case file, to load a trace file, to start the simulation, to pause it, to stop it completely, and to enter into step by step working mode.

All options except Load Study Case in Simulation menu are disabled at the beginning when no configuration is loaded in the EDUCache simulator. The particular menu items are enabled after the appropriate configuration file is loaded.

2) Loaded Address Trace Frame:

Loaded Address Trace Frame is located on the top of the left side and shows the contents of the trace file. The items consist of two parts. The first part is a core's UID showing which core should read the address. The second part is the physical address that is loaded. The item that is being examined is highlighted during the simulation.

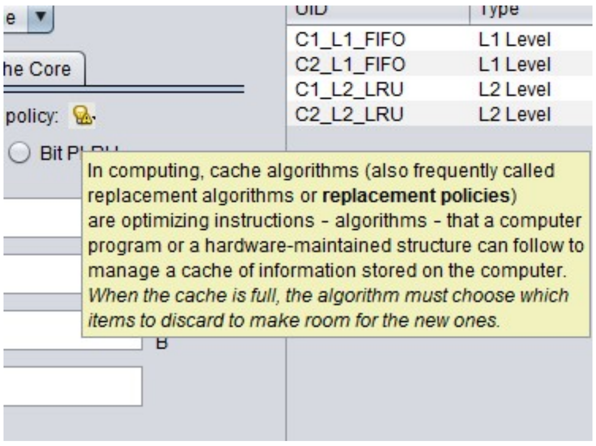
3) Verbose Output Frame:

Verbose Output Frame is placed on the bottom of the window as output pane. EDUCache simulator gives the student the explanation of the whole cache lookup process while the simulation is running. It shows the addresses that are read by cores, the search in L1 cache and selecting the set in which the address is supposed to map, the result, i.e. cache hit or miss, the cache line number if it is hit and the evicted line if the chosen set was full and read miss is generated. The whole output is written out to a text file for later revision and analysis.

4) Visual Representation Frame:

Visual Representation Frame is the main feature of simulation mode and is the third frame of the window. EDUCache gives a visual representation to the lookup process. This frame is divided in 4 parts, each representing a different level of the cache level architecture:

- *Core Pane* - each tab in this pane contains the information for particular CPU core. For example, C1 and C2 are the cores that are depicted in Figure 5. Selecting each of these tabs shows the designed cache levels in selected core presented as buttons with the UID's of the instances chosen as captions. The students click the cache level buttons to obtain the information about that cache instance and the cache sets for that level that are loaded in the Cache Sets Pane.
- *Cache Sets Pane* - this pane presents the cache sets for selected cache level instance of selected core in Core Pane according its design (cache set associativity).
- *Cache Lines Pane* - this pane presents the cache lines of selected cache set in Cache Sets Pane.
- *Cache Line Info Pane* - this pane presents the addresses located of selected single cache line in Cache Line Pane. It shows the loaded addresses in selected cache line and it can be also configured to show statistics about the selected cache line as number of cache writes and number of cache reads.



UID	Type
C1_L1_FIFO	L1 Level
C2_L1_FIFO	L1 Level
C1_L2_LRU	L2 Level
C2_L2_LRU	L2 Level

In computing, cache algorithms (also frequently called replacement algorithms or **replacement policies**) are optimizing instructions - algorithms - that a computer program or a hardware-maintained structure can follow to manage a cache of information stored on the computer. *When the cache is full, the algorithm must choose which items to discard to make room for the new ones.*

Figure 6. A tool tip explanation box showing information about replacement policy

The Visual Representation Frame is repeatedly updated during the simulation. The whole process can be described with the following sequence:

- A memory address is read from the trace file along with the information which core is doing the reading.
- Lookup begins by checking the L1 level of the selected core.
- A cache set is chosen depending on the physical address, the set associativity and number of cache sets.
- The cache lines are searched for the required element in the chosen set.
  - If the required element is found in one of the cache lines, then it is highlighted with green in the Cache Line Info pane. The cache line containing the found element in the Cache Lines Pane is highlighted also with green. The corresponding cache set of the Cache Sets Pane and L1 cache level instance in the Core Pane are also highlighted with green.
  - If the required element is not found in L1, then the same elements that are described in the previous step will be highlighted, but with red indicating that cache miss is generated. Lookup process will continue in L2 and if L2 cache miss occurs, analogue items are highlighted with red and the similar process continues for L3 cache.

The verbose output is produced and shown during the process in the Verbose Output Pane.

C. EDUCache Simulation

In this section we present the procedure to find a particular address in the caches.

Figure 8 presents the activity diagram. A memory address is read from the trace file. Lookup begins by checking L1 cache of the selected core. A cache set is chosen depending on the physical address, the set associativity and number of cache sets. Then the cache lines in the chosen set are searched for the required element. If the required element is found in one of the cache lines the element is highlighted with green in the Cache Line Info pane, the line containing the element in the Cache Lines and the corresponding cache set and L1 instance. If the element is not found in L1, the same elements described in the previous step are highlighted with red, and lookup process continues in L2 cache, with the same steps. The verbose output is produced and showed in the bottom frame simultaneously.

The following two sections describe the simulation of L1 cache hit, and L1 miss and hit in lower cache level.

1) Simulation of L1 Cache Hit:

Figure 5 depicts an example of a L1 cache hit. It shows that the second core C2 is about to require the physical address 123416. Looking at the L1 cache size, associativity and number of sets, EDUCache simulator calculates that this address should be mapped into set number 8. Set #8 contains 4 cache lines. Two steps before, C2 core required the address 12400 and it wrote into Cache Line #1 along with the items of the whole cache line, i.e. up to address 123456, including the required address 123416. Thus, L1 cache hit will occur and the address in the Cache Line Info Pane, Cache Line #1, Set #8 and C2 L1 FIFO are all highlighted with green.

2) Simulation of L1 Cache Hit:

Figure 5 Simulation of L1 Cache Miss and Hit in Lower Cache level: When a cache miss occurs in a particular

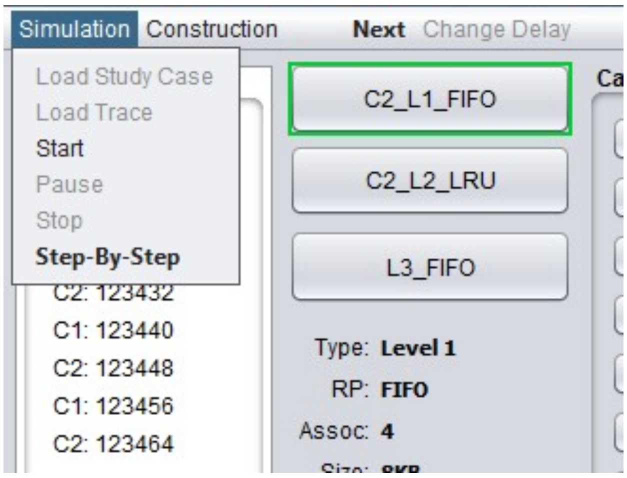


Figure 7. Menu bar in Simulation Mode

cache level, then all the cache level, cache set and cache lines that were looked up are highlighted with a red. Figure 9 depicts the scenario of L1 cache miss and successful lookup cache level L2, i.e. L1 cache miss and L2 cache hit. The EDUCache simulator highlights L1 elements with red and L2 elements with green. The results from this lookup are also seen in the verbose output.

VI. TUTORIALS AND DEMO CASE STUDIES

In this section we present several demos and case studies on the usage of the EDUCache simulator for learning purposes. They present examples of characteristic memory access patterns and will enable a visual presentation that will support the learning process and more efficient understanding of the processor and its cache memory architecture and organization. A more comprehensive study on realized tutorials and experiments using the EDUCache simulator [19] explain basic concepts of a modern processor processor and its cache memory, in the course of Computer Architecture and Organization.

A. Tutorials

Ristov et al. [19] define hands-on lab tutorials for the EDUCache simulator. The following exercises present the essence of these tutorials:

- *Exercise 1: Intro to EDUCache Environment* It introduces basic cache concepts about the multicore multi-chip processors and explains the EDUCache simulator environment and user interface to select appropriate parameter in the cache configuration. The learning objectives include at least: 1) managing the configuration files, 2) simulation of trace files, 3) analysis of results.
- *Exercise 2: Different Cache Parameters*
- The goal is to present several cache memory parameters: size, associativity and the principles of multiple cache levels. The learning objectives include at least: 1) impact of parameters on a specific program execution, 2) effect of time and space locality, 3) creation of multiple configurations files and single memory

trace files, 4) deeper analysis and explanation of the results and parameter impact.

- *Exercise 3: Overview of Cache Set Associativity and Replacement Policies*

This exercise addresses issues of cache set associativity and replacement policies, being the most complex concepts to understand. The learning objectives include at least: 1) impact of parameters on a specific program execution, 2) using different configurations and address traces; and 3) understanding the results and bringing relevant conclusions.

B. Demo Case Studies

Demo cases studies are based on special files containing a reference to configuration and trace files. They set up a practical example with a certain goal. Using these demo case study files will not only help the students in understanding computer architecture and organization focusing on CPU cache memory, but also will help in determining the average number of clocks per particular cache level hit or miss. In this section we propose several demo case studies that simulate some extremes.

- *Always Cache Hits* This example uses a trace file that always generates cache hits for a given loaded configuration. It generates a specific address pattern to access the elements stored in a particular cache level. For example, sorting a small array of elements is an algorithm that will always generate cache hits.
- *Always Cache Misses due to Cache Capacity Problem* It presents another extreme example, where cache misses are always generated due to the cache capacity problem. The address pattern is such that the required elements can not be found in a particular cache level. For example, accessing the elements of one column in a huge-enough squared matrix in raw major will always produce cache misses. The detailed analysis for storing the matrix elements in the cache can be found in [20].
- *Always Cache Misses due to Cache Associativity Problem* The last example is another extreme, since it generates cache misses due to the cache associativity problem. The address pattern creates a situation where all of the required elements can be found in a particular cache set, introducing a specific situation when only one (or small number of sets) are used out of the cache memory. Gusev and Ristov [7] give a comprehensive analysis how, when and why maximum performance drawbacks appear when using the set associative cache. For example, accessing the elements of one column for characteristic matrices in raw major will always produce cache misses since all the column elements will be stored in one cache set, and the rest of the cache will be empty.

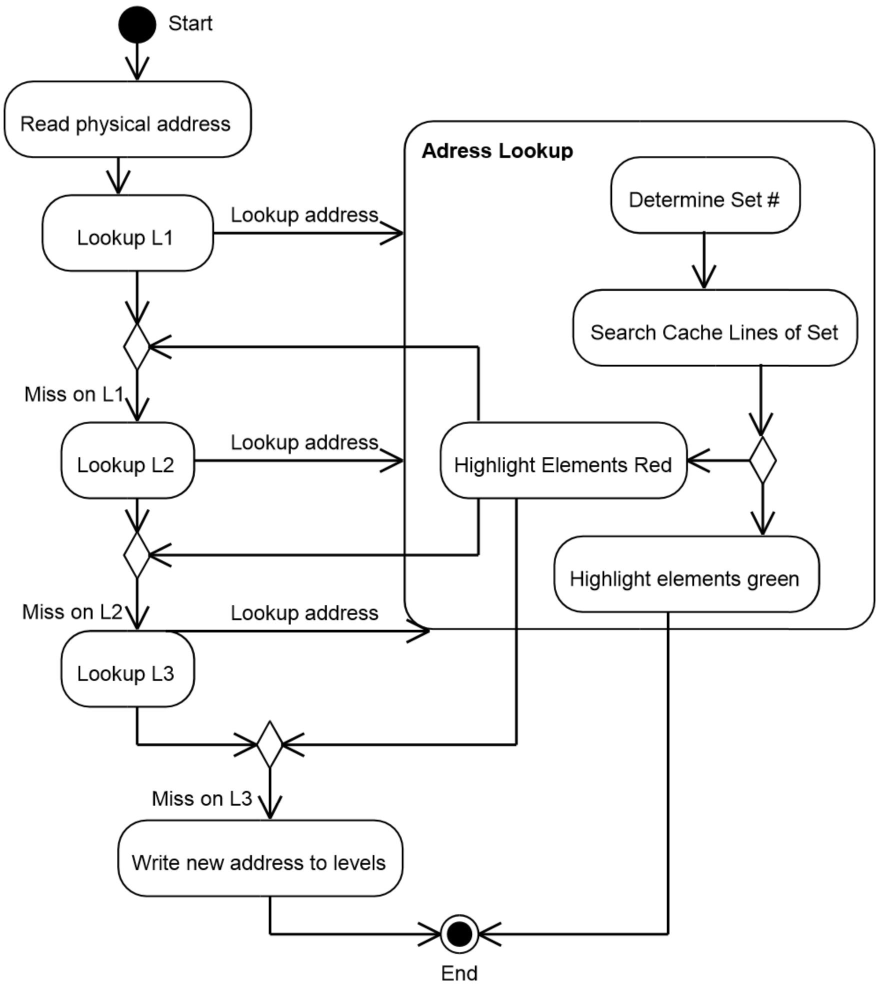
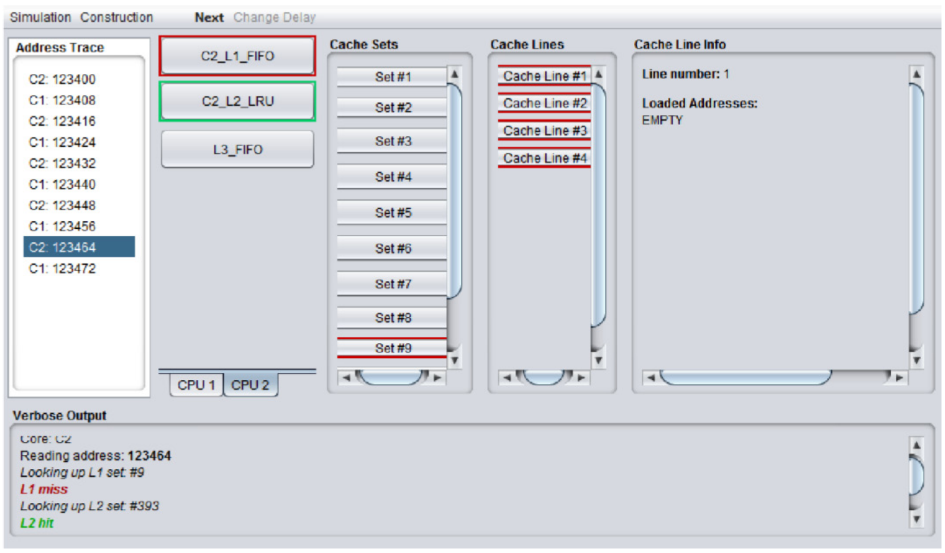


Figure 8. Activity diagram of the steps that update the visual representation



C2: 123400
C1: 123408
C2: 123416
C1: 123424
C2: 123432
C1: 123440
C2: 123448
C1: 123456
C2: 123464
C1: 123472

Line number: 1

Loaded Addresses:
EMPTY

Core: C2
Reading address: 123464
Looking up L1 set #9
L1 miss
Looking up L2 set #393
L2 hit

Figure 9. Overview of simulation mode of EDUCache simulator, L1 miss and L2 hit while reading the address 123464

More detailed explanation of the demo case study file structure is given in Appendix A.

VII. CONCLUSION AND FUTURE WORK

A new EDUCache visual simulator was developed as a support tool in the teaching process of the Computer Architecture and Organization course. It can be efficiently used by the students in learning the relevant concepts of cache memory functioning, especially in modern multichip and multicore processors.

The students have an ability to realize experiments by configuring various cache parameter independently. It enables: 1) visual understanding of the basic concepts of processing in the today's multi-core multi-layer cache architectures and organization; and 2) simulation of activities in the cache, i.e., cache misses and hits in a particular cache set and memory location for sequential and parallel execution of an algorithm. The benefits of the new developed EDUCache simulator are numerous. It has several advantages over other educational simulators in computer architecture and organization area. For example, using the EDUCache simulator, the students can interactively learn and efficiently understand the following concepts:

- The organization and processing in cache hierarchy (L1 to L3 cache levels);
- The organization and sharing of cache memory, by identifying the owners, i.e. either it is a dedicated particular cache level per core or it is shared among all cores or even shared among several CPU cores;
- The concept of ”inclusivity” between different cache levels;
- The effect of exploiting different parameters: a) sizes of the cache memory, b) the n-way cache set associativity, c) the cache line sizes; and
- The effect of choosing various cache replacement policy, being able to test various cache replacement policies per different cache levels.

The overall performance benefit of introducing the new EDUCache simulator is: 1) to enable the students learn the fundamentals of computer architecture and organization, 2) to realize experiments on performance engineering of software systems. The expected impact is that the students will easily understand the importance of computer architecture and increase their willingness to learn hardware based courses beside software based.

The usage of EDUCache simulator is not limited to the Computer Architecture and Organization course, but we plan to use in the Parallel and Distributed Computing course in the next semester. We observe that it can be used for learning hardware courses in general, due to its features and user interface. The final impact is increased student's willingness on hardware related courses and better and faster learning the relevant topics. We plan to use EDUCache simulator for further research and for example, find the optimal hardware platform to maximize the speed and speedup of cache intensive algorithms for sequential and parallel execution. Since it is initially developed for students, we plan to enable the next EDUCache version to be a free tool realized as cloud solution.

APPENDIX

EDUCACHE INTERFACES

EDUCacheSim uses a number of files for keeping data or using them as input. There are 3 types of files: Chip Configuration File (CCF), Address Trace File (ATF) and Study Case File (SCF).

A. CCF File

The CCF is the product of design mode and it is used as an input in simulation mode. It contains the information about the created architecture for a chip, by defining the number of cores per chip, cache level instances chosen for each core and the relationship between them. The file has an XML structure which makes the simulator interoperable with other similar systems. The root of the file is a Configuration tag that contains two children, CacheLevels and CacheCores. CacheLevels has 3 or more children (at least one child for every type of cache level). These children have the tag CacheLevel and each child must have the following items:

- UID - the id of the level instance
- Level - the type of cache level (1, 2 or 3)
- RP - replacement policy
- Size - the size in bytes

- LWidth - the line width in bytes
- The first level CacheCores tag has at least one child, representing a single core. A Core tag contains 6 items:
- UID - id of the core
- L1 - UID of a L1 instance • L2 - UID of a L2 instance
- L3 - UID of a L3 instance
- L1InclL2 - true if L1 is inclusive with L2
- L2InclL3 - true if L2 is inclusive with L3

Listing 1 shows an exact structure of a cache configuration file as it would look in XML.

LISTING 1: XML STRUCTURE OF A CCF

```
<Configuration >
<CacheLevels>
              <CacheLevel>
<UID>id  of            instance </UID>
<Level >[1 ,2 ,3] </ Level>
<RP>[FIFO ,LRU,BPLRU]</RP>
                            <Size>number       in           bytes </Size>
                            <LWidth>l i n e    width        in           bytes
                            </LWidth>
              </CacheLevel>
              . . .
              <CacheLevel>
                                 . . .
              </CacheLevel>
</CacheLevels>
<CacheCores>
              <Core>
<UID>UID of          core </UID>
<L1>UID of L1</L1>
<L2>UID of L2</L2>
<L3>UID of L3</L3>
                            <L1InclL2 >[true , f a l s e ]</ L1InclL2>
                            <L2InclL3 >[true , f a l s e ]</ L2InclL3>
              </Core>
              . . .
              <Core> . . .            </Core>
</CacheCores> </Configuration >
```

B. ATF File

The ATF file has a fairly simple structure as shown in Listing 2. It has a number of lines where each line consists of two comma separated values. The first value is a UID of a core created previously in some other CCF File and is used to show which core should read the following address. The second value is a physical address of a data element in main memory that the core will try to use. Commentary can be added at the beginning of the file, each line beginning with a '%' character.

LISTING 2: EXAMPLE OF AN ATF FILE WITH COMMENTARY

```
    \%Sample address t r a c e f i l e , assuming cores C1 and C2
C1 ,   123392
C2 ,   123400
C1 ,   123408
C2 ,   123416
```

C. SCF File

A SCF File contains the data about activities designed for the student to observe the working of the simulator on a particular chip configuration and trace file. it also has a XML structure as CCF File. The root of the file is a *StudyCase* node. The root node has 5 direct children:

- Title - of the study case
- Goal - what the students should learn
- Activities - description of steps to take
- ChipConfig - URI to a chip configuration file
- AddressTrace - URI to address trace file

The Activities child is the only complex element. It is consisted of a list of children nodes named Activity. Each activity node has two children. The first one is Name, for that step, and the second one is Requirement or what the students should observe. Listing 3 shows the structure of the SCF Files.

LISTING 3: XML STRUCTURE OF A SCF FILE

```
<StudyCase>
     <Title >String </ Title >
     <Goal>String </Goal>
     <A c t i v i t i e s >
             <Activity >
<Name>String </Name>
                           <Requirement>String </Requirement>
             </ Activity >
                           . . .
<Activity >           . . .          </ Activity >
     </ A c t i v i t i e s >
     <ChipConfig>URI</ChipConfig>
     <AddressTrace>URI</AddressTrace>
</StudyCase>
```

D. Simulator Output File

The Simulator Output File is generated while the simulation is running. It is basically a dump file for the verbose output log shown in the Verbose Output pane. During the simulation the lookups to the cache are explained in a readable form. After the simulation ends the statistics gathered are appended at the beginning of the file so that a student seeking only this information does not have to scroll through the whole output.

ACKNOWLEDGMENT

This work was partially financed by the Faculty of Computer Science and Engineering at the "Ss. Cyril and Methodius" University, Macedonia.

REFERENCES

[1] 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. http://dx.doi.org/10.1145/1124706.1121482

[2] M. Stojcev, I. Milentijevic, D. Kehagias, R. Drechsler, and M. Gusev, "Computer architecture core of knowledge for computer science studies," *Cyprus Computer Society Journal*, vol. 5, no. 4, pp. 39–42, 2003.

[3] M. Stolikj, S. Ristov, and N. Ackovska, "Challenging students software skills to learn hardware based courses," in *Information Technology Interfaces (ITI), Proc. of the 33rd Int. Conf. on*, 2011, pp. 339 –344.

[4] S. Ristov, M. Stolikj, and N. Ackovska, "Awakening curiosity - hardware education for computer science students," in *MIPRO, 2011 Proc. of the 34th Int. Convention, IEEE Conf. Publications*, 2011, pp. 1275 –1280.

[5] X. Liang, "A survey of hands-on assignments and projects in undergraduate computer architecture courses," in *Advances in Com-*

puter and Information Sciences and Engineering, T. Sobh, Ed. Springer Netherlands, 2008, pp. 566–570. http://dx.doi.org/10.1007/978-1-4020-8741-7_101

[6] J. L. Hennessy and D. A. Patterson, "Computer Architecture, Fifth Edition: A Quantitative Approach," MA, USA, 2012.

[7] M. Gusev and S. Ristov, "Performance gains and drawbacks using set associative cache," *Journal of Next Generation Information Technology (JNIT)*, vol. 3, no. 3, pp. 87–98, 31 Aug 2012. http://dx.doi.org/10.4156/jnit.vol3.issue3.9

[8] 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. http://dx.doi.org/10.1109/TE.2008.930097

[9] 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. http://dx.doi.org/10.1109/TE.2011.2180530

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

[11] B. B. Fraguela, R. Doallo, and E. L. Zapata, "Automatic analytical modeling for the estimation of cache misses," in *Proc. of the Int. Conf. on Parallel Architecture and Compilation Techniques (PACT '99)*. IEEE Comp. Society, 1999, pp. 221–231.

[12] 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.

[13] 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.

[14] 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 Int. Conf. on*, 2007, pp. 37 –38.

[15] A. Misev and M. Gusev, "Visual simulator for ILP dynamic OOO processor," in *WCAE '04, Proc. of the workshop on Computer architecture education: in conduction with the 31st Int. Symposium on Computer Architecture*, E. F. Gehringer, Ed. ACM, USA, 2004, pp. 87 –92.

[16] Valgrind, "System for debugging and profiling linux programs,"

[17] [retrieved: Nov, 2012]. [Online]. Available: http://valgrind.org/

[18] D. A. Patterson and J. L. Hennessy, "Computer organization and design, forth edition: The hardware/software interface," MA, USA, 2009.

[19] 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. [Online]. Available: http: //www.acm.org/education/curricula/ComputerScience2008.pdf

[20] S. Ristov, B. Atansovski, M. Gusev, and N. Anchev, "Hands-on exercises to support computer architecture students using educache simulator," University Sts Cyril and Methodius, Faculty of Computer Science, Tech. Rep. IIT-35-2013, 2013.

[21] S. Ristov and M. Gusev, "Achieving maximum performance for matrix multiplication using set associative cache," in *The 8th Int. Conf. on, Computing Technology and Information Management (ICCM2012), IEEE Conf. Publications*, ser. ICNIT '12, vol. 2, 2012, pp. 542–547.

AUTHORS

Sasko Ristov, Blagoj Atanasovski, Marjan Gusev, Nenad Anchev are with the Faculty of Information Sciences and Computer Engineering, "Ss. Cyril and Methodius" University - Skopje. Email: {sashko.ristov, marjan.gushev}@finki.ukim.mk, blagoj.atanasovski@gmail.com, nenad_ancev@hotmail.com

This article is an extended version of a paper presented at the IEEE EDUCON20213 Conference, held from March 13th-15th, 2013, in Berlin, Germany. Submitted 16 May 2013. Published asre-submitted by the authors 26 June 2013.