EDUCache Simulator for Teaching Computer Architecture and Organization

Blagoj Atanasovski, Sasko Ristov, 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: blagoj.atanasovski@gmail.com, {sashko.ristov, marjan.gushev}@finki.ukim.mk, nenad_ancev@hotmail.com

***Abstract*—Teaching computer architecture requires a lot of effort by the instructor. Introduction of simulators can improve the teaching process and increases student willingness and easier ability to learn the material. There are many visual simulators that cover courses about computer architecture and design. In this paper we present our EDUCache simulator as a supporting tool in the process of understanding the concepts of both computer architecture and computer organization. It focuses on understanding modern multi-layer, multi-cache and multi-core multi-processors. Apart of EDUCache's features to teach the students about the fundamentals of computer architecture and organization, it can be also used for performance engineering of software systems, i.e. the students will also discover the importance of necessity of computer architecture which will increase their curiosity for hardware courses in general.**

***Index Terms*—*Cache*; *CPU*; *Education*; *Multi-processor*; *Performance*.**

I. Introduction

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

Teaching computer architecture is very difficult process and requires a lot of effort from both instructors and students [2]. It is in the first study year and it is almost always totally new course for the students. Visual simulators lighten the teaching process and significantly improve CS student interest in hardware generally [3]. Teachers must select appropriate hands-on exercises, assignments and projects. A nice survey of hands-on assignments and projects is realized in [4].

Modern multi-processors use multilayer cache memory system [5] 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 our EDUCache simulator that visually presents cache hit and miss, cache line fulfillment, cache associativity problem [6], and all 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. Section III describes the architecture of EDUCache simulator. EDUCache user interface and different working modes are described and depicted in Section IV. Several demo case studies are presented in Section V. The final Section VI is devoted on 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 of visual simulators are not designed to teach the students neither about cache memory hierachy and organizarion, 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. The authors in [11] define a fully parameterizable models applicable to n-way associative caches, but only for LRU replacement policy. Our EDUCache simulator simulates both FIFO 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 the results is more complex than our EDUCache simulator. HC-Sim simulator is also based on Pin that generate traces during runtime and simulates multiple cache configurations in one run [13].

The authors in [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-of-order 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, such as openMP.

All these simulators were not primary 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. Our 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. Description of EDUCache Simulator Features and Interfaces

This section describes the main features and interfaces of our 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 ore 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 and cache levels.

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 it is shared or dedicated 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.

IV. 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. Simulation mode offers the students to load 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. Fig. 1 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. Fig. 1 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 Fig. 1.

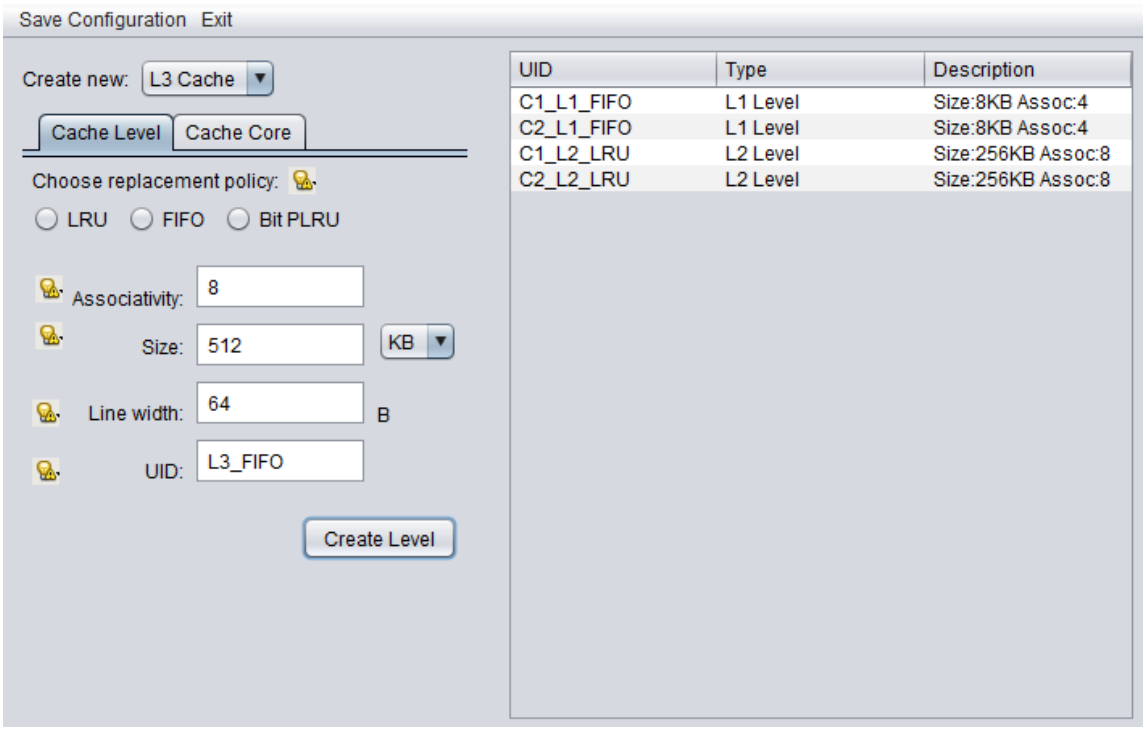

Fig. 1: Overview of design mode of EDUCache simulator - Creating L3 cache instance

After creating an 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. Fig. 2 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.

Main advantage of our EDUCache simulator compared to others is 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 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.

In order to alleviate the computer architecture learning process and the usage of the simulator, we add additional information icons next to each label. Fig. 3 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

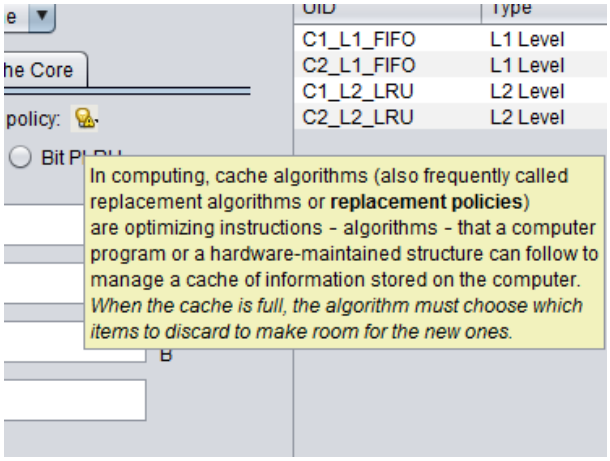

Fig. 3: A tool tip explanation box showing information about replacement policy

click on the icons or hold the mouse pointer over 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 particular cache level of particular core.

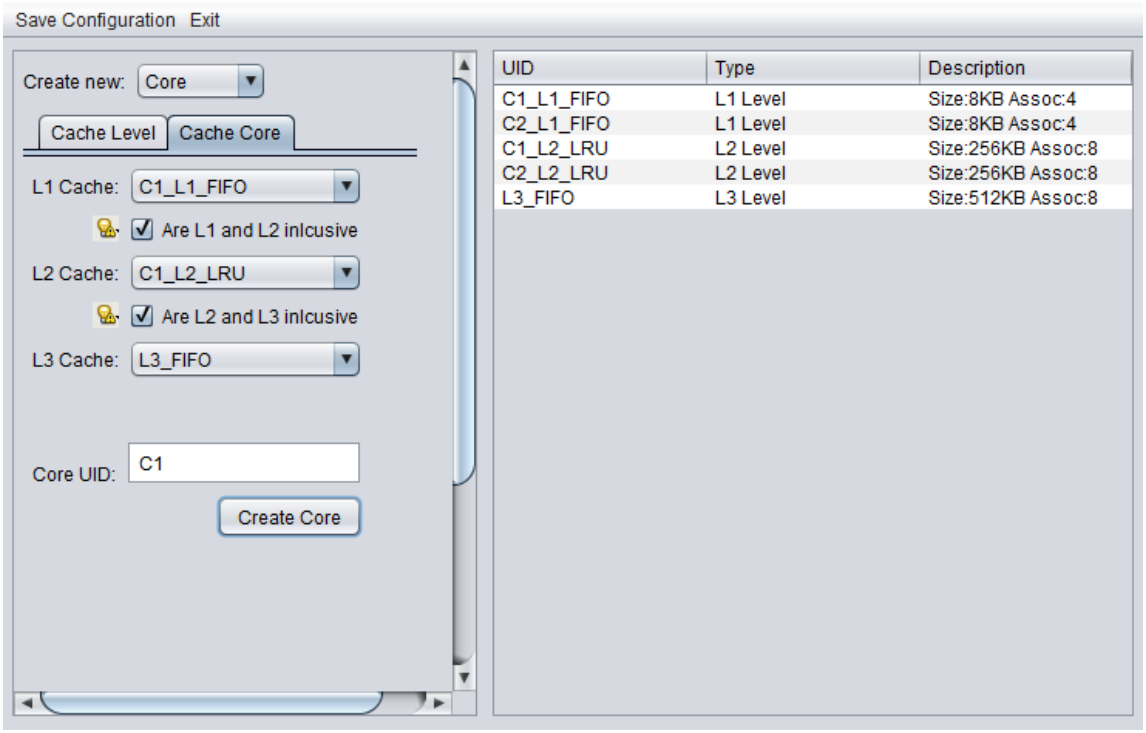

Fig. 2: Overview of design mode of EDUCache simulator - Creating a CPU core

B. Simulation Mode

If EduCache's Design mode is a mode for configuration, the Simulation mode is for working, 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.

Fig. 4 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 5.

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.

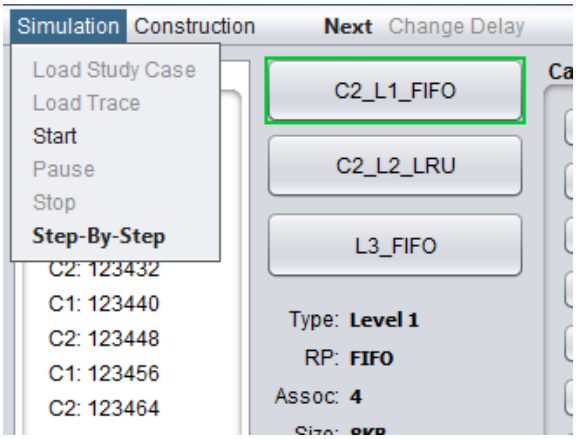

Fig. 5: Menu bar in Simulation Mode

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 are consisted 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

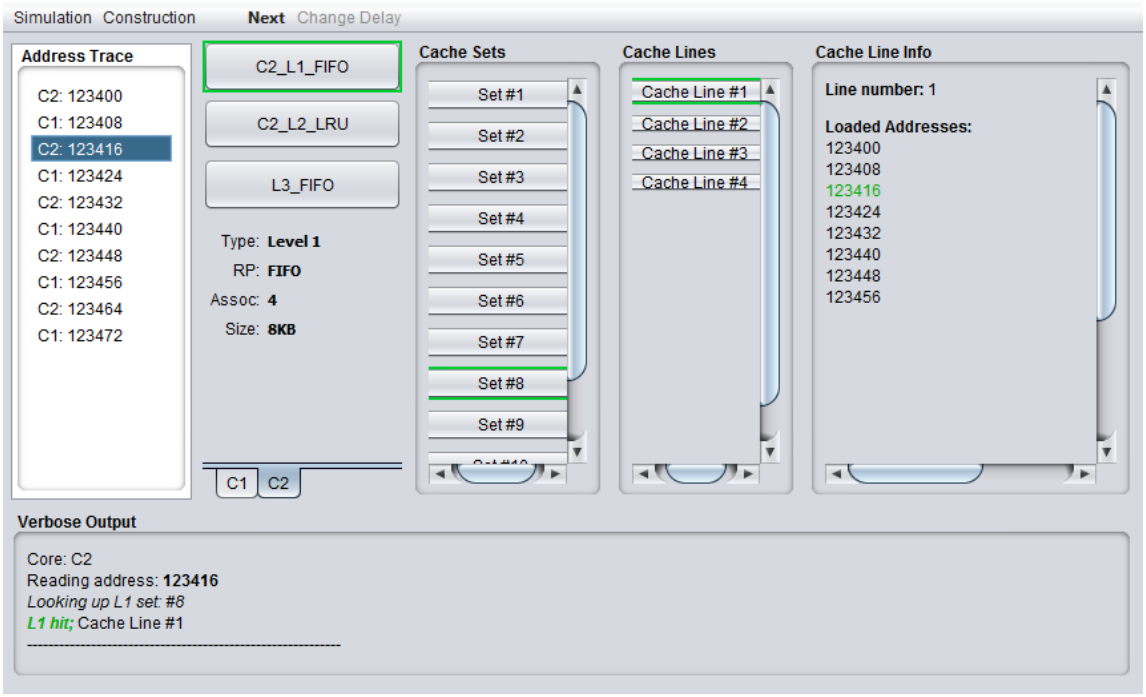

Fig. 4: Overview of Simulation mode - hit in L1 level of core C2, set #8, line #1, address 123416

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 Fig. 4. Selecting each of these tabs shows the designed cache levels in selected core presented as buttons with the UIDs 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.

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 green in the Cache Line Info pane. The cache line containing the found element in the Cache Lines Pane is highlighted also green. The corresponding cache set of the Cache Sets Pane and L1 cache level instance in the Core Pane are also highlighted 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 showed during the process in the Verbose Output Pane.

C. EDUCache Simulation

In this section we present the procedure of lookup the particular address in the caches. Fig. 6 presents the activity diagram.

In the following two sections we will describe the simulation of L1 cache hit. and L1 miss and hit in lower cache level.

1) Simulation of L1 Cache Hit: Fig. 4 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 occuer and the address in the Cache Line Info Pane, Cache Line #1, Set #8 and C2_L1_FIFO are all highlighted green.

2) Simulation of L1 Cache Miss and Hit in Lower Cache level: When a cache miss occurs in a particular cache level, then all the cache level, cache set and cache lines that were looked up are highlighted with a red. Fig. 7 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.

V. Demo Case Studies

Cases studies are special files containing a reference to both a configuration file and trace file. Their purpose is to set up a practical example with a certain goal. Using these demo case studies 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 will simulate some extremes.

A. Always Cache Hit

The first example is a trace file that will always generate cache hits for the loaded configuration. That is, always access the elements stored in particular cache level. Sorting a small array of elements is a practical algorithm that will generate always a cache hit.

B. Always Cache Misses due to Cache Capacity Problem

The other extreme example is always generating cache misses due to cache capacity problem. That is, none of the required elements can be found in a particular cache level. Accessing the elements of one column in huge-enough squared matrix in raw major will produce always cache miss. The detailed analysis for storing the matrix elements in the cache can be found in [7].

C. Always Cache Misses due to Cache Associativity Problem

Next extreme example is always generating cache misses due to cache associativity problem. That is, none of the required elements can be found in a particular cache set. The authors in [6] give a comprehensive analysis how, when and why maximum performance drawbacks appear when using set associative cache. Accessing the elements of one column for characteristic matrices in raw major will produce always cache miss since all the column elements will be stored in one cache set, and the rest of the cache will be empty.

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

VI. Conclusion and Future Work

This paper describes our EDUCache visual simulator that can be used for teaching in details the undergraduate computer architecture students the CPU cache memory architecture and organization. It allows the students to comprehend todays multi-core multi-layer cache architectures and organization configuring each cache parameter independently. It simulates cache misses and hits in particular cache set and memory location for sequential and parallel execution of particular algorithm.

Our EDUCache simulator has several advantages over other educational simulators in computer architecture and organization area. The students can interactively learn:

- The hierarchy in cache levels (L1 to L3);
- Cache owners, i.e. either particular cache level is dedicated per core or shared among all cores or even shared among several CPU cores;
- The ”inclusivity” between different cache levels, i.e. inclusive or exclusive;
- The size of the cache memory, the n-way cache set associativity, cache line sizes; and
- Cache replacement policy, with ability to have different cache replacement policies per different cache levels.

The most important benefit of our EDUCache simulator is its ability not only to teach the students about the fundamentals of computer architecture and organization, but also performance engineering of software systems, i.e. they will discover the importance of necessity of computer architecture. We believe that it will increase their willingness to teach a hardware based courses beside software based.

Our EDUCache simulator is newly developed and has not been implemented yet in the classes. We plan to introduce our simulator in courses 1) computer architecture and organization and 2) parallel and distributed processing next semester and to survey the students about its features, user interface and the positive impact to student willingness for learning hardware courses in general.

We will use EDUCache simulator in our further research to determine the optimal hardware platform to maximize the speed and speedup of cache intensive algorithms for sequential and parallel execution.

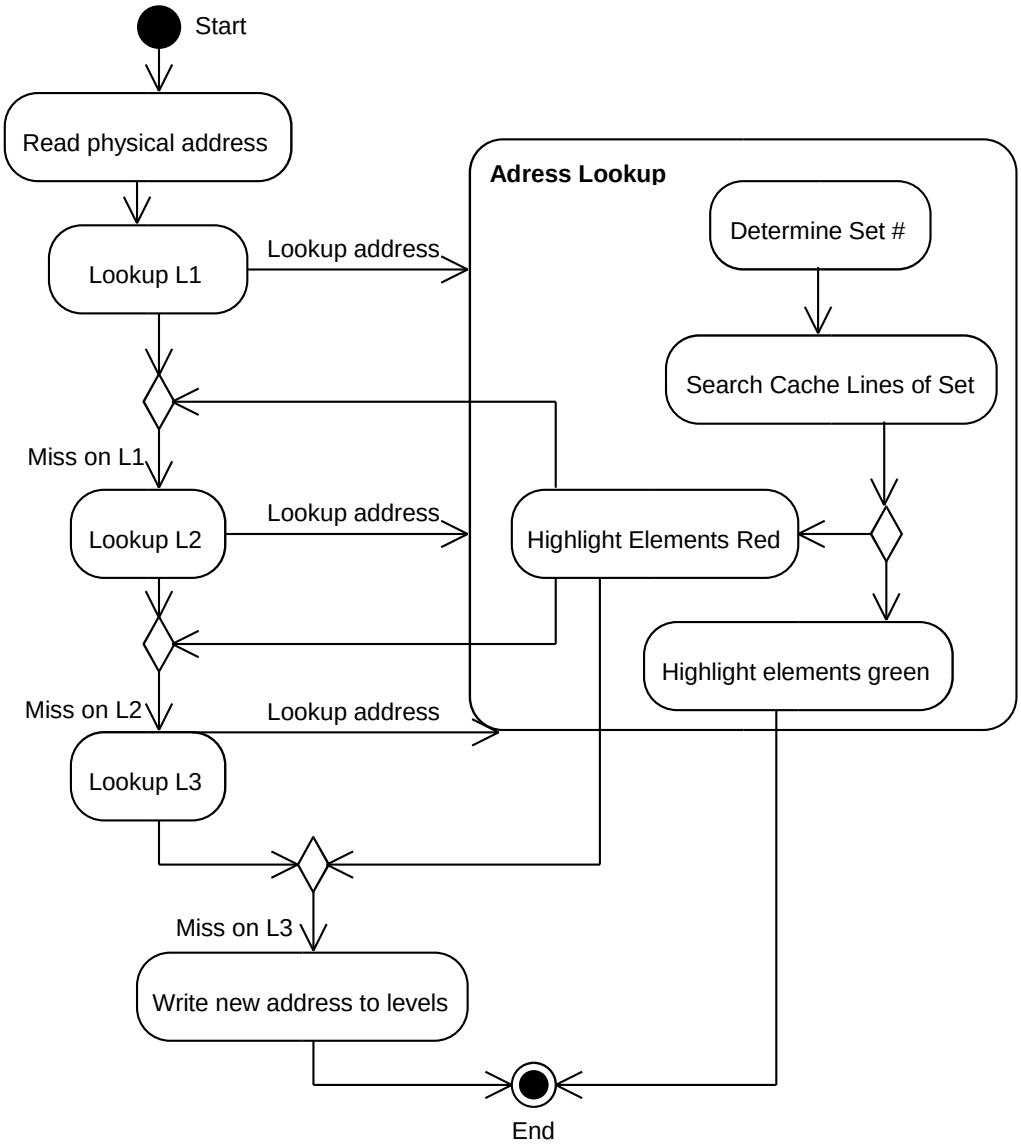

Fig. 6: Activity diagram of the steps that update the visual representation

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 architecture created for a chip. It defines the number of cores per chip, describes each of the cache level instances chosen for each core and gives 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

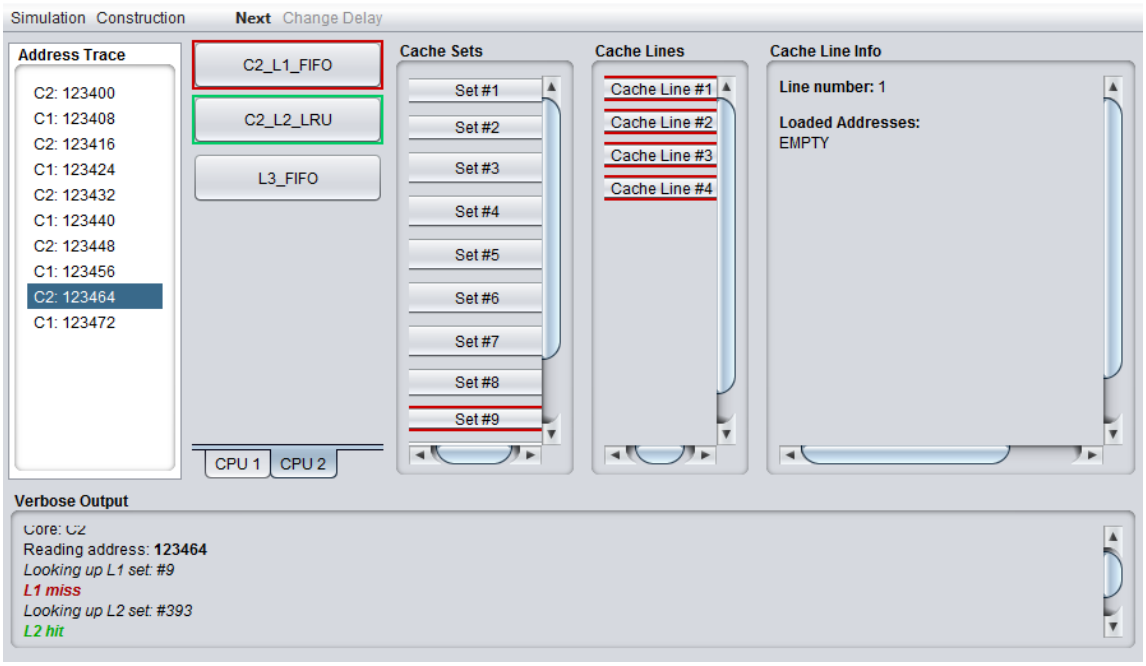

Fig. 7: Overview of simulation mode of EDUCache simulator, L1 miss and L2 hit while reading the address 123464

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>line 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,false]</L1InclL2>
                <L2InclL3>[true,false]</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 trace file, 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>
        <Activities>
                <Activity>
                        <Name>String </Name>
                        <Requirement>String
                                </Requirement>
                </Activity>
                ...
                <Activity>
                        ...
                </Activity>
        </Activities>
        <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.

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.

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

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

[4] X. Liang, "A survey of hands-on assignments and projects in undergraduate computer architecture courses," in *Advances in Computer and Information Sciences and Engineering*, T. Sobh, Ed. Springer Netherlands, 2008, pp. 566–570.

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

[6] 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.

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

[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.

[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.

[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 *Proceedings of the International Conference on Parallel Architecture and Compilation Techniques*, ser. 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 on Modeling, Benchmarking and Simulation (MoBS), co-located with ISCA'2008*, 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 *Proceedings of the 7-th IEEE/ACM/IFIP International conference on HW/SW codesign and system synthesis*, ser. CODES+ISSS '11. New York, NY, 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, 2007. MSE '07. IEEE International Conference on*, june 2007, pp. 37 –38.

[15] A. Misev and M. Gusev, "Visual simulator for ilp dynamic ooo processor," in *WCAE '04, Proceedings of the 2004 workshop on Computer architecture education: held in conduction with the 31st International Symposium on Computer Architecture*, E. F. Gehringer, Ed. ACM New York, NY, USA, june 2004, pp. 87 –92.

[16] Valgrind, "System for debugging and profiling linux programs," [retrieved: Nov, 2012]. [Online]. Available: http://valgrind.org/