MMCacheSim: A Highly Configurable Matrix Multiplication Cache Simulator

Blagoj Atanasovski, Sasko Ristov, Marjan Gusev, and Nenad Anchev

Ss. Cyril and Methodious University, Faculty of Information Sciences and Computer Engineering,
Rugjer Boshkovikj 16, 1000 Skoipje, Macedonia
blagoj.atanasovski@gmail.com, sashko.ristov@finki.ukim.mk,
marjan.gushev@finki.ukim.mk, nenad_ancev@hotmail.com

Abstract. Memory access is the bottleneck of all computations. CPU cache is introduced to speed up accessing reused and local data. Matrix multiplication is the most common representative of many linear algebra algorithms which performance directly depends of the cache. Many cache parameters exist and impact the overall computing performance such as cache type, line, size, level, associativity, and replacement policy. Therefore an optimal architecture to execute certain compute and memory intensive algorithm is desirable in most applications.
We have developed MMCacheSim simulator to predict matrix multiplication performance on particular existing or non-existing multiprocessor. MMCacheSim simulates the execution time and number of cache misses that matrix multiplication algorithm performs with particular matrix size and element size executing on processor with different cache size, line, level associativity, and replacement policy.

Keywords: CPU Cache, Multiprocessor, HPC, Simulation

1 Introduction

Basic computer system is built on the Von Neumann concept (or Eckert-Mauchly as recently recognized) with CPU, main memory and bus. The problem of matching the speed of the instruction execution with the speed of fetching and storing the data / instruction degrades the overall performance of the computer system. Modern multiprocessors use multilayer cache memory system [9] to balance the gap between CPU and main memory and to speedup data access. The *cache size* is one of the most important cache parameters since larger caches reduce miss rates, but unfortunately, require greater data access times.

The matrix multiplication algorithm provides similar performance (speed) in the same cache region [14]. n-way set associative caches produce huge performance drawbacks for cache intensive algorithms regardless of cache size [7]. *Cache line* speeds up the time locality, i.e. if sometimes a particular memory location is referenced, then it is likely that near or even the same location will

be referenced again in the near future. The decision which cache line to be replaced if all the cache lines are fulfilled in the particular set depends on *cache replacement policy*.

All these cache parameters impact the algorithm overall performance and it is difficult to select the cache with optimal parameters for particular algorithm. Even more, the same algorithm behaves differently for different input size data. Applications provide better performance when they are executed on flexible cache with reconfigurability [17]. Using a proper simulators to predict the algorithm performance can save time and wasted money for unnecessary hardware. They can be used to measure the performance of new proposed schemes [1]. The authors in [2] propose techniques to predict the performance impact using hybrid analytical models. The authors in [19] propose a technique to overcome inter-thread cache conflict misses on shared cache and develop a highly configurable multi-core cache contention MCCCSim simulator that reproduces parallel instruction execution. A predictive model is proposed in [18] to allow fast and accurate estimation of system performance degradation also due to shared cache contention in parallel execution. The authors in [5] propose a statistical cache model Statstack that models a fully associative cache with LRU replacement policy and compared the results with the traditional cache simulator.

In this paper we present a trace driven simulation based MMCacheSim simulator that takes a list of memory addresses that represent the calls to main memory and tracks the changes in the cache. An overview of several existing cache simulators is presented in Section 2. The rest of the paper is organized as follows: In Section 3 we describe the MMCAcheSim architecture and design. Section 4 describes the real and simulated experiment environments and Section 5 presents the results of the simulation and experiments. Section 6 is devoted to conclusion and future work.

2 Related Work

This section presents different purpose cache simulators that we found in the literature. Dinero IV is the cache simulator that simulates a memory hierarchy with various caches [4]. A DEW strategy [8] speeds up the simulation of multiple combinations of cache parameters. It simulates only FIFO replacement policy. The authors in [6] define a fully parameterizable models applicable to n-way associative caches, but only for LRU replacement policy. Our MMCacheSim simulates both FIFO and LRU cache replacement policies for all cache levels.

The authors in [10] propose a CMP$im simulator based on the Pin binary instrumentation tool. It is a better simulator offering multi core support and data gathering for all levels of the cache. However, the capturing the results is more complex than our MMCacheSim. HC-Sim is also based on Pin that generate traces during runtime and simulates multiple cache configurations in one run [2]. An on-line cache simulation using a retargetable application specific instruction set simulator is provided in [13]. CMPSched$im evaluates the interaction of operating system and chip multiprocessor architectures [11].

Simulators can be also used in the teaching process. Hardware courses in software oriented curriculum require a lot of effort, both from instructors and students [16]. The authors in [15] using visual simulators achieved significant improvements in grade distribution and computer science student interest in hardware. Visual EduMIPS64 helps teachers to better present the specific topics of computer architecture and also help students to learn easier [12].

In this paper we present our MMCacheSim simulator and analyze if a successful prediction of cache performance can be achieved by simulating the execution of an algorithm and measuring the number of misses on different levels of CPU cache. We build a model that can be easily configured to represent different types of cache architectures with different replacement policies. A series of experiments were performed for execution of dense matrix multiplication algorithm varying matrix sizes on real world implementations and simulation with same parameters for the CPU cache architecture.

3 MMCacheSim Simulator

This section presents the MMCacheSimulator architecture, design and class diagram, and briefly describes its inputs and outputs. The MMCacheSim simulator is implemented as a set of Java classes, each for different CPU cache parameter:

- *Cache Line* - Represents a single cache line. It is initialized with the size of the cache line, the size of the elements saved inside it, and the address of the first element saved inside. Contains methods for writing new elements in the cache line and checking if an element is in the cache line;
- *Cache Set* - Represents a collection of cache lines available for both LRU and FIFO implementations as cache replacement policies. It is initialized with the associativity and line size. Contains methods for writing an address inside the cache and with it replacing the obsolete one according to the chosen replacing policy, checking whether an address is inside the given set;
- *L1, L2, L3 Cache Levels* - The actual cache memory, also available as LRU and FIFO implementations initialized with the size, associativity and the cache line size. Contains the cache sets, the data about misses and hits made on that particular level and methods for reading from and writing to the level;
- *Processor Core* - As a real processor core would have access to the cache. Several cores may share same cache structures. The simulated model of a core is initialized with instances of cache levels, by giving different cores the same instance of a cache level we simulate sharing. A cache core has only method to read a data element. If the element is not found in the cache levels a cache miss is recorded;

Figure 1 depicts the class diagram of the LRUCore class. The *CPU Core* is the class that contains the three Cache Levels. It contains the fields to measure the number of cache misses and hits on each level for the memory calls that go through particular core. The *Cache Levels* classes also contain fields about the

number of hits and misses they generated. Since a cache can be shared among several CPU cores (mostly L3 cache), *Cache Level* also posses the information about cache misses and hits per particular CPU core.

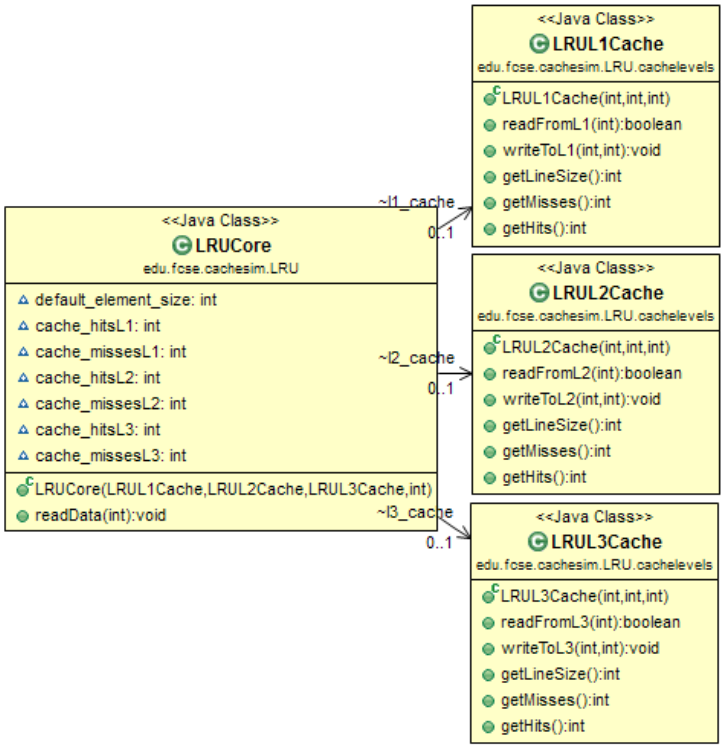

Fig. 1. MMCacheSim class diagram of a LRU Core

readData(memoryAddress) is the only method that is used during the simulation which goes through the *Cache Levels* searching if the required memory address is present in some of them going from the lowest to the highest. A sample code for inclusive caches is given in Appendix. This method also contains the logic that specifies the cache inclusivity. The *readFromLx* methods (where x denotes the cache level) in the *Cache Level* classes return a Boolean indicating whether the element is already stored in that *Cache Level*.

Figure 2 depicts the class diagram of the particular level LRUCache class. The other two classes *CacheSetLRU* and *CacheLine* contains the necessary information about particular cache level associativity.

The MMCacheSim simulates execution of the simple dense matrix multiplication algorithm. The simulation does not take into account the time required for arithmetic operations and memory writes because we are looking for the

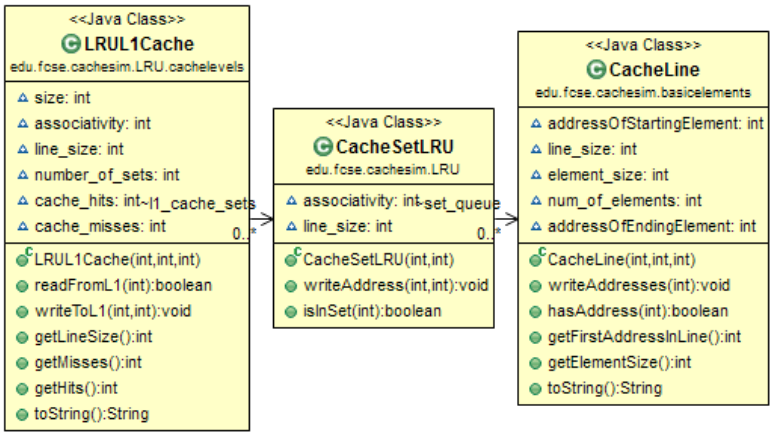

Fig. 2. MMCacheSim class diagram of the particular level LRUCache Class

effect that the cache produces when the same data is accessed multiple times and the speedup that can be gained when parallelizing the execution. The input for MMCacheSim is number of cores, cache levels, shared / dedicated cache per core, cache line, cache size, and cache replacement policy for each cache as input parameters. It returns the average clock cycles for cache hit per each cache level and cache miss for last level cache. It also measures the total clock cycles.

4 Experiment Environment

The experiments are performed on the real multiprocessors with totally different cache architectures. The first multiprocessor consists of 2 chips Intel(tm) Xeon(tm) CPU X5680 @ 3.33GHz and 24GB RAM. Each chip has 6 cores, each with 32 KB 8-way set associative L1 data cache dedicated per core and 256 KB 8-way set associative L2 cache dedicated per core. All 6 cores share 12 MB 16-way set associative L3 cache. The second server has one chip AMD Phenom(tm) 9950 Quad-Core Processor @ 2.6 GHz and 8 GB RAM. The multiprocessor has 4 cores, each with 64 KB 2-way set associative L1 data cache dedicated per core, and 512 KB 16-way set associative L2 cache dedicated per core. All 4 cores share 2 MB 32-way set associative L3 cache.

5 The Results of the Experiments

The first performed test is to determine the number of CPU cycles needed to access different levels of the cache in the simulated architectures. The same experimental tests are executed on both servers.

Figure 3 depicts the comparison of the simulation of matrix multiplication on a cache with FIFO replacement policy and cache parameters as Intel CPU.

The vertical axis represents the average number of memory accesses MA to each element of a matrix calculated as defined in (1). The values for total memory access cycles from the simulator are calculated as defined in [9]. The results prove the accuracy even for performance drawbacks due to cache associativity.

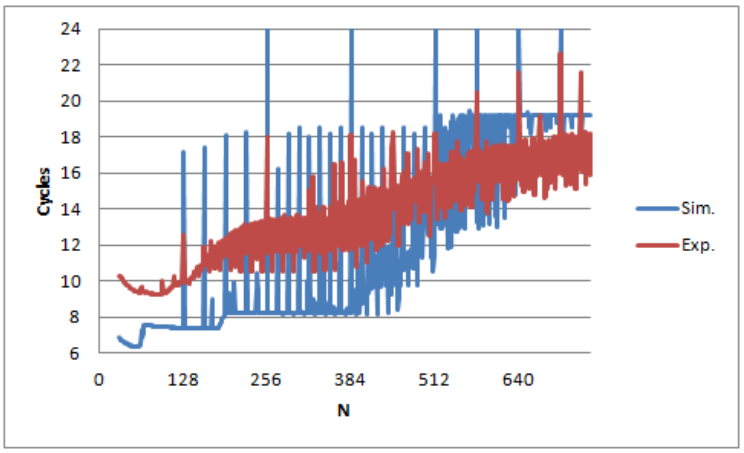

Fig. 3. Comparison CPU cycles for memory access for MMCacheSim simulation and sequential execution on Xeon server with FIFO replacement policy

$$MA = \frac{TotalMemoryAccessCycles}{N^3} \tag{1}$$

We run the simulation with LRU replacement policy for Phenom(tm) as it is AMD CPU and Opteron has LRU / PLRU replacement policy [3]. Figure 4 depicts the comparison of the number of CPU cycles used for memory access for Phenom server, LRU replacement policy, sequential execution. Because this simulation did not fit the experimental results we made two other simulations changing the replacement policy, since it was the only variable in the process. The simulation does not match neither for FIFO replacement policy as depicted in Figure 5.

The final experiment was to simulate with a new replacement policy Bit-Pseudo-LRU. Each cache line is associated with a MRU bit (most recently used) in this cache replacement policy. When the line is read the MRU bit is set to 1. When all lines in a cache set have their MRU bits set to 1, they are reset to 0. If some cache line should be replaced then the cache line in a set with the largest index that has a MRU bit 0 is replaced. Figure 6 depicts that the simulation is much closer to the experimental values. The simulation is still stepping away as the sizes of the matrices exceed the size of the cache memory. A possible explanation to the differences are: the authors in [3] show that the L3 cache

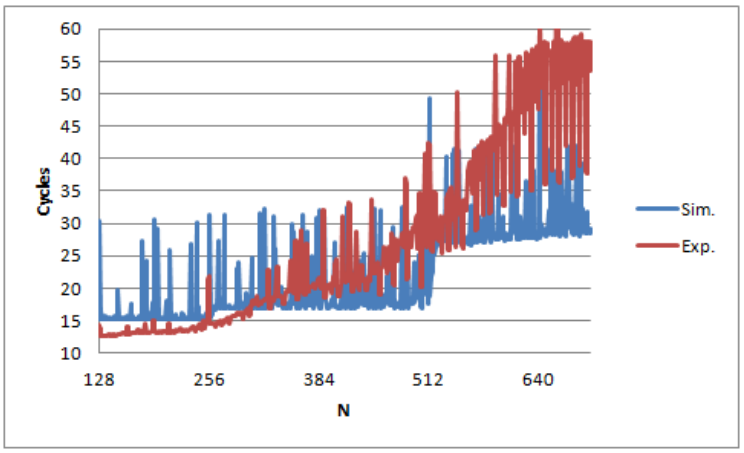

Fig. 4. Comparison of CPU cycles used for memory access for sequential execution on Phenom CPU and MMCacheSim simulation with LRU cache replacement policy

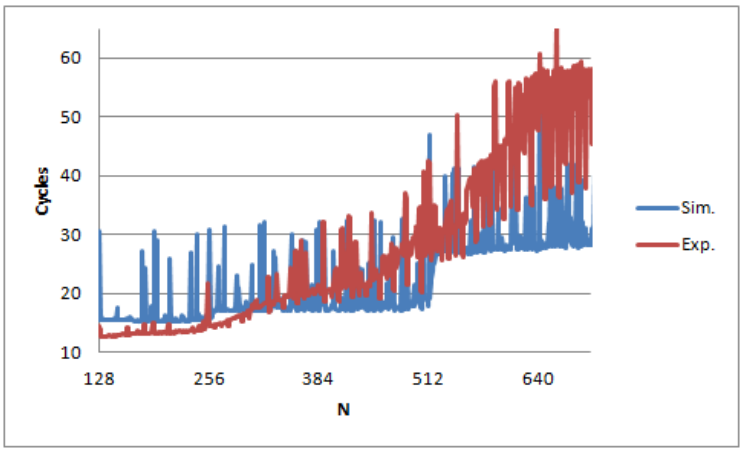

Fig. 5. Comparison of CPU cycles used for memory access for sequential execution on Phenom CPU and MMCacheSim simulation with FIFO replacement policy

at the Opteron processors uses some kind of pseudo-LRU cache replacement policy. The way of choosing the line inside the set seems to be different than the proposed Bit-PLRU policy. This is a logical explanation of the differences between simulated and experimental results, with the assumption that the cache replacement policy described in [3] is used in Phenom processors too. However this shows the ability to use the simulator not just to find favorable configurations for a certain algorithm but to research the configuration of a computer system when data for it are not available.

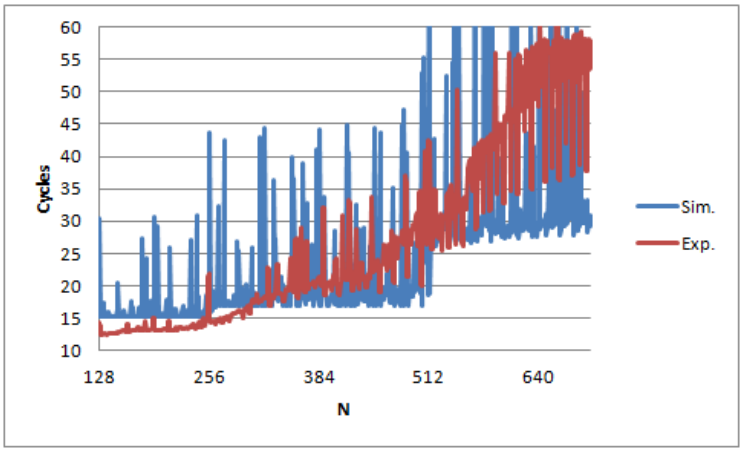

Fig. 6. Comparison of CPU cycles used for memory access for sequential execution on Phenom CPU and simulate with Bit-PLRU replacement policy

6 Conclusion and Future Work

Our MMCacheSim simulator simulates both FIFO and LRU cache replacement policies and the authors are working on implementation of other policies. All levels of cache hierarchy can be simulated. It is platform independent since the cache parameters are input parameters in the simulator.

This paper presents the simple implementation of our MMCacheSim simulator with its' features to simulate not only Matrix Multiplication algorithm execution, but any algorithm by giving a trace of memory accesses. MMCacheSim allows to change:

- The hierarchy between cache levels, to be shared between cores or dedicated;
- The inclusivity between different cache levels;
- The size of the cache memory, the associativity, cache line sizes
- Replacement policy, with ability to have different cache replacement policies per different cache levels.

The main contribution of MMCacheSim is to determine the most appropriate CPU cache architecture to achieve the best performance.

We will continue to improve MMCacheSim in order to decrease the time required for the results of simulation. Also we plan to implement additional utility classes to automate the process of building the required configurations to make the computer architecture teaching and learning process most appropriate.

References

1. An, B.S., Yum, K.H., Kim, E.J.: Scalable and efficient bounds checking for large-scale cmp environments. In: Proc. of the Int. Conf. on Par. Arch. and Compilation Techniq. pp. 193–194. PACT '11, IEEE Comp. Soc. (2011)
2. Chen, Y.T., Cong, J., Reinman, G.: 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. pp. 295–304. CODES+ISSS '11, ACM, New York, NY, USA (2011)
3. Conway, P., Kalyanasundharam, N., Donley, G., Lepak, K., Hughes, B.: Cache hierarchy and memory subsystem of the amd opteron processor. IEEE Micro 30(2), 16–29 (Mar 2010)
4. Edler, J., Hill, M.D.: Dinero iv trace-driven uniprocessor cache simulator (2012), `http://pages.cs.wisc.edu/~markhill/DineroIV/`
5. Eklov, D., Hagersten, E.: Statstack: Efficient modeling of lru caches. In: Performance Analysis of Systems Software (ISPASS), 2010 IEEE International Symposium on. pp. 55 –65 (march 2010)
6. Fraguela, B.B., Doallo, R., Zapata, E.L.: Automatic analytical modeling for the estimation of cache misses. In: Proc. of the Int. Conf. on Par. Arch. and Compilation Techniq. pp. 221–. PACT '99, IEEE Comp. Society (1999)
7. Gusev, M., Ristov, S.: Performance gains and drawbacks using set associative cache. Journal of Next Generation Information Technology (JNIT) 3(3), 87–98 (31 Aug 2012)
8. Haque, M.S., Peddersen, J., Janapsatya, A., Parameswaran, S.: Dew: a fast level 1 cache simulation approach for embedded processors with fifo replacement policy. In: Proc. of the Conf. on Design, Automation and Test in Europe. pp. 496–501. DATE '10 (2010)
9. Hennessy, J.L., Patterson, D.A.: Computer Architecture, Fifth Edition: A Quantitative Approach (2012)
10. Jaleel, A., Cohn, R.S., Luk, C.K., Jacob, B.: 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)
11. Moses, J., Aisopos, K., Jaleel, A., Iyer, R., Illikkal, R., Newell, D., Makineni, S.: Cmpsched$im: Evaluating os/cmp interaction on shared cache management. In: Performance Analysis of Systems and Software, 2009. ISPASS 2009. IEEE International Symposium on. pp. 113 –122 (april 2009)
12. Patti, D., Spadaccini, A., Palesi, M., Fazzino, F., Catania, V.: Supporting undergraduate computer architecture students using a visual mips64 cpu simulator. Education, IEEE Transactions on 55(3), 406 –411 (aug 2012)

13. Ravindran, R., Moona, R.: Retargetable cache simulation using high level processor models. Aust. Comput. Sci. Commun. 23(4), 114–121 (Jan 2001)
14. Ristov, S., Gusev, M.: Superlinear speedup for matrix multiplication. In: Information Technology Interfaces, Proceedings of the ITI 2012 34th International Conference on. pp. 499–504 (2012)
15. Ristov, S., Stolikj, M., Ackovska, N.: Awakening curiosity - hardware education for computer science students. In: MIPRO, 2011 Proc. of the 34th Int. Convention, IEEE Conference Publications. pp. 1275 –1280 (may 2011)
16. Stolikj, M., Ristov, S., Ackovska, N.: Challenging students software skills to learn hardware based courses. In: Information Technology Interfaces (ITI), Proceedings of the ITI 2011 33rd Int. Conf. on. pp. 339 –344 (june 2011)
17. Tao, J., Kunze, M., Nowak, F., Buchty, R., Karl, W.: Performance advantage of reconfigurable cache design on multicore processor systems. International Journal of Parallel Programming 36(3), 347–360 (Jun 2008)
18. Xu, C., Chen, X., Dick, R.P., Mao, Z.M.: Cache contention and application performance prediction for multi-core systems. In: ISPASS'10. pp. 76–86 (2010)
19. Zwick, M., Durkovic, M., Obermeier, F., Bamberger, W., Diepold, K.: Mcccsim - a highly configurable multi core cache contention simulator. Tech. rep., Lehrstuhl fr Datenverarbeitung, TU Mnchen (2009)

Appendix: Sample Code for *readData(memoryAddress)*

```
if (l1_cache.readFromL1(addressInMemory)) {
        cache_hitsL1++;
}
else {
        cache_missesL1++;
        if (l2_cache.readFromL2(addressInMemory)) {
                cache_hitsL2++;
        }
        else {
                cache_missesL2++;
                if (l3_cache.readFromL3(addressInMemory)) {
                        cache_hitsL3++;
                }
                else {
                        cache_missesL3++;
                        L3_cache.writeToL3(addressInMemory,
                                element_size);
                }
                l2_cache.writeToL2(addressInMemory,
                        element_size);
        }
        l1_cache.writeToL1(addressInMemory, element_size);
}
```