PERFORMANCE OF PRIVATE CACHE REPLACEMENT POLICIES FOR MULTICORE PROCESSORS

Matthew Lentz and Manoj Franklin

Department of Electrical and Computer Engineering,
University of Maryland, College Park, USA
mlentz743@gmail.com
manoj@eng.umd.edu

ABSTRACT

Multicore processors have become ubiquitous, both in general-purpose and special-purpose applications. With the number of transistors in a chip continuing to increase, the number of cores in a processor is also expected to increase. Cache replacement policy is an important design parameter of a cache hierarchy. As most of the processor designs have become multicore, there is a need to study cache replacement policies for multi-core systems. Previous studies have focused on the shared levels of the multicore cache hierarchy. In this study, we focus on the top level of the hierarchy, which bears the brunt of the memory requests emanating from each processor core. We measure the miss rates of various cache replacement policies, as the number of cores is steadily increased from 1 to 16. The study was done by modifying the publicly available SESC simulator, which models in detail a multicore processor with a multi-level cache hierarchy. Our experimental results show that for the private L1 caches, the LRU (Least Recently Used) replacement policy outperforms all of the other replacement policies. This is in contrast to what was observed in previous studies for the shared L2 cache. The results presented in this paper are useful for hardware designers to optimize their cache designs or the program codes.

KEYWORDS

Multicore, Cache memory, Replacement policies, & Performance evaluation

1. INTRODUCTION

All major high-performance microprocessor vendors are currently selling multicore chips. Future generations of these processors will undoubtedly have an increasing number of cores. Multicore architectures exploit the inherent parallelism present in programs, which is the primary means of increasing processor performance, besides decreasing the clock period and the memory latency. This high performance is based on the assumption that the memory system does not significantly stall the processor cores.

Most modern multi-core processors incorporate multiple levels of the cache hierarchy. Typically, the top level of the hierarchy consists of *private* L1 caches, as the top level needs to support the high bandwidth requirements of all the cores [1][2]. Moreover, the working sets of the threads executing in multiple cores will not cause interference in each other's L1 cache. Also, small private caches allow fast access. The subsequent levels of the cache hierarchy ─ L2 and L3 ─ are

Natarajan Meghanathan et al. (Eds) : ICCSEA, SPPR, VLSI, WiMoA, SCAI, CNSA, WeST - 2014
pp. 01–07, 2014. © CS & IT-CSCP 2014 DOI : 10.5121/csit.2014.4708

generally shared between multiple cores. Because the memory requests arriving at the L2 cache have already been filtered by the private L1 caches, the shared L2 cache can handle the bandwidth requirements. Figure 1 illustrates such a cache hierarchy for a 2-core processor.

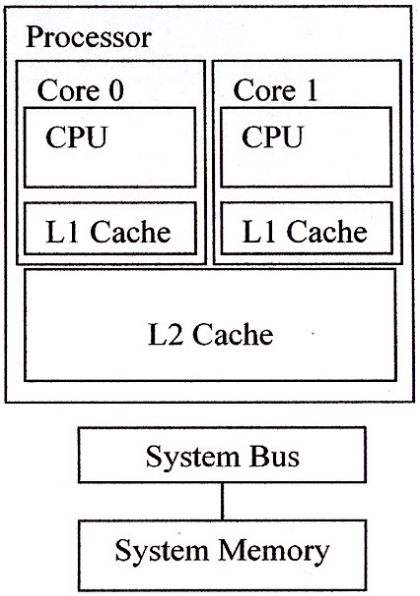


Figure 1. A 2-core processor with private L1 caches and a shared L2 cache

This paper studies the suitability of various cache replacement policies for the private L1 caches, as future processors increase the number of cores in each processor. Given the drastic change in processor hardware design, it is important to ensure that cache replacement policies are chosen such that they scale well with an ever-increasing core count. With increasing numbers of cores, caches will end up with more and more cache blocks that are replicated among the various cores as a direct result of shared variables in a multi-threaded program. The change in how the caches are used may have an impact on which cache replacement policy performs the best with increasing processor core counts.

In a set-associative cache, the cache replacement policy determines which block (in a set) to replace when there is a miss. Some of the widely used replacement policies are LRU (Least Recently Used), FIFO (First-In First-Out), and RAND (Random). Among these, LRU generally has the lowest miss rates for single-core systems, as it most closely correlates with the concept of temporal locality. However, it is also the one that has the most hardware complexity; the cache needs to maintain LRU information for each set and update this information during every memory access.

1.1. Prior Work

Cache replacement policies have been explored extensively for single-core systems over the last three decades. However, in today's multicore environment, there is a need to re-evaluate the policies, especially as the number of cores increases. Several multi-core cache studies have been done recently [3, 4, 5, 6, 7]. However, all of these studies were directed at the shared L2 cache. Because L1 cache hits do not propagate to the L2 cache, the L2 cache observes only a filtered memory access stream. The behavior of the L2 cache is therefore different from that of the private L1 caches. Therefore, there is a need to investigate cache replacement policies for the private L1 caches when running multiple threads at the same time. Although several single-core studies have

been done for the L1 cache in the past, those results may not be directly applicable to a multi-core environment where the per-core cache is smaller than that provided in a typical single-core processor. Moreover, the behavior of a private cache can be affected by how the shared cache levels below it behave. This paper studies the performance of various cache replacement policies for the L1 private caches, as the number of processor cores is varied from 1 to 16.

The rest of this paper is organized as follows. Section 2 describes the experimental framework and methodology we used for this study. Section 3 presents the experimental results and analysis. The paper concludes in Section 4 with a summary of the findings.

2. EVALUATION METHODOLOGY

We use detailed execution-based simulation to evaluate the performance of different cache replacement policies. Execution-based simulation captures subtle effects not possible with trace-based simulation. For example, different cache replacement policies might cause the execution to go along paths that are different from that followed in the trace. For evaluation, we use the publicly available SESC (SuperESCalar) simulator [8][9], a cycle-accurate simulator that models in detail a multicore processor with a multi-level cache hierarchy. SESC models different processor architectures, such as single processors, chip multi-processors, and processors-in-memory. It models a full out-of-order pipeline with branch prediction, caches, buses, and every other component of a modern processor necessary for accurate simulation.

2.1. Cache Replacement Policies

The SESC simulator incorporates the LRU and RANDOM replacement policies for the L1 caches. We modified the simulator to include additional cache policies that might prove to be more scalable when it comes to multi-threaded programs running on many cores. The implementations were tested with both single threaded and multi-threaded programs to ensure their validity. Specifically, we included the following three replacement policies:

a. Most Recently Used (MRU)
b. First In-First Out (FIFO)
c. First In-First Out with 2nd Chance (FIFO2)

LRU and FIFO are both common cache replacement policy schemes, and as such they were included in our study. MRU was included because it works well for programs that access a given amount of memory in the same way multiple times (such as looping through an array more than once), as it saves older entries that will then provide cache hits upon the following times through. FIFO2 was included because it does provide a higher percentage of cache hits than FIFO, due to the fact that it allows old entries a chance to stay in the queue through the checking of a reference bit that is reset at every replacement. However, this gain naturally comes with a higher implementation cost [2].

2.2. Cache Configuration

The cache configuration simulated includes private L1 instruction and data caches, and a shared L2 cache. Each of the L1 caches was 32kB (for both the instruction and the data cache) with an associativity of 4, while the L2 cache was 512kB with an associativity of 8. Both the L1 and the L2 caches use a 32 byte cache block size.

2.3. Benchmarks Simulated

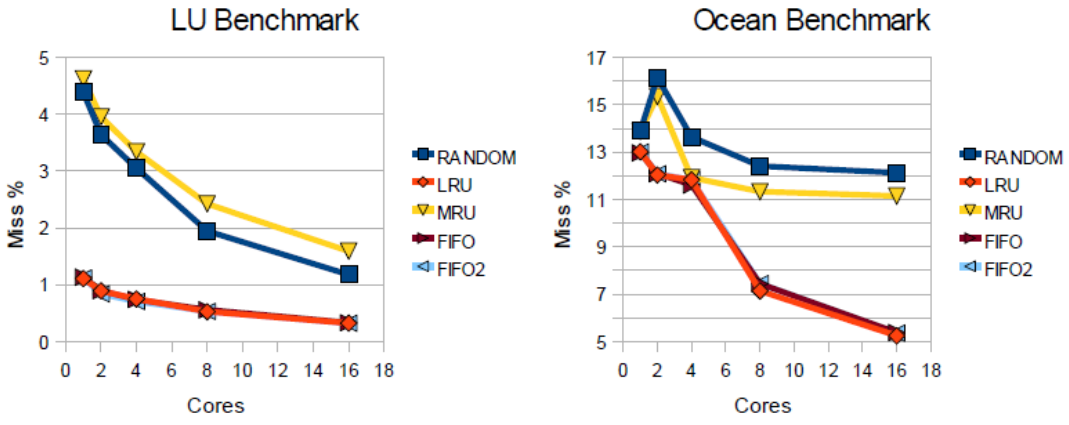
In order to analyze which cache policy performs the best as the number of cores in the processor increases, it is important to have a wide variety of benchmarks that test a number of different cache-usage patterns. The benchmarks must also be multi-threaded and scalable up to 16 threads in order to test a number of processor configurations and ensure that all available core resources are being utilized. The benchmarks that we used consist of some of the SPLASH2 benchmarks published by Stanford: FFT, LU, Radix, Ocean, and Raytrace [10]. These benchmarks are selected based on their relevance to multicore processors and cache memories. We used pre-compiled SPLASH2 benchmarks from a link on the mailing list archive [11]. Each benchmark is run by itself on the multi-core system. Core 0 runs the initialization code of the program, and then spawns the parallel threads, which run on all the cores (including core 0).

3. EXPERIMENTAL RESULTS AND ANALYSIS

For analyzing the data that was gathered from the simulations, we focus primarily on the miss ratios for the private L1 Data caches. The L1 Instruction cache miss percentages were extremely similar between the various benchmarks, regardless of the number of cores for the simulation. The differences were on the order of around just 0.01%. Given the large number of reads and writes to the L1 Instruction caches, the inclusion of the L1 Instruction cache hit/miss data would suppress differences that could be seen in the L1 Data cache hit/miss data.

3.1. Results

Figure 2 presents the miss ratios we obtained for the private L1 data caches. Data is presented as a separate graph for each benchmark. For each graph, the X axis depicts the number of cores and the Y-axis depicts the miss ratio as a percentage. Each graph shows 5 plots, corresponding to the 5 cache replacement policies investigated. Each data point represents a miss ratio calculated as a weighted average across each of the cores present, where the weight for each core is determined by the number of instructions that are executed on the core compared with the total number of instructions executed. This weight really only differs for the spawning processor, due to the extra instructions that it needs to execute in order to spawn the remaining 1-15 threads.



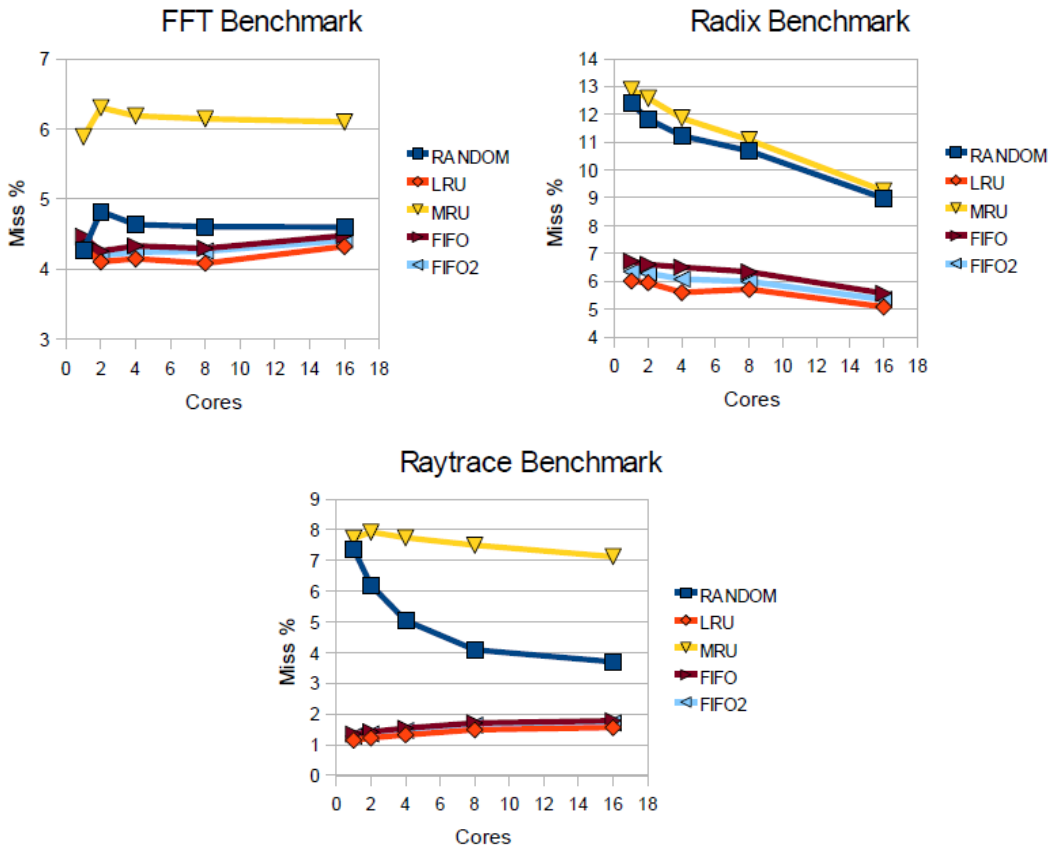


Figure 2. Miss ratios obtained for the private L1 data caches

3.2. Analysis

Overall, the results are very consistent across the benchmarks, with LRU showing the best results in terms of performance scalability, as the number of cores in the processor is increased. This was the expected outcome, given the strong performance that LRU delivers on single-core systems, due to being a close approximation for the OPT (optimum) cache replacement policy. Future work involves considering modifications to the basic LRU scheme to see if additional performance gains are possible in multicore environments.

One trend to note is how closely FIFO and FIFO2 follow LRU in terms of the miss percent as the number of cores on the processor increases. This is due to how closely both FIFO and FIFO2 can approximate an LRU implementation – especially FIFO2 due to the ability to push older, yet still actively referenced, cache lines to the back of the replacement queue: allowing it to perform replacements similar to that of an LRU implementation.

Another trend to note is the relatively poor performance of MRU across all configurations and benchmarks, with the exception of the Ocean benchmark running on a single core, where it came surprisingly close to LRU, FIFO, and FIFO2. Looking at other miss ratios for that benchmark, it can be seen that LRU, FIFO, and FIFO2 end up trending in a vastly different direction than MRU as the number of cores increases. The constant private L1 cache sizes across all cores, regardless of how many cores there are in the processor, is most likely the cause of MRU not scaling very well with increasing core counts for the Ocean benchmark. The single processor data workload

makes an MRU implementation work pretty well in comparison with the other replacement policies, but the lower per-processor data workload with increasing core counts brings about the large difference in performance.

The performance of RANDOM is generally bad, except for FFT in which case its miss ratios are comparable to that of FIFO. In general, the RANDOM policy is not a good choice for the top level of the cache hierarchy in a multicore environment.

4. CONCLUSIONS

In this paper, we explored a wide range of cache replacement policies for the top level of the memory hierarchy in a multi-core system. We varied the number of cores from 1 to 16, in order to study the scalability of each replacement policy. The analysis of the experimental results points clearly to LRU being the most scalable cache replacement policy for private L1 caches, as it consistently performed the best over all the benchmark simulations and for all core counts. This result jibes well with the strong performance that LRU had exhibited in previous-era single-core systems. FIFO and FIFO2 follow the same general trend of LRU in terms of scalability across all of the benchmarks, trailing it by a relatively small amount at any given data point. MRU did not prove to be a very useful cache replacement policy, as it only tends to work well for certain data request patterns, such as looping over the same array multiple times. The RANDOM policy, while sometimes popular in systems in which the overhead of any cache replacement calculation is impossible to deal with, also did not perform that well. Clearly, for the multicore configurations simulated and the benchmarks considered, LRU is the most scalable and FIFO/FIFO2 follow very closely with its performance trend.

ACKNOWLEDGEMENTS

The authors would like to thank everyone, just everyone! An important bug fix for the SESC simulator came from Alireza Haghdoost, a graduate student in Computer Engineering at Sharif University of Technology, who provided a fix that solved a segmentation fault that occurred when running SESC for more than two processor cores.

REFERENCES

[1] Rajeev Balasubramonian, Norman Jouppi, and Naveen Muralimanohar, (2011), "Multi-Core Cache Hierarchies", Synthesis Lectures in Computer Architecture, Morgan & Claypool Publishers.

[2] John Hennessy and David Patterson (2007) *Computer Architecture A Quantitative Approach*, 4th ed. Morgan Kaufmann.

[3] T.S.B. Sudarshan, Rahil Abbas Mir, and S. Vijayalakshmi (2004) "Highly Efficient Implementations for High Associativity Cache Memory", *Proc. 12th IEEE International Conf. on Advanced Computing and Communications*, Vol. 10, No. 5, pp87-95.

[4] Mohamed Zahran (2007) "Cache Replacement Policy Revisited," *Proc. Annual Workshop on Duplicating, Deconstructing, and Debunking (WDDD)* held in conjunction with the International Symposium on Computer Architecture (ISCA).

[5] Rahul V. Garde, Samantika Subramaniam, and Gabriel H. Loh (2008) "Deconstructing the Inefficacy of Cache Replacement Policies", *Proc. Annual Workshop on Duplicating, Deconstructing, and Debunking (WDDD)* held in conjunction with the International Symposium on Computer Architecture (ISCA).

[6] Tripti S. Warrier, B. Anupama, and Madhu Mutyam (2013) "An Application-Aware Cache Replacement Policy for Last-Level Caches", *Architecture of Computing Systems (ARCS) Lecture Notes in Computer Science* Volume 7767, pp. 207-219.

[7] S. Muthukumar and P. K. Jawahar (2014) “Sharing and Hit based Prioritizing Replacement Algorithm for Multi-Threaded Applications”, *International Journal of Computer Applications*, Vol. 90, No. 9.
[8] Jose Renau, SESC. http://sesc.sourceforge.net
[9] Pablo Ortego & Paul Sack (2004) “SESC: SuperESCalar Simulator”.
[10] Christian Bienia, Sanjeev Kumar, and Kai Li, (2008), “Parsec vs SPLASH-2: A Quantitative Comparison of two multithreaded Benchmark Suites on Chip Multiprocessors”, IEEE International Symposium on Workload Characterization, pp. 47-56.
[11] "Stanford Parallel Applications for Shared Memory." 07 Sept 2001. Web. 20 Oct 2009. <http://www-flash.stanford.edu/apps/SPLASH/>.

AUTHORS

Matthew Lentz is a graduate student in the Ph.D. program in the ECE Department at the University of Maryland at College Park, where he also obtained his B.S. in Computer Engineering. He is broadly interested in the areas of networking and systems research.

Manoj Franklin is a faculty member of the Electrical and Computer Engineering Department at the University of Maryland at College Park. His research interests are in the broad areas of computer architecture and systems.