Fixed Segmented LRU Cache Replacement Scheme with Selective Caching

Kathlene Morales and Byeong Kil Lee
Department of Electrical and Computer Engineering
University of Texas at San Antonio
San Antonio, United States
kathlene.morales@ieee.org

Abstract— **Cache replacement policies are an essential part of the memory hierarchy used to bridge the gap in speed between CPU and memory. Most of the cache replacement algorithms that can perform significantly better than LRU (Least Recently Used) replacement policy come at the cost of large hardware requirements [1][3]. With the rise of mobile computing and system-on-chip technology, these hardware costs are not acceptable. The goal of this research is to design a low cost cache replacement algorithm that achieves comparable performance to existing scheme. In this paper, we propose two enhancements to the SLRU (Segmented LRU) algorithm: (i) fixing the number of protected and probationary segments based on effective segmentation ratio with increasing the protected segments, and (ii) implementing selective caching, to achieve more effective eviction, based on preventing dead blocks from entering the cache. Our experiment results show that we achieve a speed up to 14.0% over LRU and up to 12.5% over standard SLRU.**

Keywords-cache, replacement policy, cache bypassing, segmented LRU, dead blocks.

I. INTRODUCTION

Cache replacement policies are an essential part of the memory hierarchy used to bridge the gap in speed between CPU and memory [4]. Recently, *SLRU*, which was originally proposed as a cache management scheme for disk systems, is applied to the processor cache replacement [1]. The basic idea of the SLRU replacement policy is that if a line has been accessed while occupying the cache space, it should be more difficult to be evicted than a line that has never been accessed. This scheme is implemented by adding a reference bit onto each cache line. This bit divides the cache set into two segments: *the protected segment* and *the probationary segment*. All lines entering the cache are initially part of the probationary segments. If a cache hit occurs on a line in the probationary segment, that line is then promoted to the protected segment. All victims are selected from the probationary segment using LRU replacement policy. Cache lines in the protected segment are only victimized if the probationary segment is empty. The advantage of SLRU over LRU is that it better exploits the temporal locality of lines by protecting more frequently used lines. Gao *et al* has shown that SLRU has the potential to perform better than LRU by adding enhancements such as aging, random promotion and cache bypassing [1][2].

In this paper, we propose two enhancements to the SLRU algorithm. First, in Section II, we apply the fixed number of lines in the protected and probationary segments to emphasize on protected segments. Second, in Section III, for more effective eviction, we add a selective caching method to the proposed scheme that prevents predicted dead blocks from entering the cache. We discuss the performance improvements from the proposed enhancements in Section IV, and outline the hardware costs required to implement it in Section V. Section VI summarizes the proposed work and discusses possible future work.

II. SLRU WITH FIXED SEGMENTED SIZES

Based on our observation with the SLRU algorithm, we found that often, only one or two lines occupied the protected segment. We propose a SLRU algorithm with a constant number of protected and probationary ways, called *fixed SLRU* to increase the protected segments and avoid dynamic segmentation cost. In contrast to SLRU, the proposed scheme handles the eviction of lines from the protected segment. For example, when a cache hit occurs in the probationary segment, it is promoted to the protected segment. To maintain the fixed ratio, a line from the protected segment must be evicted. This victim is chosen using LRU replacement policy.

To describe the fixed SLRU policy, we use the notation N:P, where N is the number of protected segments, and P is the number of probationary segments. N + P must be equal the associativity of the cache. The ratio between protected and probationary segments affects the cache performance. In order to find the optimum ratio, multiple simulations were performed using the SPEC 2006 benchmarks with the default cache configuration laid out by [5]: a 1K 16-way set associative cache, with 64 bit block size. All traces were simulated for 100 million instructions, after fast-forwarded 40 billion instructions.

TABLE I. BEST SEGMENTATION RATIO OF THE PROBATIONARY AND PROTECTED SEGMENTS

Benchmark	Best Segmentation Ratio
bzip2	11:5
mcf	15:1
hmmer	14:2
gcc	None
sjeng	9:7 – 15:1
namd	11:5
groamcs	14:2
milc	None
soplex	11:5
povray	8:8 – 14:2

Table 1 shows the best performing segmentation ratios for the benchmarks. Three benchmarks such as *bzip2, milc* and *namd* have their lowest CPI with the ratio of 11:5. Two benchmarks (*sjeng* and *povray*) have their lowest CPI over a range of segment ratios, which included 11:5. Three other benchmarks including *hmmer, mcf* and *gromacs* have their lowest CPI at a segment ratio over 11:5, but overall, higher segment ratios performed better for these benchmarks. For two benchmarks (*milc* and *gcc*), the segment ratio did not affect the CPI. Based on our experimental results, we determined 11:5 to be the optimum ratio, and we use it in our proposed algorithm as a fixed segmentation ratio.

III. SLRU WITH SELECTIVE CACHING

Our second enhancement to SLRU is a selective caching method which selects instructions to be bypassed based on their reference history. Jimenez *et al* has shown that bypassing "dead" blocks, which are blocks that are replaced in the cache before they are accessed, can effectively increase performance [3]. We predict that blocks that were considered dead when they last occupied the cache, will tend to be "dead" blocks the next time they occupy the cache. By bypassing these blocks, the cache does not have to needlessly evict lines that are more likely to be accessed to make room for dead blocks.

To implement this to the *fixed SLRU*, an additional one bit per cache line is required. This bit will represent whether a line has been accessed at least once while in the cache. If it has not, its tag will be updated to a table, called the bypass table. Based on our experiments, we found the bypass table to be most effective when it held a maximum of 16 tags. After a tag has been used to bypass the cache once, the line is cleared from the table. Tags were first written to any empty lines in the table. If no empty lines are available, tags are overwritten to the first line in the table. Selective caching can be implemented without increasing cache access latency by treating the bypass table as an extra way in the cache.

IV. RESULTS AND ANALYSIS

Our proposed algorithm, *fixed SLRU with selective caching*, was compared to both LRU and standard SLRU. It was tested using five floating point benchmarks and five integer benchmarks. Simulations were performed with the proposed replacement algorithm applied to the LLC (Last Level Cache). All other level of caches applied with LRU. A 1M 16-way set-associative LLC cache with 64 bit block size was used. The traces were executed for 100 million instructions after skipping an initial 40 billion instructions.

The results of our simulations can be seen in Figure 1. For single core simulations, we achieve an average speed up of 1.67% over LRU (maximum up to 14.0%), and 1.85% over standard SLRU (maximum up to 12.5%). Only one benchmark (*hmmer*) experienced a significant decrease in CPI versus LRU. This work has shown that fixed SLRU with selective caching has a potential as a replacement algorithm. Additional enhancements to this algorithm could result in a better performance replacement algorithm.

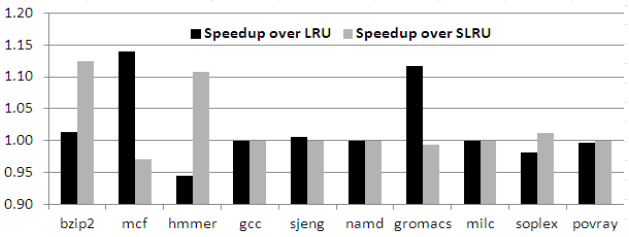

Figure 1. Simulation Results for Proposed Scheme.

V. HARDWARE COSTS

For a 16-way cache, an additional six bits per cache line is required. Four bits are required to indicate the stack position, one bit is used to distinguish between the probationary and protected segment, and one bit is used to mark whether a line has been referenced while in the cache. The bypass table we used held 16 tags. Jimenez has shown that a partial tag of 16 bits can accurately represent 64 bit data [3], so the bypass table requires 256 bits of memory. In total, single core implementation for a 16-way cache requires 96 bits per cache set and 256 bits for the bypass table.

Compared to other replacement policies, the hardware requirements for this algorithm are relatively low. For example, the dueling segmented SLRU policy proposed by Gao *et al* achieved speedups over LRU up to 8.6%, but at the cost of 102 bits per set, plus another 23.4K of additional hardware. This algorithm would only be feasible in high performance and high cost machines, but not in mobile or system-on-chip implementation. Compared to LRU, our proposed algorithm only requires an extra 32 bits per set, and 256 bits for the bypass table.

VI. CONCLUSION

Our proposed algorithm has shown its potential for high performance. With the rise of mobile technology, and system-on-chip, low cost replacement algorithms are essential. Since our proposed algorithm did not use the ideal segment ratio for every benchmark, an enhancement that adjusts the segment ratio based on performance could improve speed up.

ACKNOWLEDGEMENTS

This work was supported in part by the National Science Foundation under awards CCF-1063106.

REFERENCES

[1] H. Gao and C. Wilkerson,"A dueling segmented LRU replacement algorithm with adaptive bypassing," 1st JILP: Cache Replacement Championship, France, 2010.

[2] R. Pendse and R. Bhagavathula, "Pre-fetching with the segmented LRU algorithm," The 42nd Midwest Symposium on Circuits and Systems, 2000.

[3] D. Jimenez, "Dead block replacement and bypass with a sampling predictor," 1st JILP: Cache Replacement Championship, France, 2010.

[4] D. A. Patterson, and J. L. Hennesey, "Computer architecture: a quantitative approach," Morgan Kaufmann, 2003.

[5] 1st JILP Workshop on Computer Architecture Competitions (JWAC-1): Cache Replacement Championship. http://www.jilp.org/jwac-1/, June, 2010.