Modified Pseudo LRU Replacement Algorithm

Hassan Ghasemzadeh, Sepideh Mazrouee, Mohammad Reza Kakoee
Department of Computer Engineering
Damavand Branch
Islamic Azad University
h.ghasemzadeh@ece.ut.ac.ir, s.mazrouee@gmail.com, kakoee@cad.ece.ut.ac.ir

Abstract

Although the LRU replacement algorithm has been widely used in cache memory management, it is well-known for its inability to be easily implemented in hardware. Most of primary caches employ a simple block replacement algorithm like Pseudo LRU to avoid the disadvantages of a complex hardware design. In this paper, we propose a novel block replacement scheme, MPLRU (Modified Pseudo LRU), by exploiting second chance concept in Pseudo LRU algorithm. A comprehensive comparison is made between our algorithm and both true LRU and other conventional schemes such as FIFO, Random and Pseudo LRU. Experimental results show that MPLRU significantly reduces the number of cache misses compared to the other algorithms. Simulation results reveal that in average our algorithm can provide a value of 8.52% improvement on the miss ratio compared to the Pseudo LRU algorithm. Moreover, it provides 7.93% and 11.57% performance improvement compared to FIFO and Random replacement policies respectively.

1. Introduction

In high performance computer systems, the performance of the memory is often one of the most critical issues. Caching is mainly the simplest cost-effective way to achieve higher memory performance. The main parameters that affect the performance of the cache memories are block size, cache size, placement method, replacement scheme and write policy [1-2]. As the speed gap between the processor and memory is growing, several techniques have been proposed to compensate or tolerate memory latency. Researches have presented several replacement algorithms to improve the performance of the caches and reduce their costs [3-15].

Cache memories are commonly organized in three ways: direct-mapped, set associative and fully associative. A set associative cache with *n* blocks is called n-way set associative cache and holds *n* blocks per set. Cache designers generally use set associative cache organization because it offers a good balance between hit rates and implementation costs.

There are three conventional cache replacement algorithms: *"least recently used"* (LRU) algorithm, *"First In First Out"* (FIFO) algorithm, and *Random* replacement algorithm. The idea behind the LRU algorithm, which is one of the most popular replacement policies, is to reduce the chance of discarding information that will be referred soon. In this algorithm, the accesses to the blocks are recorded and the replaced block is the one that has not been used for the longest time. FIFO algorithm removes blocks as order as they had been brought into the cache and *Random* replacement algorithm uses a block number that is generated by a pseudo random generator circuit to determine the block that should be discarded.

This paper introduces a new cache replacement algorithm which is a modified version of the Pseudo LRU algorithm. Pseudo LRU is a good approximation of LRU which has been employed in designing several popular cache systems [3-5].

The rest of the paper is organized as follows. In Section 2, we present some of the related works. Section 3 describes how the basic Pseudo LRU algorithm works. In Section 4, we describe the organization of the proposed replacement scheme, Modified Pseudo LRU. Performance evaluation of the algorithm has been presented in Section 5, and finally we conclude the paper in Section 6.

2. Related Work

Cache replacement algorithms to reduce the number of misses have been extensively studied in the past. Many replacement policies have been proposed, but only a few of them are widely adopted in caches such as true LRU scheme and Pseudo LRU [6].

The LRU replacement is widely used for management of the cache memories to reduce the miss count. This policy exploits the principle of temporal locality and evicts the cache line which has not been used for the longest time. Recently, many efforts have been made to propose different algorithms based on LRU scheme. We classify existing algorithms into two categories: 1) the replacement algorithms which improve the LRU scheme. 2) the replacement algorithms which approximate the basic LRU policy. These categories are described in more detail in the following paragraphs.

2.1. Improved LRU Algorithms

Although LRU relies heavily on the presence of temporal locality and reduces the miss rates, only concentrating on temporal locality leads to the saturation of the miss count. Hence, several cache replacement algorithms have been developed to further reduce the miss rates [7-15]. For example, Jiang *et al* [7] have proposed a new replacement algorithm based on LRU called LIRS (Low Inter-reference Recency Set) to address a few limitations of LRU. They used the recent reuse distant as the history information of each block to make a replacement decision. Pendse *et al* [8] exploited the presence of spatial locality in the basic LRU scheme using a pre-fetching method. They showed that this variation of LRU provides a measure of 60% improvement in the miss rates for instruction cache. In [9], Lee *et al* showed that the LRFU (Least Recently Frequently Used) can gain up to a 30% performance improvement over the basic LRU policy. Another LRU-based algorithm is pipeline LRU which modifies the processor pipeline to compensate the latency involved in the algorithm [10]. The LRU-K algorithm is another derivation of LRU whose replacement decision is made by comparing access times to $(K)^{th}$, $(K-1)^{th}$, …, $(N)^{th}$ blocks.

2.2. Approximate LRU Algorithms

Although the LRU replacement algorithm produces good results regarding the miss rates, it requires large amount of hardware to keep the track of long access history in high associative caches. This hardware complexity directly affects the memory access time and leads to the implementation of caches with simple block replacement policies which are very easy to be implemented in hardware. For example, Fatemi *et al* [16] proposed an approximation to the basic LRU which has been implemented for video compression purposes. Another approximation is named Pseudo LRU which was presented in [3] and has been widely used in the Pentium processors.

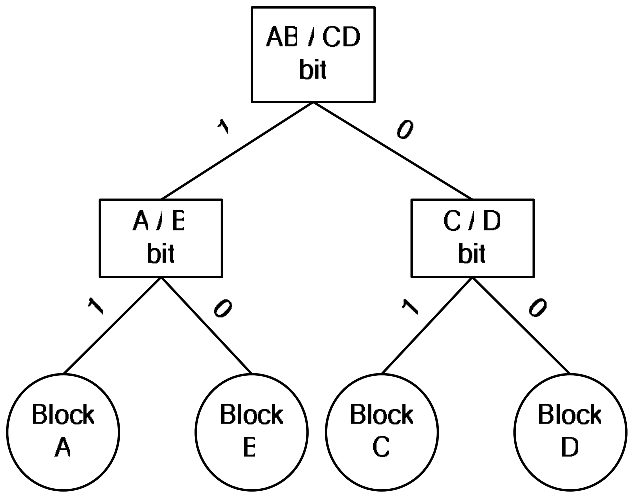

Figure 1. BPLRU algorithm, binary tree structure in a 4-block set

3. Basic Pseudo-LRU Algorithm[1]

In this Section, we briefly present the organization of BPLRU algorithm. The main idea with this algorithm is to approximate the true LRU scheme using a binary tree structure. This structure exactly needs *n-1* memory cells in an n-way set associative cache to hold the recency order of the referenced blocks. The tree which is shown in Figure 1 corresponds to a 4-way set associate cache. Assume that there are four blocks, named A, B, C and D within each cache set. The blocks are presented in the leaves and there are thee bits to keep the history information of the blocks: AB/CD, A/B and C/D bits.

On a memory reference, the value with each history bit is updated. Table 1 shows the influence of each memory access over the history bits. When a memory reference occurs to make an access to block A or block B, the AB/CD bit is set to 1, and whenever block C or block D are referenced, the AB/CD is set to 0. Similarly, the A/B bit is used to determine whether block A is accessed on a memory reference or block B. If the referenced block is A, the AB bit is set to 1 and this bit is set to 0 if an access to B occurs. The same scenario happens to C/D bit while block C or block D are referenced.

[1] For simplicity, in the rest of the paper we use "BPLRU" instead of "Basic Pseudo LRU".

Table 1. BPLRU algorithm, updating history information stored in binary tree nodes.

Referenced Block	AB/CD bit	A/B bit	C/D bit
A	1	1	no change
B	1	0	no change
C	0	no change	1
D	0	no change	0

Table 2. BPLRU algorithm, selecting least recently used block

AB/CD bit	A/B bit	C/D bit	Replaced/LRU block
0	0	0	A
0	0	1	A
0	1	0	B
0	1	1	B
1	0	0	C
1	0	1	D
1	1	0	C
1	1	1	D

It is obvious that if the referenced block does not exist in the cache, a cache miss occurs and none of the history bits change. In this situation, a replacement decision is made to find out which block must be evicted from the cache. The decision is made based on the history information stored in the history bits. Table 2 shows how a replaced block is selected by the Pseudo LRU algorithm. As an example, we explain the replacement decision when AB/CD, A/B and C/D bits have their history values of 0, 0 and 1 respectively. The Pseudo LRU algorithm assumes that neither block A nor block B has been accessed during the last memory reference because the value of the AB/CD bit is 0. The next step in making the replacement decision is to decide whether block A is the least recently referenced or block B. Since the value of the A/B bit is 0, the block that has been accessed during recent memory references is B. As a result, block A is the least recently referenced and should be evicted from the cache.

4. Modified Pseudo-LRU Algorithm[2]

The key weakness of the BPLRU cache replacement policy is that it does not have enough hysteresis to mimic ideal LRU. The binary nature of the tree means that the nodes at the top of the tree do not contain sufficient information about nodes at the bottom of the tree. A closer look at the operation of BPLRU reveals the differences between ideal LRU and BPLRU algorithms. We use a simple example to illustrate it. Assume that a cache memory holds four blocks per set named A, B, C, D. The accessed blocks are A, B, C respectively. It is obvious that block D is the least recently used because it has never been used during recent memory references. However, Pseudo LRU introduces A as the least recently used block because the last referenced block is C and BPLRU assumes that non of C or D are least recently used blocks. We made a few changes in BPLRU behavior to improve its performance.

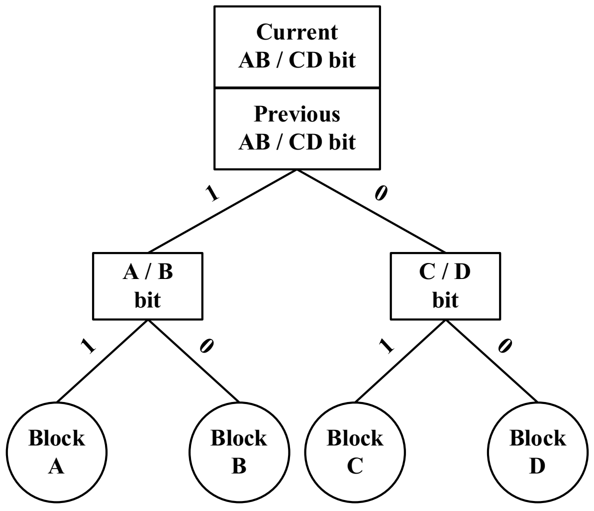

Figure 2. MPLRU algorithm, binary tree structure in a 4-block set

We classify the nodes within binary tree which keep the history of the blocks into two groups. First, the nodes which keep the information of two blocks, such as A/B bit or C/D bit. We call these nodes as Two Block Access Indicator (TBAI). Second, the nodes which keep the information of more than two blocks, such as AB/CD bit which represents the history information of four blocks. We call these nodes as Multiple Block Access Indicator (MBAI). Thus, in a 4-block set, there is only one MBAI node and there are two TBAI nodes.

We recall from Section 3 that in BPLRU each node classifies cache blocks into two groups. BPLRU makes a wrong decision whenever two blocks from different groups are accessed. For example, in Figure 1, the first node of binary tree divides four blocks into two groups: AB and CD. When the access sequence is A,C

[2] For simplicity, in the rest of the paper we use "MPLRU" instead of "Modified Pseudo LRU".

two blocks from different groups are accessed, and the replaced block is chosen by mistake. This incorrect decision is only made on MBAI nodes. The binary tree shown in Figure 1 has just one MBAI node that is the main source of mistake in selecting least recently used block.

The main idea behind our algorithm is to save the previous status of each MBAI node and make the replacement decision based on previous and current statuses. Figure 2 shows the binary tree related to a 4-way set associative cache which exploits MPLRU algorithm. The tree has two history bits for AB/CD. The first bit that keeps the status during last memory reference is called previous AB/CD bit, and the second bit that indicates current memory access is named current AB/CD bit. When a miss occurs and a block is required for replacement, we use previous bit to make a replacement decision. On a memory reference, the value with current bit is copied to previous bit and current bit is updated based on the block that is referenced now. Table 3 shows the update mechanism.

Table 3. MPLRU algorithm, updating history information stored in binary tree nodes.

Referenced Block	Current AB/CD bit	A/B bit	C/D bit
A	1	1	no change
B	1	0	no change
C	0	no change	1
D	0	no change	0

Table 4 shows a simple example to illustrate how a LRU block is selected by MPLRU algorithm and how our algorithm makes a better replacement decision than BPLRU. We assume that the cache can hold only four blocks per set. As an example, block A is accessed at times 1,6 and 8. Based on the update mechanism on BPLRU in Section 3, at time 1, block D is the least recently used block. At time 2, block D is accessed and history bits are updated. Based on values in Table 1, LRU block is B, whereas in MPLRU, block C is selected as replaced block. We mention again that in MPLRU algorithm, the replacement decision is made based on previous version of AB/CD bit instead of current value. As it is clear in Table 2, MPLRU has a closer behavior to true LRU than BPLRU. The replaced blocks introduced by BPLRU are different from true LRU at times 2, 5, 7 and 10, whereas MPLRU has different results at times 3, 4 and 8.

5. Performance Evaluation

5.1. Experiment Settings

We use trace-driven simulations with several types of workloads to evaluate the MPLRU and compare it with other algorithms. The cache simulator introduced in [17] is modified to support different types of cache replacement algorithms. We made the development of several new functions that evaluate the behavior of BPLRU and MPLRU schemes.

Table 4. Example of making replacement decision in different algorithms

Time		1	2	3	4	5	6	7	8	9	10
Referenced		A	D	B	C	B	A	D	A	B	C
LRU Block	BPLRU	D	B	C	A	D	D	B	C	C	A
	MPLRU	D	C	A	D	A	D	C	B	C	D
	True LRU	D	C	C	A	A	D	C	C	C	D

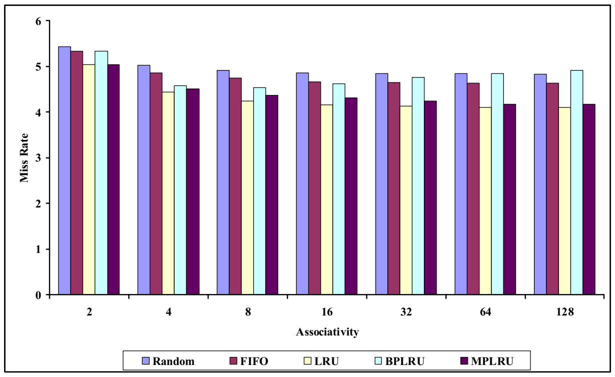

Figure 3. Miss rates versus associativity for different algorithms in an 8KB cache memory

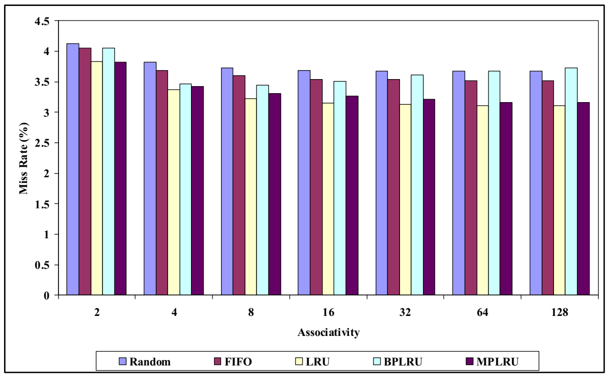

Figure 4. Miss rates versus associativity for different algorithms in a 16KB cache memory

Workloads used in this trace-driven simulation includes fifteen different kinds of program, namely *dec0, fora, forf, fsxzz, lvex, lisp, macr, memxx, mul2, mul3, pasc, ue02, spice, gcc, tex*. The three latter benchmarks include 1,000,000 references per trace file, and the other workload contains at least 10,000 references each. Both data and instruction references are collected and used for simulation. In addition to three traditional replacement algorithms, i.e., Random, FIFO and LRU, two developed replacement policies, i.e., BPLRU and MPLRU are chosen for performance comparison and evaluation.

5.2. Simulation Results

In this Section, we present the performance results of the MPLRU algorithm using workloads applied to the simulator. We compared the MPLRU approach with four other cache replacement policies. These policies are the most common replacement schemes employed in the processors, including FIFO, Random, LRU and BPLRU. For all these algorithms, we used a 32 byte block size (lines per set) in the primary unified caches and evaluated different schemes in terms of miss rates.

To test the effectiveness of our approach, we made the results for different cache sizes. Figures 3-6 show the simulation results. Through comprehensive comparison between MPLRU and other algorithms, it is shown that MPLRU outperforms all traditional schemes except true LRU algorithm. In particular, the MPLRU can provide average values of 8.52%, 7.93% and 11.57% improvement in the miss ratio over BPLRU, FIFO and Random replacement algorithms respectively. Another observation is that the MPLRU improvement over other schemes increases as the cache can hold more amounts of instructions and data.

To get insights into the superiority of MPLRU over other replacement algorithms, one noticeable point is that MPLRU operates much closer to LRU policy. It means that there is a little difference between true LRU and MPLRU based on our simulation results. We demonstrated that the true LRU algorithm operates in average 2.21% better that the MPLRU in terms of miss rates.

In summary, MPLRU performs significantly better than BPLRU, FIFO and Random replacement algorithms and does not have much distance to true LRU scheme.

5.3. Overhead Analysis

Pseudo-LRU replacement algorithm is known for its simplicity and efficiency. Comparing the time and space overhead of MPLRU and BPLRU, we show that MPLRU keeps the BPLRU merit of low overhead. The time overhead of MPLRU algorithm is *O(1)*, which is almost the same as BPLRU with a few additional operations such as those on copying previous status bit into current status bit. Extra bit for keeping a copy of each MBAI bit is additional space overhead of the MPLRU algorithm.

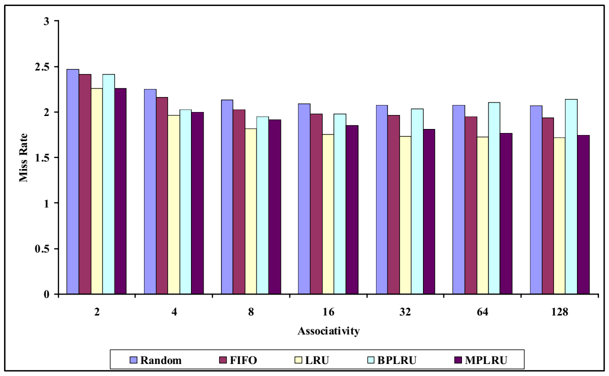

Figure 5. Miss rates versus associativity for different algorithms in a 32KB cache memory

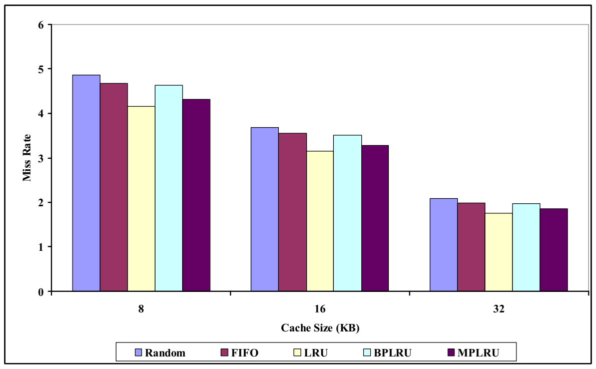

Figure 6. Miss rates versus cache size for different algorithms in 16-way set associative caches

For an n-way set-associative cache, the binary tree in BPLRU has at least *n-1* nodes and needs at least *n-1* bits. In MPLRU, there are *n/2-1* MBAI nodes and *n/2* TBAI nodes, leading to total of *n-1* nodes. To store a copy of each MBAI node, we require *n/2-1* extra bits. The total storage here is *3n/2-2* bits compared to *n-1* bits for Pseudo-LRU.

6. Conclusion

In this paper, we presented an improved version of BPLRU (Basic Pseudo LRU) block replacement algorithm called MPLRU (Modified Pseudo LRU). Results obtained using a trace-driven cache simulator for different cache configurations reveals that the MPLRU can provide significant performance improvements with respect to the traditional replacement algorithms such as FIFO, Random and Pseudo LRU.

7. References

[1] J.L. Hennessy, D.A. Patterson, "Computer Architecture Quantitative Approach", 2nd Edition, Morgan-Kaufmann Publishing Co., 1996.

[2] A.J. Smith, "Cache Memories. Computing Survey", Vol. 14, No. 4, pp. 473-530, 1982.

[3] J. Handy, The Cache Memory Book, Academic Press, San Diego, pp. 47-67, 1993.

[4] K.A. Hurd, A 600MHZ 64b PA-RISC Microprocessor, ISSCC Digest of Technical Papers, pp 94-95, February 2000.

[5] H. Ghasemzadeh and O. Fatemi, Pseudo-FIFO Architecture of LRU Replacement Algorithm, Proc. 9th IEEE International Multi Topic Conference (INMIC), December 23-25, 2005.

[6] J. Jeong and M. Dubios, Cost-Sensitive Cache Replacement Algorithms, The Ninth International Symposium on High-Performance Computer Architecture (HPCA-9'03), 2002.

[7] S. Jiang, X. Zhang, Making LRU Friendly to Weak Locality Workloads : A Novel Replacement Algorithm to Improve Buffer Cache Performance, IEEE Transactions on Computers, Vol. 54, No. 8, Aug. 2005.

[8] R. Pendse and R. Bhagavathula, Performance of LRU Replacement Algorithm with Pre-fetching, IEEE Transactions on Computers, pp. 86-89,1999.

[9] D. Lee, J. Choi and H. Choe, Implementation and Performance Evaluation of LRFU Replacement Policy, IEEE Transactions on Computers, pp. 106-111, 1997.

[10] R. Pendse, Pipeline LRU Block Replacement Algorithm, Proc. 43rd Midwest Symp. On Circuits and Systems, Lansing MI, Aug. 8-11, 2002.

[11] D. Lee, et. Al., LRFU : a spectrum of policies that subsumes the least recently used and least frequently used policies, IEEE Transaction on Computers, vol. 50, no. 12, pp. 1352-1361, 2001.

[12] A.W. Wayne and J.L. Baer, Modified LRU Policies for Improving Second-Level Behavior, proceeding 6th International Symposium on High-Performance Computer Architecture, pp. 49-60, 2000.

[13] R. Karedla, J.S. Love and B.G. Wherry, Caching Strategies to Improve Disk System Performance, IEEE Computer, pp. 38-46, March 1994.

[14] N.P. Jouppi, Improving Direct-Mapped Cache Performance by the Addition of a Small Fully-Associative Cache and Prefetch Buffers, Proc. 17th Annual International Symposium on Computer Architecture, pp. 364-373, March 1990.

[15] Yoon, J., Min, S.L., and Cho, Y., Buffer cache management: predicting the future from the past, International Symposium on Parallel Architectures, Algorithms and Networks (ISPAN'02), pp. 92-97, 2002.

[16] O. Fatemi, F. Idris and S. Panchanathan, FPGA Implementation of the LRU Algorithm for Video Compression, IEEE 1994 International Conference on Acoustics, Speech, and Signal Processing, pp. 337-344, June 1994.

[17] http://www.cs.wisc.edu/~arch/www/tools.html