Article can be accessed online at http://www.publishingindia.com

Pre-eminent Performance in a Multi-Cache Memory with all Level Replacement: An Analytical Study

Rasmi Prakash Swain*, Debabala Swain, Bijay Paikaray*****

Abstract

Cache memory is a high speed & fast memory. It can reduce the speed in between memory & processor. CPU takes the help of different level of cache memory to find the adequate data. If it not finds the data in level 1 cache. L1 then it access L2. For the assessment of data different page replacement algorithms are used. The replacement algorithms which works efficiently on L1 may not be as efficient on L2. So different access patterns are required. Cache memory works with principle called principle of locality. Whenever a page/word/block is requested from CPU, first of all it is searched on L1 if the required page is found in L1 it is a hit else a miss. When L1 is saturated and it is a miss then a block from L1 is to be evicted to create a space for the required page. Different page replacement algorithms such as LRU, LFU and FIFO are used on various pairs of cache hierarchy for result analysis. This paper motivates the new researchers to develop a new novel replacement algorithm that can be tested with performance superior to other existing algorithms.

Keywords: Multi-cache Performance, All Level Replacement, L1 & L 2 Locality.

1. Introduction

Cache memory plays the role to reduce latency between main memory and processor. Cache memory works with the principle of locality. The principle of locality can be stated as spatial or temporal. When processor requests for a block, first it looks into L1, if found then it is called a hit otherwise a miss. WhenL1 is completely mapped and a block found missed then a block from L1 needs to be evicted to create a space for the required block. Various replacement algorithms, such as LRU, FIFO, LFU etc are used to select the victim page.L1 is having better temporal locality than L2, as L2 is accessed when a miss occurs on L1.L1 holds the hot pages as usual and cold pages are placed in main memory after being taken off. Whenever a page is evicted from L1 it will be placed on L2. The nature of pages which should reside on L2 should be neither too cold nor too hot i.e. moderate. Place for hot pages is L1 and that for cold pages is in the main memory ([1],[2]). In genuine, by going through different algorithms it can be found that cache hold the hot pages but from the above discussion it is clear that this is not the requirement for L2 cache. Thus the replacement algorithm which is suitable on L1 may not be suitable on L2 [2]. Initially if any word/block/page is referenced then it will suffer with compulsory miss. If a reference suffers a miss because of saturated cache or capacity miss, then replacement algorithm will evict a word/block/page. Recently, frequency criteria's can be considered during the creation of replacement algorithms. The replacement algorithms such as Least Recently Used (LRU), First in First out (FIFO), Least Frequently Used (LFU) has been considered for different eviction of word/page/block.

* M.Tech Scholar, Dept. of Computer Science & Engg Centurion University of Technology & Management Bhubaneswar, Odisha, India. E-mail: rasmi_swn@yahoo.co.in

** Asst. Professor, Dept. of Computer Science & Engg Centurion University of Technology & Management Bhubaneswar, Odisha, India. E-mail: debabala.swain@gmail.com

*** M.Tech Scholar, Dept. of Computer Science & Engg Centurion University of Technology & Management Bhubaneswar, Odisha, India. E-mail: bijaypaikaray87@gmail.com

2. Cache Memory

Computer memory organization can be classified into different levels depending on the closeness to the processor. At the highest level are the processor registers. Next comes one or more levels of cache denoted as L1, L2, and L3 etc. Then memory, which is usually made out of a dynamic random - access memory (DRAM), comes in the next level. All of these are considered internal to the computer system. External memory, removable media such as ZIP cartridges, optical disks, and tape are also being a part of that hierarchy. As one proceed down towards the memory hierarchy, decreasing cost/bit, increasing capacity, and slower access time can be found. Since the fastest memory is more expensive, slower memory can be used for trading off access time and cost. The data and programs should be organised in memory. Especially the words which are most needed should be in fastest memory. In general, it is likely that most future accesses to main memory by the processor will be to locate recently accesses. So the cache automatically retains a copy of some of the recently used words from the DRAM. If the cache is designed properly, then most of the time the processor will request memory words that are already in the cache.

Multi-level cache hierarchies introduce three major problems in cache replacement. The first is the hiding of locality of reference by the upper cache [3]. The second is data redundancy, where blocks are saved in multiple cache levels [8]. The third is the lack of information about the blocks' attributes (e.g., their file, the application that issued the I/O request) in the lower level caches [6].

3. Related Works

3.1. Partition Based Replacement Algorithm on L1 [7]

In merge sort the reference pattern can be divided into three categories. The idea is to partition the cache in two parts P1 and P2 as shown in Fig. 1, very small part for fixed cache (P1) and a large part for variable cache (P2). In the fixed part of the cache the replacement takes place for only for selected few elements. For the first category of references, after initial misses first few elements use the fixed part cache. Similarly for the second category of reference, the same fixed part cache is used for last few references as now there will be no reference from first part. Now for the reference of third category the rest of the cache i.e. the variable part cache is used for merging the two sorted lists. Thus for L1 we have developed Partition Based Replacement (PBR-L1) Algorithm and policies like LRU, FIFO and LFU on L1 for various cache sizes.

3.2. Dual Queues Cache Replacement Algorithm Based on Sequentially Detection (CRASD) [8]

CRASD chooses the largest sequential block set to drop blocks. During replacement, the chosen set will not be the largest any longer because its blocks are continuously ejected and itssequentially decreases. However, we still evict blocks from the set rather than choosing a new one. To guarantee the continuous replacement, we use a pointer to point to the currently replaced set PSrep.

Table 1. Comparative Analysis of Page Replacement Algorithms with PBR_L1[7]

Cache size		*16*	*56*	*96*	*136*	*176*	*216*	*256*
L1	*L2*							
LRU	**LRU**	60.89	43.21	33.89	21.88	21.73	19.14	12.5
	FIFO	61.96	40.63	28.13	18.75	18.75	18.75	12.5
	LFU	61.23	43.83	34.03	21.88	21.83	19.24	12.5
FIFO	**LRU**	60.3	43.12	33.74	21.88	21.88	19.43	12.5
	FIFO	61.52	40.63	28.13	23.97	18.75	18.75	12.5
	LFU	60.79	43.75	33.84	21.88	21.88	19.63	12.5
PBR_L1	**LRU**	46.68	28.86	20.12	12.5	12.5	12.5	12.5
	FIFO	24.07	13.45	6.05	0	0	0	0
	LFU	47.02	29.25	20.17	12.5	12.5	12.5	12.5

Figure 1. Analysis of Varying List Length on L1[7]

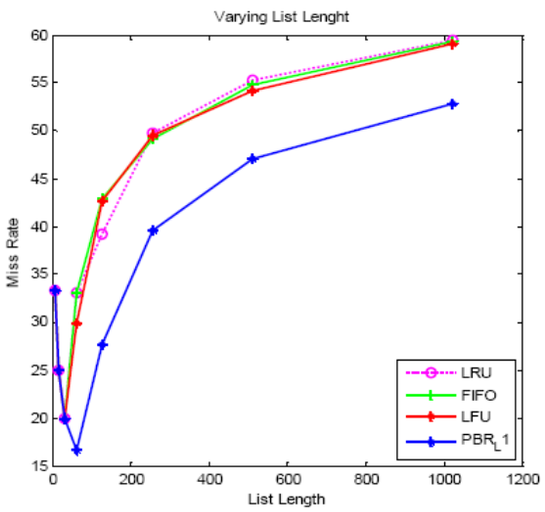


Figure 2. L1 Size: 8, L2 Varied from 16) L1-PBR-L1[7]

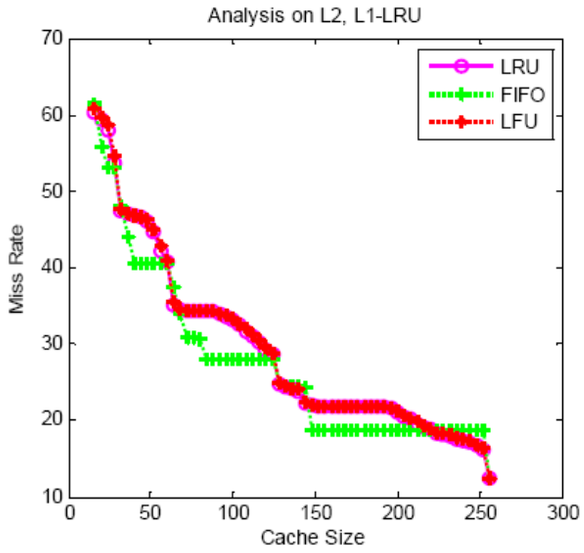


Figure 3. Effective Access Time Per Block Achieved by CRASD, ARC and LRU[8]

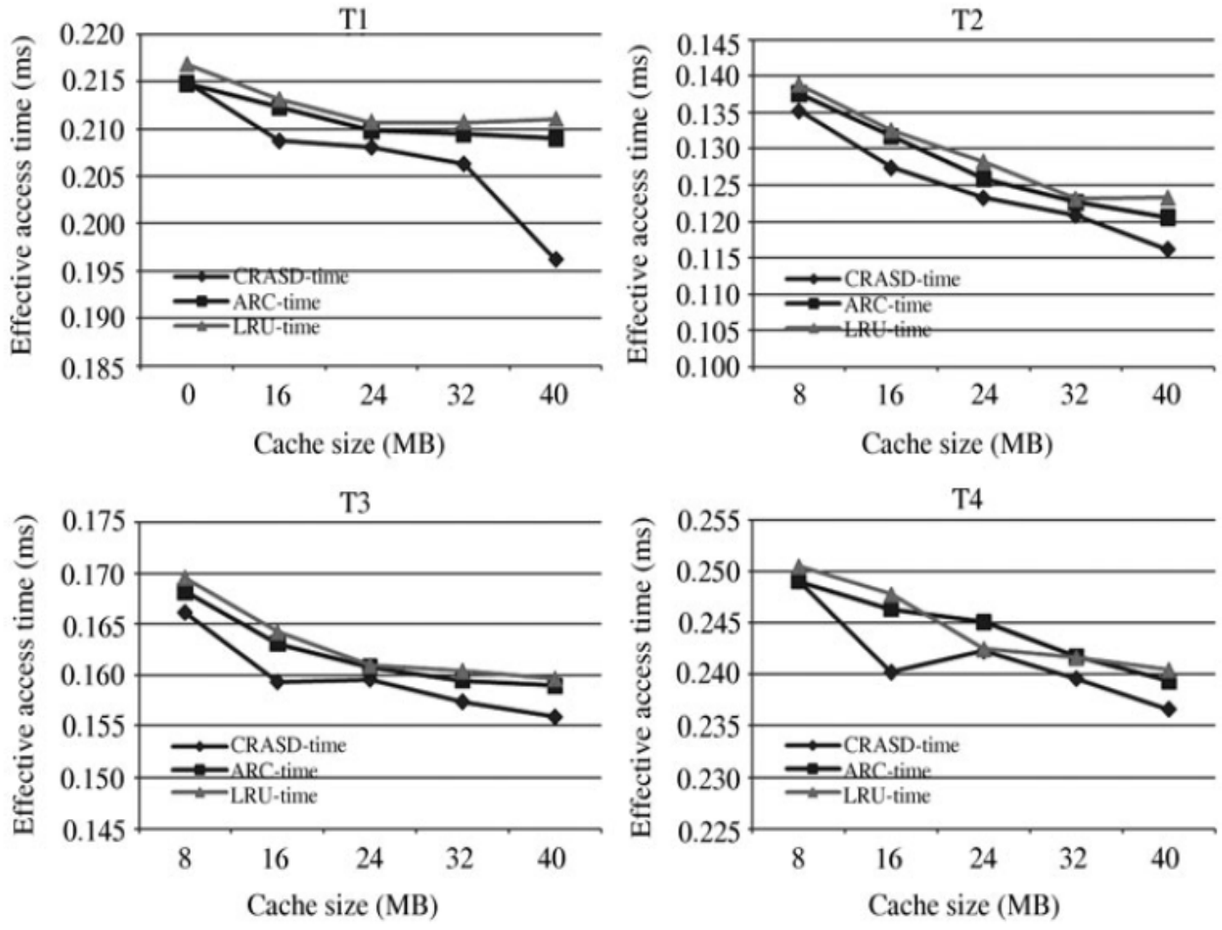
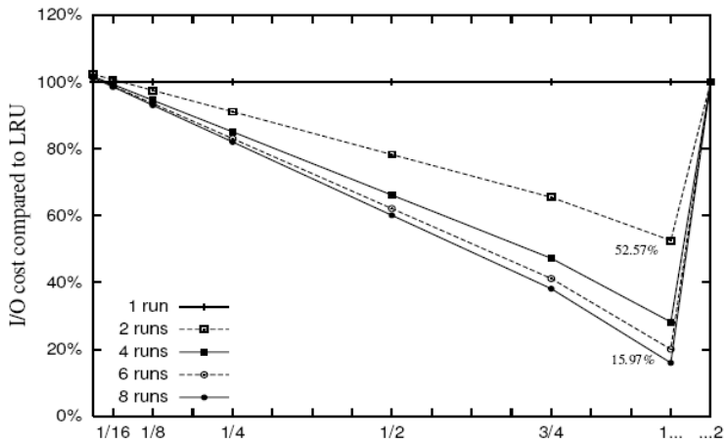


Table 3. Comparative Analysis of Different Policies with KARMA[9]

Algorithm	*Basic-Algorithm*	*Double-Algorithm*	*Global-Algorithm*
LRU	Basic-LRU(LRU+LRU)	Double-LRU	Demote
2Q	Basic-2Q(LRU+2Q)	Double-2Q	Global-2Q
ARC	Basic-ARC(LRU+ARC)	Double-ARC	Global-ARC
MultiQ	Basic-MultiQ (LRU+MultiQ)	Double- MultiQ	Global- MultiQ
LRU-SP	Basic- LRU-SP(LRU-SP+LRU)	N/A	(SP-Karma)

Figure 4. Performance Evaluation of Karma over LRU on Query 3 Runs Multiple Times[9]



Meanwhile, we need to avoid discarding blocks from the set that currently accessed block belongs to, which we call currently accessed set.

Table 2. Characteristics of Traces Used in CRASD Simulations [8]

Trace name	*Number of requests*	*Unique blocks*
T1	17483577	6130732
T2	14002305	4379329
T3	14564573	7194783
T4	1125822	665652

We use PSreq to indicate that set. On a cache hit to a block *p*, CRASD marks the sequential set *s* that the block *p* belongs to as currently accessed set PSreq. If *s* is also the currently replaced set PSrep, we need to stop replacing blocks from *s* by changing the value PSrep to null. On a cache miss to a block *p*, if a free cache slot is available, *p* is loaded into the slot, and then CRASD modifies the corresponding sequential block sets. Depending on the disk address of *p*, three cases may occur: create a new sequential block set, modify an existing sequential block set, or merge two existing sequential block sets into one (the third case is shown in Fig.2). After that, CRASD sorts the priority queue based on the modifications. Finally, a procedure for protecting currently accessed set is carried out similar to what is done on a cache hit. If the buffer cache is full, an extra eviction procedure must run before loading the block *p*. CRASD checks whether the set PSrep is empty. If PSrep is not empty, the block that has the minimum disk address (LBA) in PSrep will be discarded. Otherwise, CRASD checks whether PSreq is the first set in the queue Q. If so, mark the second set to PSrep, else mark the first set to PSrep. After that, we can discard a block as the case in which PSrep is not empty.

3.3. Karma: Know-it-all Replacement for a Multilevel Cache [9]

This technique divides the disk blocks into sets based on their expected access pattern and access frequency.

Each set is allocated its own cache partition, whose size depends on the frequency of access to the set, its size, and the access pattern of its blocks. Partitions accessed with higher frequency are placed at a higher cache level. Each partition is managed by the replacement policy most suited for its set. Since the lower cache levels are supplied with

the application's hints, they can independently compute the partitioning, allocating space only for partitions that do not fit in the upper levels. Karma is applicable to any application which is able to provide general hints about its access patterns.

4. Conclusion

In this paper we presented different cache management mechanisms that determine the cache placement & replacement policies on a per-block basis which is performed in individual level if a multi cache memory. The key idea is to predict a missed block's temporal locality before inserting it into the cache and choose the appropriate insertion policy for the block. In our future work we will include developing and analyzing other temporal locality prediction schemes and also investigating the interaction of our mechanism with pre-fetching. We also need to demonstrate its performance through both theoretical analysis and trace-driven experiments for both unicore as well as multicore processors.

References

1. Hennessy, J. L. & Patterson, D. A. (1996). *Computer Architecture: A Quantitative Approach* (2nd ed.).
2. Zhou, Y., Chen, Z. & Li, K. (2004). *Second-Level Buffer cache Management.* IEEE Transactions on Parallel and Distributed Systems (TPDS), July, 15(7), 505-519.
3. Zhou, Y., Chen, Z. & Li, K. (2004). *Second-level Buffer cache management.* IEEE Transactions on Parallel and Distributed Systems, 15(6), 505-519.
4. Muntz, D. & Honeyman, P. (1992). *Multi-level caching in distributed file systems - or - your cache ain'tnuthin' but trash.* In USENIX Winter Conference.
5. Chen, Z., Zhou, Y. & Li, K. (2003). *Eviction-based placement for storage caches.* In USENIX Annual Technical Conference.
6. Sivathanu, M., Bairavasundaram, L. N., Arpaci-Dusseau, C. A. & Arpaci-Dusseau, R. H. (2005). *Database-aware semantically-smart storage.* In USENIX Conference on File and Storage Technologies (FAST).
7. Gupta, R. & Tokekar, S. (2010). Proficient Pair of Replacement Algorithms on L1 and L2 Cache for Merge Sort. *Journal of Computing*, March, 2(3), 171-175.
8. Nong, X., Yingjie, Z., Fang, L. & Zhiguang, C. (2012). Dual queues cache replacement algorithm based on sequentiality detection Science China. *Information Sciences,* January, 55(1), 191-199.
9. Yadgar, G., Factor, M. & Schuster, A. (2007). *Karma: Know-it-All Replacement for a Multilevelc Ache*. 5th USENIX Conference on File and Storage Technologies, FAST '07.