Cache Memory: An Analysis on Optimization Techniques

Article · April 2015

CITATIONS
9

READS
7,400

2 authors, including:

Munam Ali Shah
COMSATS University Islamabad
208 PUBLICATIONS **2,519** CITATIONS

SEE PROFILE

Cache Memory: An Analysis on Optimization Techniques

Muhammad Waqas Ahmed*, Munam Ali Shah
Department of Computer Science
COMSATS Institute of Information Technology
Islamabad, Pakistan
*Email: muhammadwaqas.raja [AT] gmail.com

Abstract—**Processor speed is increasing at a very fast rate comparing to the access latency of the main memory. The effect of this gap can be reduced by using cache memory in an efficient manner. This paper will discuss how to improve the performance of cache based on miss rate, hit rates, latency, efficiency, and cost.**

Keywords—Cache optimization; cache miss; latency; memory hierarchy

I. INTRODUCTION

The most frequently used data or instructions are kept in cache so that it can be accessed at a very fast rate improving overall performance of the computer. There are multiple level of cache with last level being largest and slowest to the first level being fastest and smallest. In most of the processor first level cache (L1) reside in the processor and second level cache (L2) and third level cache (L3) are on separate chip [1][2]. In multi-core processors, each processor has its own L1 cache. While last level cache is being shared by all the cores [3]. The clock of the processor is several hundred times faster than the access latency of main memory [2]. Cache provides the service to reduce this gap and make the performance of system better.

Cache miss is failure to find the required instruction or data in cache. If data is not found in cache then it would be brought into the cache from main memory. A typical memory hierarchy is shown in Fig. 1 and obtained from [4]. Mainly, cache misses are of three types: i) *compulsory misses* which take place when a memory location is accessed for the first time, ii) *conflict misses* which occur due to insufficient space when two blocks are mapped on the same location and iii) *capacity misses* takes place due to small space as cache can't hold every block that we want to use.

Cache miss rate and miss penalty are the two major factors which effect cache performance. Time required to handle a cache miss is called cache penalty [5]. There are numerous methods to reduce cache miss rates which include victim cache which is a location for temporary storage of cache line which is abolished from cache [6][7], Column associative caches reduces miss rate of direct mapped caches [8], pre-fetching of cache lines [9]. Caches misses can be avoided by understanding the factors which can cause misses and then they can be removed by the programmers from their applications using different CPU profilers [10] and also by rearranging and reorganizing the data [11]. Cache penalty can be reduced by using multi-level cache [12][13]. Cache performance can also be improved by reducing cache hit time [14]. In multi-level cache, usually the first level is kept smaller and faster while second level is larger and much slower than first level. If we increase the cache pipeline stages, we can decrease the gap between cache assess time and processor cycle time.

While discussing multi-core processor for cache optimization, there is a major problem related to the performance of cache which is the cache pollution in last level cache. Cache pollution occurs when data of the weak locality replaces the data of strong locality. Because last level cache is shared in by all the cores of multi-core processor so it will affect all the cores. To address this issue, a user level control system is introduced which will be discussed in this paper briefly [3]. Another issue with multiprocessing environment is a state of inconsistency takes place when data in private cache is shared between multiple caches in different cores of processor [14][15].

There are different mapping techniques through which data from main memory is mapped on to the cache and then used by the processor. These techniques have direct impact on the performance of processor speed. This paper discusses different cache mapping techniques and their effect on performance.

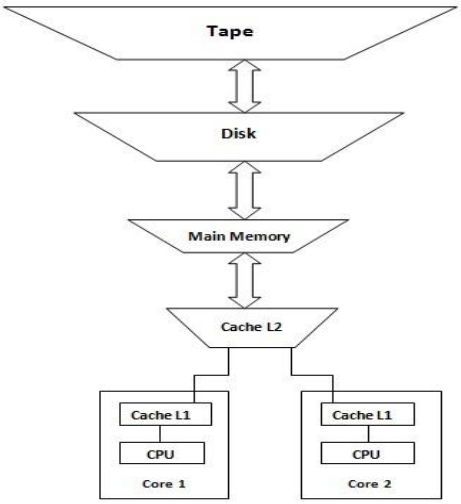

Fig. 1. Multi-Core Memory Hierarchy

Direct map cache is the simplest cache mapping but it has low hit rates so a better approach with slightly high hit rate is introduced which is called set-associative technique. Few of the

other approaches are also discussed in this paper, e.g., sector cache which helps to reduce false sharing [16], and column associative cache which reduces interference misses [8].

One way of improving cache performance is to predict future access of data or instructions that need to be replaced from cache[17]. When cache miss occurs data will be fetched from main memory and then it will be decided which data will be removed from the cache in order to place newly fetched data. For this purpose, we discuss performance of different algorithms for replacement methods [18]. We discuss the performance of 'Least Recently Plus Five Least Frequently Replacement Policy' [19] and 'Blocked algorithm' [20] which reduces cache misses.

The paper discusses cache structure, cache mapping techniques, strategies to reduce cache misses, techniques for avoiding cache penalties and attaining higher cache hit rates. The rest of the paper is organized as follow: Section II reviews related work, Section III analyzes the performance of existing cache replacement algorithms. The paper is concluded is Section IV.

II. Related Work

As there is huge Gap between processor speed and memory access latency, so a lot of effort has been put in this regard to reduce this gap. There is lot of work done to overcome this gap that is hardware based, compiler based and operating system based. Number of replacement algorithms has been implemented for the purpose.

A. Replacement Algorithms

There are different replacement polices which are used to reduce miss rates and make cache performance better. Few of them are Least Recent Used (LRU), Least Frequently used (LFU) and few others. One of the proposed policy Least Recently Plus Five Least Frequently (LR + 5LF) combines LRU and LFU which reduces cache miss rates more than LRU and LFU. In this policy, specific values are given to each block which coincides with LRU and LFU policies. Then these values are combined by an algorithm to give an overall value to the block [19].

Direct mapped cache has very fast access time comparing to other mapping techniques. However, they suffer conflict misses. The problem can be solved through different methods. One of them is column associative cache which adds a rehash bit to each cache set. If multiple addresses share same memory location, they are replaced [12]. One way of improving performance of direct mapped cache is to keep the balance of accessing cache sets. This can be achieved by reducing the size of decoder which helps to reduces access to widely used cache sets. The less accessed cache sets utilization is increased by using replacement policy and programmable decoder. However, balanced cache consume high power for each access comparing to direct mapped caches[21]. Conflict misses can be reduced by saving blocks which are being replaced in victim cache. When a miss occurs in main cache instead of going to main memory first data is searched in victim cache. So if data is present in victim cache then there would be no need to go to main memory. However, chip area cost is high for this technique. Sector caches in which cache lines are divided into blocks are used to reduce false sharing. A false sharing reduction matrix works well to identify the false sharing. The idea is to share data between cache lines and sub blocks. When false sharing occurs then only affected part is replaced [16].

Predicting dead block which may not be accessed again before they are replaced should be evicted. The evicted cache blocks are said to be virtual victim cache. This will eventually increase the hit rate as there would be more live blocks in cache [22]. Victim caches have also been implemented in sectored cache which reduces cache misses. Less number of bit are used in sectored cache to store tags comparing to no-sectored cache [16]. Optimum cache partition (OCP) is another replacement policy which uses cache hint along with LRU replacement policy. Hints are generated by the compiler [17].

B. Cache Algorithms

Cache-oblivious algorithm is used in CPU caches. It has information neither of cache size nor of length of cache line. Because of the fact that it does not have any information about parameters of cache so it will perform better on any size of caches [23]. Cache-aware iterative algorithm is also used to improve performance.

The design of replacement algorithms such as LRU is modified which picks number of cache lines that are not used recently for replacement. Before evicting these cache lines one more check is placed on them which checks cache line for least frequently used out of those selected cache lines and then that with lowest frequency will be evicted or replaced. This algorithm has better performance than LRU [24]. Block bypassing is also a technique to reduce miss rate of L1 cache. Block bypassing algorithm is also used for last level caches as their miss latency is higher than Level 1. It is based on feedback loops. It helps to reduce conflict misses [25].

C. Design Based Optimization

One way of reducing the gap between CPU cycle and memory latency is to use a multi-level cache. Misses of first level can be reduced by introducing second level cache[12].A user level technique has been implemented which is User Level Cache Control (ULCC) through which user can control the allocation of space in cache in their programs and hence reduce cache pollution. A remapping module is used to allocate cache space by remapping between virtual and physical pages. This technique is very complex to implement and using this method produces less hit rates. [3][26].

Different methods are used to make cache's performance better. One of them is to make different banks of cache that can be accessed at different latencies. The proposed design is non-uniform cache access (NUCA). Cache in this design is

TABLE 1 COMPARISON OF CACHE OPTIMIZATION METHODS ON BASIC OF MISS RATES(MR), MISS PENALTY (MP), HIT TIME (HT), HIT RATES (HR), POWER CONSUMMATION (PC), ACCESS TIME (AT), COST, COMPLEXITY, CYCLE TIME(CT).

Techniques	**Cache Comparison Parameters**								
	MR	***MP***	***HT***	***HR***	***PC***	***AT***	***Cost***	***Complexity***	***CT***
Larger Block Size[13][4]	Decrase compulsiry misses, increase conflict misses	High	Less	High	High	Slow	N/A	0	High
Higer Associativity[4][1][27]	Reduce capicity,conflict misses	Less	High	Low	High	Fast	High	1	High
Larger Cache[5][13][28]	Reudce Miss Rates, cache cohernce issue	N/A	High	High	High	Slow	High	2	Low
Way Predictaion[4]	Reduces Conflict Misses	High	High	High	Low depends on way prediction	Slow	N/A	2	High
Column Associative Cache[8]	Reduce Conflict Misses	N/A	High	Low	High	Fast Same as Direct Mapped	High	2	High
Reactive associative cache[29]	Reduce Misses	Less	Higher than Direct Mapped	High	N/A	Slow	Same as Direct Mapped	1	Low
Multilevel Cache[4]	Cache Cohernce Misses takes place	Low for L1	High	High	High	Slow	High	2	Higher than Direct mapped
Victim Cache[30] [31]	Reduce Cache Misses	Low	Comparable with Direct Mapped Cache	High	Low	Medium	High	1	Higher than Direct Mapped
Pipelined Cache[4]	N/A	Low	High	Low	N/A	Fast	High, Hardware Added	2	Cache with three pipeline stages have less cycle time
Jigsaw[32]	Reduce Interfernce Misses	Low	High	Low	Low	Fast	High, Extra Hardware required	3	N/A
Cache Miss Lookaside[33]	Reduces Conflict Misses	Less	High	Low	High	Fast	High	2	Equal to 2 way set associative
Predictive Sequential Associative Cache[34]	Miss rate same as 2 way set Associative	Same As Direct Mapped Cache	Same As Direct Mapped Cache	Low	Same As Direct 2 way set Associative	Same as Direct Mapped Cache	Same as Column Associative Cache	1	Same as Direct Mapped Cache
ULCC[3]	Elimaenate Cache Pollution	N/A	High	Low	N/A	Fast	High	2	N/A
Small cache[13]	Increase Miss Rate	N/A	High	N/A	Low	Fast	Low	0	N/A
LR+5LF [19]	Reduces Cache misses with greater amount than LRU, FIFO and LFU at L1 cache and same for L2	Same as LRU and LFU	Same as LRU and LFU	Same as LRU and LFU	Same as LRU and LFU	Same as LRU and LFU	High, extra Hardware Needed	2	N/A
Resizing and Remapping[35]	N/A	N/A	N/A	N/A	Very Low	N/A	N/A	2	N/A

managed in such a way that fastest banks serve most of data. A switched networks is used which moves data to faster banks in steps. Low latency access is the main feature of NUCA designs [36].Cache organizations have huge impact on performance of cache memory. Interference and scalability issues have been solved by jigsaw. In jigsaw software defines the shares (bank Partitions) and size of virtual cache. It helps to define how data would be mapped to shares. Every share has a unique id. Jigsaw produces better performance than NUCA [32].

D. Compiler And Prediction Based Optimization

One of the cache optimization techniques which was used in the past is to optimize the loops through compiler. Loops are reduced to smaller size so that accessed data can be set in the cache. Then compiler will analyze all the task which will share same data, those tasks will be assigned to one processor. In this way all the tasks will be executed consecutively which will use same data from cache [37]. Compilers have been used for replacement decisions of cache by generating hints for future accessed data. A set of compiler algorithms have been developed which does the prediction about the data to be reused in near future. These predications are used to improve the hit rates. The algorithm used for the purpose is evict-me which uses cache line tag of one bit. So when ever evict-me tag is set for any cache line, the cache line will be replaced[17]. Both these techniques are complex and consume extra energy.

Different methods are used to implement two way set associative cache. Out of them one method is predictive sequential associative cache. Implementing set associative cache through this method brings access time almost equal to direct mapped cache. This method uses different predication which helps to reduces access time and searching time. However, this method has low hit time compared to direct mapped cache [34]. Producing the next data to be used by cache is one way to improve the performance of the cache. So instead of waiting to occur a cache miss and then to fetch the data from main memory, data perfecting is used to produce data in advance which is net to be used [38]. When a block is evicted from last level cache in a multi-processor structure cache usage is not taken under consideration for every processor. A new scheme is introduced to overcome this problem which allows different processors to put some constraints on each cache set [39].

E. Web Based Optimization

The World Wide Web provides access to large shared data objects. Server overloading is a major problem faced by web users. As Internet usage is increasing at a very rapid rate its bandwidth is likely to suffer in near future. Mostly used objects are cached to locations closed to the clients. Perfecting, cache placement and replacement, cache coherency, cache contents, user access patterns, load balancing, proxy placement and dynamic data caching are few of the methods used for web cache optimization [40]. In Content Centric Network (CCN) cache size is allocated heterogeneously to improve cache performance [41].

III. Performance Evaluation

Different methods and techniques for cache optimization which have been implemented in the recent past have been analyzed. We compare most of these methods and techniques for different parameters and present them in Table 1. These methods are compared with each other on bases of the listed parameters. Miss rates (MR), Miss penalty (MP), Hit time (HT), Hit Rates (HR), Power Consummation (PC), Access Time (AT), Cost, Complexity, Cycle Time (CT). All of these methods have some advantages and disadvantage. Table 1 summarizes all these advantages and disadvantages.

Larger block size of cache is the simplest way to reduce the miss rate however they will produce longer hit times along with higher cost. Larger block size cache also increase the conflict misses. In contrast smaller block size cache reduces cache hit times. Multi-level caches are quite efficient in improving the overall system performance however hierarchy design of multi-level caches is very complex.

IV. Conclusions

In this paper, we discussed and evaluated the performance of different techniques and algorithms used for cache optimization. We summarized our finding in Table 1. We believe that it is hard to identify a certain cache optimization technique as the best choice in all cases. Each technique is associated with its design constraints, advantages and limitations. Different techniques could be further enhanced. For example, the rate of conflict misses is reduced by using larger block size, larger cache and way prediction methods. However, using larger block size may increase miss penalty, reduced hit time and power consumption. On the other side, larger cache produces slow access time and high cost. It is associated with cache coherence problem. Higher associativity produce fast access time but they have low cycle time. Victim cache reduces miss rate at a high cost comparing to Cache miss look aside. LR + 5LF is a very good technique for reducing cache misses at a high rate comparing to some of the other techniques like LRU, FIFO, LFU. However, it is more complex then all of these methods. Cache pollution is eliminated by ULCC and also they have fast access time but complexity is high. Miss penalty is reduced by using pipelined cache which is very complex method. In future, we intend to enhance the multilevel cache by addressing its coherence issue. We will investigate how MESI could be integrated to achieve better performance of multilevel cache.

References

[1] J. S. Yadav, M. Yadav, and A. Jain, "CACHE MEMORY OPTIMIZATION," *International Conferences of Scientific Research and Education*, vol. 1, no. 6, pp. 1–7, 2013.

[2] H. Dybdahl, "Architectural Techniques to Improve Cache Utilization" *Diss. PhD thesis, Norwegian University of Science and* Technology, 2007.

[3] X. Ding and K. Wang, "ULCC : A User-Level Facility for Optimizing Shared Cache Performance on Multicores." *Acm sigplan notices, Vol. 46, No. 8,* ACM, 2011.

[4] S. Paper, R. Sawant, B. H. Ramaprasad, S. Govindwar, and N. Mothe, "Memory Hierarchies-Basic Design and Optimization Techniques Survey on Memory Hierarchies – Basic Design and Cache Optimization Techniques," 2010.

[5] Fu, John WC, and Janak H. Patel. "Data prefetching in multiprocessor vector cache memories." *ACM SIGARCH Computer Architecture News*, Vol. 19. No. 3. ACM, 1991.

[6] J. R. Srinivasan, "Improving cache utilisation," *Phd Diss., University Of Cambridge*, no. 800, 2011.

[7] N. P. Jouppi, "Improving direct-mapped cache performance by the addition of a small fully-associative cache and prefetch buffers," *ACM SIGARCH Comput. Archit. News*, vol. 18, pp. 364–373, May 1990.

[8] A. Agarwal and S. D. Pudar, "Column-associative Caches: A Technique For Reducing The Miss Rate Of Direct-mapped Caches," *Proc. 20th Annu. Int. Symp. Comput. Archit.*, pp. 179–190. 1993.

[9] P. Cao and E. W. Felten, "A Study of Integrated Prefetching and Caching Strategies." Vol. 23. No. 1. ACM, 1995.

[10] A. Pesterev, N. Zeldovich, and R. T. Morris, "Locating cache performance bottlenecks using data profiling," *Proc. 5th Eur. Conf. Comput. Syst. - EuroSys '10*, p. 335, 2010.

[11] T. M. Chilimbi, B. Davidson, and J. R. Larus, "*Cache*-conscious structure definition," *ACM SIGPLAN Not.*, vol. 34, no. 5, pp. 13–24, May 1999.

[12] S. Przybylski, M. Horowitz, and J. Hennessy, "Characteristics of Performance-Optimal Multi-Level Cache Hierarchies," *ACM SIGARCH Computer Architecture News*, Vol. 17. No. 3. ACM, 1989.

[13] C. Science and S. Engineering, "Survey on Hardware Based Advanced Technique for Cache Optimization for RISC Based System Architecture," vol. 3, no. 9, pp. 156–160, 2013.

[14] M. R. Marty, "Amdahl's law in the multicore era." *IEEE* vol. no. 7, pp. 33-38, 2008

[15] Wada, Tomohisa, Suresh Rajan, and Steven A. Przybylski. "An analytical access time model for on-chip cache memories." *Solid-State Circuits, IEEE Journal*, pp. 1147-1156, 1992.

[16] K. Liu and C. King, "On the effectiveness of sectored caches in reducing false sharing misses," *Proc. Int. Conf. Parallel Distrib. Syst.*, pp. 352–359, 1997.

[17] K. S. McKinley, a. L. Rosenberg, and C. C. Weems, "Using the compiler to improve cache replacement decisions," *Proceedings.International Conf. Parallel Archit. Compil. Tech.*, pp. 199–208, 2002.

[18] S. Hardware and C. C. Optimizations, "IBM Research Report," vol. 23998, 2006.

[19] A. Abdelfattah and A. A. Samra, "Least Recently Plus Five Least Frequently Replacement Policy (LR + 5LF)," *Int. Arab J. Inf. Technology*, vol. 9, no. 1, pp. 16–21, 2012.

[20] S. Lam, E. Wolf, and E. Rothberg, "The Cache Performance of Blocked." *ACM SIGOPS Operating Systems Review, 25.Special Issue (1991):* 63-74, 1991.

[21] C. Zhang, "Balanced Cache : Reducing Conflict Misses of Direct-Mapped Caches through Programmable Decoders," *ACM SIGARCH Computer Architecture News.* Vol. 34. No. 2. IEEE Computer Society, 2006.

[22] D. Burger and D. A. Jim, "Using Dead Blocks as a Virtual Victim Cache," *Proceedings of the 19th international conference on Parallel architectures and compilation techniques*, ACM, 2010.

[23] M. Frigo, C. E. Leiserson, H. Prokop, and S. Ramachandran, "Cache-Oblivious Algorithms," *ACM Trans. Algorithms*, vol. 8, no. 1, pp. 1–22, Jan. 2012.

[24] M. Kampe, P. Stenstrom, and M. Dubois, "Self-correcting LRU replacement policies," *Proc. first Conf. Comput. Front. Comput. Front. - CF'04*, p. 181, 2004.

[25] M. Kharbutli and Y. S. Y. Solihin, "Counter-Based Cache Replacement and Bypassing Algorithms," *IEEE Trans. Comput.*, vol. 57, no. 4, 2008.

[26] A. Sandberg, D. Eklöv, and E. Hagersten, "Reducing Cache Pollution Through Detection and Elimination of Non-Temporal Memory Accesses," *Proceedings of the 2010 ACM/IEEE international conference for high performance computing, networking, storage and analysis. IEEE Computer Society*, November, 2010.

[27] E. Rotenberg, S. Bennett, and J. E. Smith, "A trace cache microarchitecture and evaluation," *IEEE Trans. Comput.*, vol. 48, no. 2, 1999.

[28] N. Miguel and D. Cerqueira, "Cache : Why Level It," *Proceedings of the 3rd Internal Conf. on Computer Architecture*, pp. 19–26, 2002.

[29] B. Batson and T. N. Vijaykumar, "Reactive-associative caches," *Proc. 2001 Int. Conf. Parallel Archit. Compil. Tech.*, pp. 49–60, 2001.

[30] H. Wasserman, "Victim-Caching for Large Caches and Modern Workloads," *University of California, Berkeley* 1996.

[31] D. Stiliadis and a. Varma, "Selective victim caching: a method to improve the performance of direct-mapped caches," *Proc. Twenty-Seventh Hawaii Int. Conf. Syst. Sci. HICSS-94*, pp. 412–421, 1994.

[32] N. Beckmann and D. Sanchez, "Jigsaw: Scalable software-defined caches," *Proc. 22nd Int. Conf. Parallel Archit. Compil. Tech.*, pp. 213–224, Sep. 2013.

[33] B. N. Bershad, T. H. Romer, and J. B. Chen, "Avoiding Con ict Misses Dynamically in Large Direct-Mapped Caches." *ACM SIGPLAN Notices*, Vol. 29. No. 11. ACM, 1994.

[34] B. Calder, D. Grunwald, and J. Emer, "Predictive sequential associative cache," *Proceedings. Second Int. Symp. High-Performance Comput. Archit.*, no. February, 1996.

[35] S. Ramaswamy, S. Yalamanchili, and C. Architectures, "Improving Cache Efficiency via Resizing + Remapping," *Computer Design, 2007. ICCD 2007, 25th International Conference on. IEEE, 2007*.

[36] C. Kim, D. Burger, and S. W. Keckler, "An adaptive, non-uniform cache structure for wire-delay dominated on-chip caches," *ACM SIGARCH Comput. Archit. News*, vol. 30, no. 5, p. 211, Dec. 2002.

[37] K. Ishizaka, M. Obata, and H. Kasahara, "Cache Optimization for Coarse Grain Task Parallel Processing using Inter-Array Padding." *Springer Berlin Heidelberg*, pp. 64-76. 2004.

[38] S. Vanderwiel, D. J. Lilja, and S. Vanderwiel, "A Survey of Data Prefetching Techniques A Survey of Data Prefetching Techniques," *Proceedings of the 23rd International Symposium on Computer Architecture,* no. October, 1996.

[39] J. Chang and G. S. Sohi, "Cooperative cache partitioning for chip multiprocessors," *Proc. 21st Annu. Int. Conf. Supercomput. ICS 07*, p. 242, 2007.

[40] J. Wang, "A survey of web caching schemes for the Internet," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 29, no. 5, p. 36, Oct. 1999.

[41] Y. Xu, Y. Li, T. Lin, Z. Wang, W. Niu, H. Tang, and S. Ci, "A novel cache size optimization scheme based on manifold learning in Content Centric Networking," *J. Netw. Comput. Appl.*, vol. 37, pp. 273–281, Jan. 2014.