International Journal of Advance Research in Science and Engineering
Vol. No.4, Special Issue (01), Spetember 2015
www.ijarse.com
IJARSE
ISSN 2319 - 8354

AN OVERVIEW OF HARDWARE BASED CACHE OPTIMIZATION TECHNIQUES

Swadhesh Kumar[1], Dr. P K Singh[2]

[1,2]*Department of Computer Science and Engineering,*

Madan Mohan Malaviya University of Technology, Gorakhpur, Uttar Pradesh, (India)

ABSTRACT

Cache Memory is a high speed semiconductor memory acts as a buffer between CPU and Main Memory. In current generation processors, the processor- memory bandwidth is the main bottleneck, because a number of processor cores sharing it through the same processor memory interface or bus. The on chip memory hierarchy is an important resource that should be managed efficiently against the raising performance gap between processor and memory. This Paper yields a comprehensive survey to improve the cache performance on the basis of miss rate, hit rate, latency, efficiency and cost.

Keywords: Conflict Miss, Compulsory Miss, Capacity Miss, Miss Rate, Hit Rate, Latency, Efficiency.

I. INTRODUCTION

Cache provides the fastest possible storage after the registers used to kept the most frequently used data or instructions so that it can be accessed quickly. In multi core chips, cache is shared by multiple cores on a chip allows different cores to share data and an update performed by one core can be seen other cores with no need for cache coherence methods. There are multiple levels of cache memory with first level being smallest and fastest to last level being largest and slowest. Generally, in most of processors first level cache resides in the processor, second level cache and third level cache are on separate chip [1]. In multicore processors, each processor core has its own L1 cache while last level cache is shared by all the cores [2].

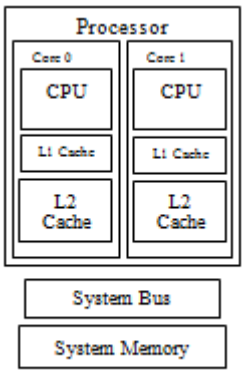

Figure1.1

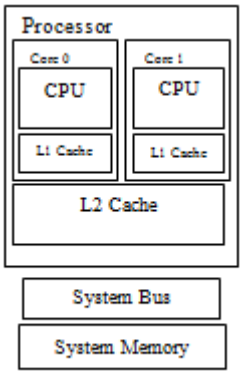

Figure 1.2

Fig. 1.1 and Fig. 1.2 shows two typical multicore processor architecture. Fig. 1.1 shows multicore processor with separate L2 cache while Fig. 1.2 shows multicore processor with shared L2 cache. The clock of processor is some hundred faster than the access latency of main memory [3]. Cache provides the service to reduce this gap and make system performance better. Cache miss is failure to find the required instruction or data in memory and if a miss occurs then would be brought in to the cache from main memory in the form of blocks. The three 'C' model sorts all misses in to three simple categories:

1.1 Compulsory

The very first access to a block can not be in the cache, so the blocks must be brought n to the cache. Compulsory misses are those that occure even if there is an infinite cache.

1.2 Capacity

If the cache can not contain all the blocks required during execution of a program, capacity misses will occure because of blocks being discarded and later retrieved due to limted size of cache.

1.3 Conflict

If the block placement strategy is not fully associative , conflict misses will occure due to insufficient space when two blocks are mapped on the same location.

The two major factors effects cache performance are miss rate and miss penalty. The time needed to handle miss is known as miss penalty[4]. There are several methods to reduce miss rates which include victim cache or cache line which is eliminated from cache[5][6]. Direct mapped cache is a popular design choice for processors but it suffer systematic interference misses when more than one address maps in to the same cache set. Column associative caches minimize miss rate of direct mapped caches[7]. Cache misses can be reduced by understanding the causing factors and factors can be removed by by the programmers from applications using different CPU profilers and also by reorganizing and rearranging data[8]. Multilevel cache can be used to reduce miss penalty[9][10] and cache performance can also be optimized by reducing the hit time[10]. In multi-level cache, the first level is small but faster while the second level is large but slower than first level. This significantly improves the average memory access time of a system when each level must have a significantly larger capacity than the level above it in the hierarchy. Locality of references seen by each level decreases as one gets deeper in the hierarchy and, requests to recently referenced data are handled by the upper levels of the memory system. By increasing cache pipeline stages, the gap between processor cycle time and cache access time can be reduced. In multicore processor, there is a problem of cache pollution occurs n the last level of cache. Cache pollution takes place when the data of weak locality replaces data of strong locality. Since the last last level of cache is shared by all cores of processor so all the cores get affected. To address this issue, a user level control system is introduced.

There are various kinds of mapping techniques used to map data from main memory to cache. These mapping techniques directly effects the processor speed. In this paper, the various mapping techniques and their impact on processor performance is discussed.

Direct mapping is a simple mapping technique where a particular block of data from main memory can be placed in a fixed location in to the cache. Direct map caches are simple to design but have a low hit rate. A

better approach introduced is set-associative mapping with improved hit rate[11]. In this paper, higher associativity to reduce miss rate is discussed along with some other techniques to reduce miss rate such as using larger blocks, using large size cache and compiler optimization. Way prediction and pseudo associative cache are also discussed to reduce miss rate. In way prediction technique, extra bits are kept in the cache for the prediction of the set of next cache access[12].

The organization of rest of paper is as fallows: The next section I defines the survey and review of related work, Section III defines the analysis of the performance of various optimization techniques and finally Section IV defines the conclusion.

II. RELATED WORK

2.1 Techniques to Reduce Cache Miss Rate

Hardware based optimization techniques to reduce cache miss rate are:

2.1.1 Using Large Size Cache

Large caches reduce the capacity misses [4]. In larger cache there is less chance that there will be conflict but the drawback is larger hit time and higher cost [9].

2.1.2 Using larger Blocks

Using large blocks is a simplest method to reduce the compulsory misses because larger blocks take advantage of spatial locality [9].

2.1.3 Higher Associativity

Higher associativity reduces conflict misses and comes at the cost of increased hit time [12]. Practical results show that an 8- way set associative cache has, generally same miss rate as fully cache. Direct mapped cache of size N has approximately same miss rate as 2- way set associative cache of size N/2. This observation called 2:1 cache rule of thumb and held for cache sizes less than 128 KB.

2.1.4 Way Prediction and Pseudo Associative Cache

Way Prediction is a technique where extra bits are kept in the cache for prediction of the set of next cache access [13]. Here, the multiplexer is set early to select desired block and only one tag comparison is needed when accessing cache. If prediction is correct then there is a fast hit, but if not then it tries other block, it changes the way predictor and has an extra clock cycle latency. Pseudo Associative caches are also called column associative cache. In this cache the space is logically divided in to two zones. For every visit Pseudo Associative cache will act as direct mapped in first zone so each block has only one place to appear in cache. In case of hit case of hit this cache is just like the direct mapped cache. In case of miss, CPU will visit a specified location in another zone and if cache hits this time a pseudo hit happens and then the block is swapped for the block of the first entry.

2.1.5 Compiler Optimization

Compiler optimization technique reduces miss rates without any hardware change. This reduction comes from optimized software. Huge performance gap between processor and main memory has motivated compiler designers to review the memory hierarchy to see if compile time optimization can improve performance. So the research is divided between improvements in instruction misses and in data misses. Merging Arrays, Loop Interchange, Loop Fusion and Blocking are optimizations found in many modern compilers [14].

2.2 Techniques to Reduce Cache Hit Time

Hardware based optimization techniques to reduce cache hit time are:

2.2.1 Small and Simple Cache

The index portion of address to read the tag memory and then comparison with address is a time consuming portion of cache hit. So small cache can be faster and help the hit time. But for second level cache, it is critical to keep cache small enough to fit on the same chip as the processor to avoid time penalty of going off-chip. So for lower level caches some design strike a compromise by keeping tags on chip and data off- chip, resulting a fast tag check and provide greater capacity for separate memory chips [9].

2.2.2 Avoiding Address Translation During Indexing of the Cache

Virtual addresses generated by CPU have to be translated in to physical address used by traditional caches. The guideline of making the common case suggests that we use virtual addresses for cache because hits are more common than misses. Such caches are termed as virtual cache. Virtual caches eliminate address translation from hit time but they might have to be flushed every time process is switched so by storing process identifier alongside address tag in cache, flushing can be avoided until operating system recycles process identifier. Another reason why virtual caches are not popular is that operating system and user programs may use two different virtual addresses for the same physical address. One solution to get the best of both virtual and physical cache is to use part of the page offset same in both virtual and physical addresses to index the cache [14].

2.2.3 Pipeline Cache Access to Increase Bandwidth

This optimization is merely to pipeline cache access so that effective latency of level one cache hit can be multiple clock cycles, giving fast clock cycle time and high bandwidth but slow hits. For instance, the pipeline for Pentium Processor took one clock cycle to access instruction cache, for Pentium Pro through Pentium III it took two clock cycles and for Pentium IV it takes four clock cycles. This division increases stages of pipeline, leading to high penalty on mispredicted branches and more clocks between issue of the load and the use of data [12].

2.2.4 Trace Cache

To find lots of instruction level parallelism, it is also a challenge to find enough instruction every cycle without using dependencies. The Trace cache is an instruction cache in processor that keeps dynamic instruction sequences after they have been fetched and executed. In order to follow instructions at subsequent times, there is no need to go regular cache or memory for the same instruction sequence. The main advantage of trace cache is that it reduces the needed fetch bandwidth on processing pipeline [14].

2.3Techniques to Reduce Cache Miss penalty

Hardware based optimization techniques to reduce cache miss penalty are:

2.3.1 By Using Multilevel Cache

It is defined as multiple levels of cache, with small size fast cache is backed up by another large size slow cache. In multilevel caches operation is proceeds by first checking the fastest cache or level one cache. In multilevel cache the second level cache is much bigger than first level because the second level cache contains everything of first level [12]. If size of second level cache is not much bigger than first level then the local miss rate will be high.

2.3.2 Reads Priority Over Writes on Miss

A write buffer is a place to implement this optimization. The simplest way is that for a read miss to wait until write buffer is empty. The write buffers may create hazards since they contain updated value of location needed when a read miss occurs, that is a read after write hazard through memory. The solution is to check the write buffer contents when a read miss occurs. If there are no conflicts and memory system is available, send the read before the writes to reduce the miss penalty [14]. Most of the processors gives reads priority over write on miss.

2.3.3 Merging Write Buffers

If the write buffer is empty, then data and full address are written in the buffer by the processor. The processor continues working while the write buffers prepare to write the content to memory. If the buffer has other modified blocks inside it, then the addresses can be checked to see if address of new data. If, it matches then the new data is combined with entry. This is known as merging write buffers. Write merging is used in Sun Niagara and many other processors [14]. If the buffer is full and there is no address match then the processor and cache must wait until the buffer has empty entry.

2.3.4 Victim Cache

Victim cache contains the dirty blocks that are discarded from main cache because of a miss [15]. It is a fully associative cache with size 4 to 16 lines residing between a direct mapped L1 cache and next level of memory. When a cache miss occurs then before going to the next level victim cache is checked. If the desired address found in victim cache, then the desired data is returned to CPU. Victim cache reduces the impact of conflict misses [16].

2.3.5 Early restart and Critical words first

On the basis of observation it is clear that CPU normally needs only one word of the block at particular time [4]. So according to these techniques, not to wait for the complete block to be loaded before sending the requested word and restarting CPU. According to critical word first technique: First request the missed word from memory and pass it to the CPU immediately. The CPU continues execution while filling the remaining of words n the block. Since the critical word first fetched so it is also called the wrapped fetch or requested word first. And according to Early Start technique: Fetch words in sequential order but when the requested word of block arrives, send it to the CPU to continue execution [17].

III. PERFORMANCE EVALUATION

In this paper, we have discussed and analyzed different cache optimization techniques implemented in recent past. We have compared most of these techniques for different parameters and summarized them in Table1. Various parameters on the basis of which different techniques have been compared are: Miss Rates(MR), Hit Rates(HR), Miss Penalty(MP), Hit Time(HT), Power Consumption(PC), Access Time(AT), Cost, Complexity and Cycle Time(CT). All of these techniques have some advantages and disadvantages, also summarized in Table1.

Table 1 Comparison Of Cache Optimization Techniques On The Basis Of Miss Rates(Mr), Miss Penalty (Mp), Hit Time (Ht), Hit Rates (Hr), Power Consumption (Pc), Access Time (At), Cost, Complexity And Cycle Time(Ct).

Techniques	Comparison Parameters								
	MR	***MP***	***HT***	***HR***	***PC***	***AT***	***Cost***	***Complexity***	***CT***
Larger Cache	Reduce Miss Rates, cache coherence issue	N/A	High	High	High	Slow	High	2	Less
Larger Block	Decrease compulsory misses, increase conflict misses	High	Less	High	High	Slow	N/A	0	High
Higher Associativity	Reduce capacity, conflict misses	Less	High	Low	High	Fast	High	1	High
Way Prediction	Reduces Conflict Misses	High	High	High	Low depends on way	Slow	N/A	2	High
Compiler Optimization	Reduce Miss Rates	High	Less	High	Less	Fast	N/A	3	N/A
Small and simple cache	Increase Miss Rate	N/A	High	N/A	Less	Fast	Less	0	N/A
Avoiding Address Translation during indexing of the Cache	Reduce Miss Rate	N/A	Less	N/A	Less	Fast	N/A	2	Less
Pipelined Cache	N/A	Less	High	Low	N/A	Fast	High, Hardware Added	2	Less with 3 stages
Trace Cache	N/A	N/A	Less	High	Less	Fast	N/A	2	N/A
Multilevel cache	Cache Coherence Misses takes place	Low for L1	High	High	High	Slow	High	2	Higher than Direct
Reads Priority over writes on miss	N/A	Less	Less	N/A	N/A	N/A	N/A	2	Less
Merging Write Buffers	N/A	Less	N/A	N/A	Less	N/A	High	3	Less
Victim Cache	Reduce Cache Misses	Less	Comparable with Direct	High	Low	Medium	High	1	Higher than Direct Mapped
Early restart and Critical word first	N/A	Less	N/A	N/A	N/A	Fast	N/A	2	Less

IV. CONCLUSION

In this paper, we have discussed and analyzed discussed and determined the performance of various techniques used for cache optimization. We have summarized our findings in Table1. We found that its hard to identify particular cache optimization as the best choice in all the cases. Every technique has its design constraints, advantages and limitations. Some techniques could be further enhance. For instance conflict misses can be reduced by using larger block size, larger cache and way prediction method. But using larger block size may increase miss penalty and power consumption. On the other side larger caches are costly and produces slow access time. It is also associated with cache coherence problem. Higher associativity has fast access time but low cycle time. Victim cache reduces conflict misses at high cost comparing to cache miss. To conclude, all three schemes Reducing Hit Time, Reducing Miss Rate and Reducing Miss Penalty should be collectively applied for better cache optimization.

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] 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.

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

[4] 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.

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

[6] 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.

[7] 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.

[8] 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.

[9] 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.

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

[11] Eric Rotenberg, Steve Bennet, "A Trace Cache Microarchitecture and Evaluation IEEE TRANSACTIONS ON COMPUTERS, VOL. 48, NO. 2, FEBRUARY 1999.

[12] 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

[13] C. Kozyrakis, "Advanced Caching Techniques." 2008.

[14] David A. Patterson, John L. Hennessy. "Computer organization and design: the hardware/software interface". 2009.

[15] 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.

[16] H. Wasserman, “Victim-Caching for Large Caches and Modern Workloads,” University of California, Berkeley 1996.

[17] Markus Kowarschik, Christian Weis, "An Overview of Cache Opitimization Techniques and Cache-Aware Numerical Algorithms".2003