Automation, Control and Intelligent Systems
2020; 8(3): 24-28
http://www.sciencepublishinggroup.com/j/acis
doi: 10.11648/j.acis.20200803.11
ISSN: 2328-5583 (Print); ISSN: 2328-5591 (Online)

An Overview of Cache Memory in Memory Management

Ademodi Oluwatosin Abayomi, Ajayi Abayomi Olukayode, Green Oluwole Olakunle

Computer Engineering Department, School of Engineering, Lagos State Polytechnics, Ikorodu, Lagos, Nigeria

Email address:
ademodi.o@mylaspotech.edu.ng (A. O. Abayomi.), ajayi.a@mylaspotech.edu.ng (A. A. Olukayode.),
green.o@mylaspotech.edu.ng (G. O. Olakunle)

To cite this article:
Ademodi Oluwatosin Abayomi, Ajayi Abayomi Olukayode, Green Oluwole Olakunle. An Overview of Cache Memory in Memory Management. *Automation, Control and Intelligent Systems.* Vol. 8, No. 3, 2020, pp. 24-28. doi: 10.11648/j.acis.20200803.11

Received: July 14, 2020; **Accepted**: August 7, 2020; **Published**: October 30, 2020

Abstract: Cache memory are used in small, medium and high speed Central Processing Unit (CPU) to hold provisionally those content of the main memory which are currently in use. Preferably, Caches ought to have low miss rates, short access times, and power efficient at the same time. The design objectives are frequently gainsaying in practice. Nowadays, security concern about caches information outflow is based on the proficient attack of the information in the memory and the design for security in the cache memory are even more controlled and typically leads to significant cache performance. Fault tolerance is an additional advantage of the cache architecture which can be guaranteed in the memory to overcome the processor speed gap in the memory, the routine gap between processors and main memory continues to broaden, increasingly aggressive implementations of cache memories are needed to bridge the gap. In this paper, the objective is to make cache memory unsurprising as seen by the processor, so it can be used in hard real time system to achieve this we consider some of the issues that are involved in the implementation of highly optimized cache memories and survey the techniques that can be used to help achieve the increasingly stringent design targets.

Keywords: Cache Memory, Central Processing Unit, Main Memory, Processor

1. Overview of Cache Memory

Cache is very essential in the main memory of the central processing unit of a computer system [1], the processors confirm if the data from the location is ready in the cache when attempting to read from or write to a location in main memory. If so, the processor will read from or write to the cache instead of the much slower main memory, which makes the processes within the system much faster. Nearly all current desktop and server CPUs have at least three autonomous caches [2]: an instruction cache to speed up executable instruction fetch, a data cache to speed up data fetch and store, lastly a translation lookaside buffer (TLB) used to speed up virtual-to-physical address translation for both executable instructions and data. A single TLB can be provided for access to both instructions and data, or a separate Instruction TLB (ITLB) and data TLB (DTLB) can be provided. The data cache is usually organized as a hierarchy of more cache levels (L1, L2, etc).

Figure 1 shows the representation of a simple basic cache. In this scheme, every time the CPU executes a read or write, the cache may interrupt the bus transaction; thereby permit the cache to decrease the response time of the system. Before discussing this cache model, let’s define some of the common terms used when talking about cache.

Cache Hits

Cache hit occur when the cache contains the required information requested for in the main memory

Cache Miss

Cache miss occur when the cache does not contain the required information requested for in the main memory

Cache Consistency

Snoop and Scarf are methods of cache consistency, cache is said to be snoop when a cache is watching the address lines for transaction and scarf When a cache takes the information from the data lines, Since cache is a reproduction of a small piece main memory, it is important that the cache always reflects what is in main memory.

2. Introduction

Cache is tiny high speed memory naturally Static RAM (SRAM) that can holds most up to date and accurate to use pieces of main memory [3]. CPU caches are usually use to connect memories for correlation in the memory; cache is highly link because of it complex structure. Therefore, most CPU cache memories are organized as two-dimensional arrays [4]. The first and second dimensions are connected set in the caches memories. The location of the ID is established by a function of the address bits of the memory request. The line ID within a set is determined by matching the address tags in the target set with the reference address. The caches location that are connected with one and the other are commonly referred to as direct-mapped caches while caches location that its connection are greater than one are referred to as set-associative caches [5]. The cache is totally connected if there is only one position. Each cache entry consists of several data, and a tag that identifies the main memory address of that data. Memory request can be fulfilled by the cache by comparing the requested address with the address tags in the tag array. Cache can be accessed by its two components, by accessing the tag array and then carry out tag comparison to determine if the data is in the cache. The other is to approach the data array to fetch out the requested data. The outcome of the tag assessment is used to pick the requested line from within the set driven out of the data array for a set-associative cache. Majority of computer caches are accessed on real memory address, whereas the ALU generates the virtual memory address.

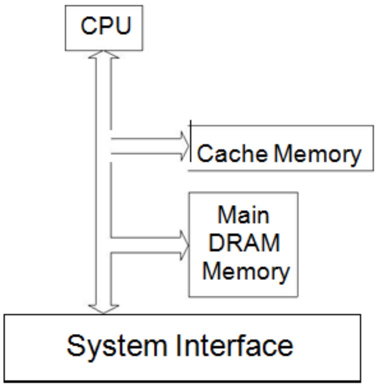

Figure 1. *Basic cache representation.*

This paper is organized as follows: sections I, briefly discuss an overview of cache memory. Section II; introduce the basics of cache memory as related to computer main memory. Section III, discuss cache portioning as used in multicore processor. Section IV; gives an understanding of multi-core chips cache management. Section V, discuss cache performance. Section VI conclude this paper.

3. Cache Partitioning

Cache partitioning is crucial to the efficient uses of multicore processors [6]. Cache partitioning can be emphasized to be the division of joint caches among a series of programming threads running concurrently on diverse cores. Most public multicore processors used this days still use cache designs from uniprocessors, which does not reflect on the meddling among multiple cores. Meanwhile, an amount of cache partitioning technique has been projected with diverse optimization aims, as well as implementation, equality, and eminence of service. Most obtainable studies, including the ones cited above, were estimated by replication. Though replication is adapted, it acquires different restriction in assessing cache partitioning method. The most accepted one is the decelerated replication speed; it is infeasible to run great, complex and dynamic real-world programs to completion on a cycle-accurate simulator [7]. A standard simulation-based research may only assume a few billion directives for a program, which is coequal to about one second of performance on a real machine. The complex arrangement and dynamic conduct of simultaneously running programs can be hardly portrayed by a short execution. Moreover, the effect of operating systems can hardly be assessed in simulation-based studies because the full influence cannot be noticed in a short simulation time. This restriction may not be the most serious care for microprocessor design, but is becoming increasingly relative to system architecture design. In addition, careful measurements on real machines are reliable, while evaluations on simulators are prone to inaccuracy and coding errors.

4. Multi-Core Chips Cache Management

Multiple cores chip cache management consider cache memory as whether the caches should be shared or local to each core in any system [8]. Realising shared cache certainly introduces more cabling and difficulties. But then, having one cache per chip, rather than core, significantly reduces the amount of space needed, and thus one can include a larger cache in the memory to awfully optimize the storage.

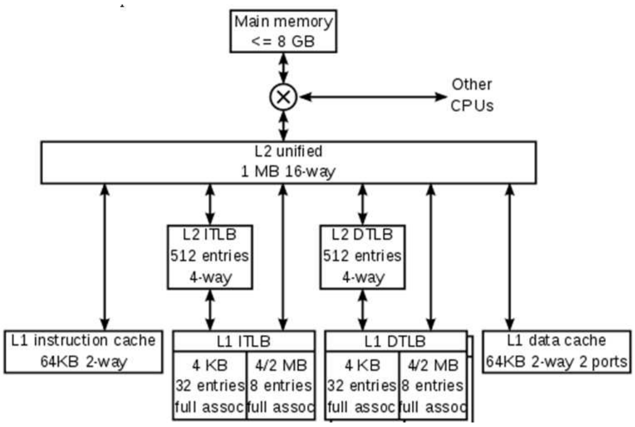
Main memory <= 8 GB
Other CPUs
L2 unified 1 MB 16-way
L2 ITLB 512 entries 4-way
L2 DTLB 512 entries 4-way
L1 instruction cache 64KB 2-way
L1 ITLB
4 KB 32 entries full assoc
4/2 MB 8 entries full assoc
L1 DTLB
4 KB 32 entries full assoc
4/2 MB 8 entries full assoc
L1 data cache 64KB 2-way 2 ports

Figure 2. *K8 core Cache hierarchy of AMD Athlon 64 CPU.*

Normally, distributing the L1 cache in the memory is adversesince the resulting increase in latency would make each core run very much slower than a single-core chip. Though, for the highest-level cache, the last one called before accessing memory, having a global cache is desirable for several reasons, such as allowing a single core to use the whole cache, reducing data redundancy by making it possible for different processes or threads to share cached data, and reducing the convolution of utilized cache consistent protocols [9]. For instance, an eight-core chip with three levels may include an L1 cache for each core, one intermediate L2 cache for each pair of cores, and one L3 cache shared between all cores.

a. Multi-level and Specialisations cache of K8 AMD Athlon 64 CPU

b. The following are K8 cache memory

1. An instruction cache
2. An instruction TLB
3. Data TLB
4. Data cache

1. An instruction cache

The instruction cache maintains copies of 64-byte lines of memory, and fetches 16 bytes each cycle. Each byte in this cache is stored in ten bits rather than eight, with the extra bits marking the boundaries of instructions. The cache has only parity protection rather than ECC, because parity is smaller and any damaged data can be replaced by fresh data fetched from memory.

2. The instruction TLB

This maintains copies of page table entries (PTEs). Each cycle's instruction fetch has its virtual address translated through this TLB into a physical address. Each entry is either four or eight bytes in memory. Because the K8 has a variable page size, each of the TLBs is split into two sections, one to keep PTEs that map 4 KB pages, and one to keep PTEs that map 4 MB or 2 MB pages. The split allows the fully associative match circuitry in each section to be simpler. The operating system maps different sections of the virtual address space with different size PTEs.

3. The data TLB

The data TLB has two copies which maintain identical entries. The two copies allow two data accesses per cycle to translate virtual addresses to physical addresses. Like the instruction TLB, this TLB is split into two kinds of entries.

4. The data cache

The data cache maintains copies of 64-byte lines of memory. It is split into 8 banks (each storing 8 KB of data), and can fetch two 8-byte data each cycle so long as those data are in different banks. There are two copies of the tags, because each 64-byte line is spread among all eight banks. Each tag copy handles one of the two accesses per cycle.

c. Advantages of K8 core cache

The following are areas were multi-core cache memory in a central processing unit can be beneficial to the system.

1. Multiple-level caches

K8 core cache is characterized with multiple-level caches. There are second-level instruction and data TLBs, which store only PTEs mapping 4 KB. Both instruction and data caches, and the various TLBs, can fill from the large unified L2 cache. This cache is exclusive to both the L1 instruction and data caches, which means that any 8-byte line can only be in one of the L1 instruction cache, the L1 data cache, or the L2 cache. It is, however, possible for a line in the data cache to have a PTE which is also in one of the TLBs—the operating system is responsible for keeping the TLBs coherent by flushing portions of them when the page tables in memory are updated.

2. Prediction information

Another K8 core cache advantages is that it can predict information stored in the memory. This aspect of the cache is not shown in the above diagram. In this class of CPU, the K8 has fairly intricate branch prediction, with tables that help predict whether branches are taken and other tables which predict the targets of branches and jumps. Some of this information is associated with instructions, in both the level 1 instruction cache and the unified secondary cache.

3. Trap policy

The trap policy of the K8 core cache cannot be overwhelmed, because ituses an interesting trick to store prediction information with instructions in the secondary cache. Lines in the secondary cache are protected from accidental data corruption, by either ECC or parity, depending on whether those lines were evicted from the data or instruction primary caches. Since the parity code takes fewer bits than the ECC code, lines from the instruction cache have a few spare bits. These bits are used to cache branch prediction information associated with those instructions. The net result is that the branch predictor has a larger effective history table, and so has better accuracy.

Cache algorithms

Cache reads are the most common CPU operation that takes more than a single cycle [10]. Program execution time tends to be very sensitive to the latency of a level-1 data cache hit. A great deal of design effort, and often power and silicon area are expended making the caches as fast as possible.

The simplest cache is a virtually indexed direct-mapped cache [11]. The virtual address is calculated with an adder, the relevant portion of the address extracted and used to index an SRAM, which returns the loaded data. The data is byte aligned in a byte shifter, and from there is bypassed to the next operation. There is no need for any tag checking in the inner loop – in fact, the tags need not even be read. Later in the pipeline, but before the load instruction is retired, the tag for the loaded data must be read, and checked against the virtual address to make sure there was a cache hit. On a miss, the cache is updated with the requested cache line and the pipeline is restarted.

An associative cache is more problematic, because some form of tag must be read to determine which entry of the cache to select. An N-way set-associative level-1 cache usually reads all N possible tags and N data in parallel, and then chooses the data associated with the matching tag. Level-2 caches sometimes save power by reading the tags first, so that only one data element is read from the data SRAM.

5. Cache Performance

Cache performance measurement has become imperative in modern times where the speed gap between the memory and processor performance is increasing exponentially [12]. Therefore, performance of a cache can be quantified in terms of the hit cost, miss rates, and the miss penalty, where a cache hit is a memory access that finds data in the cache and a cache miss is one that does not find data in the cache and the cost of a cache hit is roughly the time to access an entry in the cache, miss penalty is the additional cost of replacing a cache line with one containing the desired data. Table 1 shows the relationship improvement in the instruction code of a data size as it increase from 1kilobyte to 128 kilobytes, data cache and unified cache also shows relative improvement in percentage.

Access time = hit cost + miss rate * miss penalty= Fast memory access time + miss rate * slow memory access time

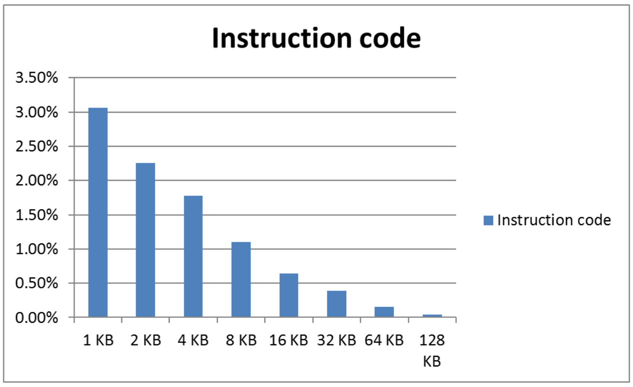

Figure 3. *Instruction code analysis.*

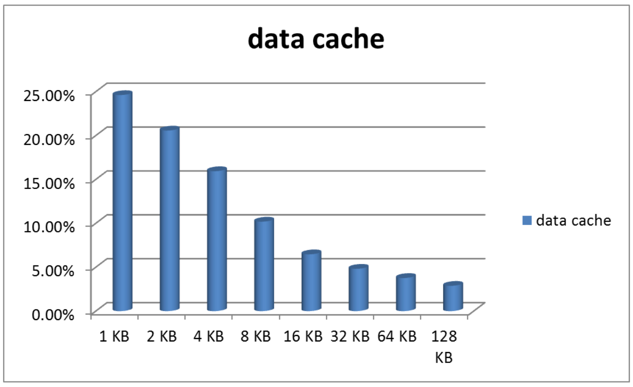

Figure 4. *Data cache analysis.*

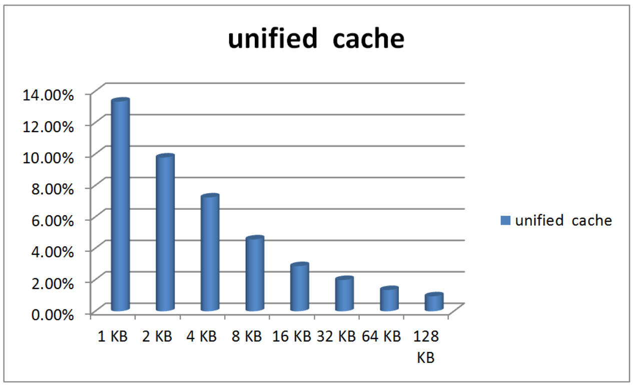

Figure 5. *Unified cache analysis.*

Table 1. *Analysis of an improved performance of a cache memory.*

Data size	Instruction code	Data cache	Unified cache
1 KB	3.06%	24.61%	13.34%
2 KB	2.26%	20.57%	9.78%
4 KB	1.78%	15.94%	7.24%
8 KB	1.10%	10.19%	4.57%
16 KB	0.64%	6.47%	2.87%
32 KB	0.39%	4.82%	1.99%
64 KB	0.15%	3.77%	1.35%
128 KB	0.02%	2.88%	0.95%

6. Conclusion

Cache is basically a high speed piece of memory that stores a snapshot of main memory which facilitates the processor higher performance. Caches are applied in different methods in the memory for better performance, though the basic perceptions of caches are the same. From table 1 above it shows that as the data size increases the instruction code are reducing whilst the data cache are also reducing hence better and improved performance. However, the features in the Pentium (R) Processors implementation cross several of the initially defined boundaries. It is important to understand that the Pentium (R) Processor uses only one method to implement cache.

References

[1] Kumar, M. A., & Francis, G. A. (2017, February). Survey on various advanced technique for cache optimization methods for risc based system architecture. In 2017 4th International Conference on Electronics and Communication Systems (ICECS) (pp. 195-200). IEEE.

[2] Nagasako, Y., & Yamaguchi, S. (2011, March). A server cache size aware cache replacement algorithm for block level network Storage. In 2011 Tenth International Symposium on Autonomous Decentralized Systems (pp. 573-576). IEEE.

[3] Chang, M. T., Rosenfeld, P., Lu, S. L., & Jacob, B. (2013, February). Technology comparison for large last-level caches (L 3 Cs): Low-leakage SRAM, low write-energy STT-RAM, and refresh-optimized edram. In 2013 IEEE 19th International Symposium on High Performance Computer Architecture (HPCA) (pp. 143-154). IEEE.

[4] Bao, W., Krishnamoorthy, S., Pouchet, L. N., &Sadayappan, P. (2017). Analytical modeling of cache behavior for affine programs. Proceedings of the ACM on Programming Languages, 2 (POPL), 1-26.

[5] Kosmidis, L., Abella, J., Quiñones, E., &Cazorla, F. J. (2013, March). A cache design for probabilistically analysable real-time systems. In 2013 Design, Automation & Test in Europe Conference & Exhibition (DATE) (pp. 513-518). IEEE.

[6] Wang, Y., Ferraiuolo, A., Zhang, D., Myers, A. C., & Suh, G. E. (2016, June). SecDCP: secure dynamic cache partitioning for efficient timing channel protection. In Proceedings of the 53rd Annual Design Automation Conference (pp. 1-6).

[7] Zhang, M., Zhuo, Y., Wang, C., Gao, M., Wu, Y., Chen, K.,...& Qian, X. (2018, February). GraphP: Reducing communication for PIM-based graph processing with efficient data partition. In 2018 IEEE International Symposium on High Performance Computer Architecture (HPCA) (pp. 544-557). IEEE.

[8] Barroso, L. A., Gharachorloo, K., McNamara, R., Nowatzyk, A., Qadeer, S., Sano, B.,... & Verghese, B. (2000). Piranha: A scalable architecture based on single-chip multiprocessing. ACM SIGARCH Computer Architecture News, 28 (2), 282-293.

[9] Chetlur, S., & Catanzaro, B. (2019). U.S. Patent No. 10,223,333. Washington, DC: U.S. Patent and Trademark Office.

[10] LiKamWa, R., &Zhong, L. (2015, May). Starfish: Efficient concurrency support for computer vision applications. In Proceedings of the 13th Annual International Conference on Mobile Systems, Applications, and Services (pp. 213-226).

[11] Dayalan, K., Ozsoy, M., &Ponomarev, D. (2014, October). Dynamic associative caches: Reducing dynamic energy of first level caches. In 2014 IEEE 32nd International Conference on Computer Design (ICCD) (pp. 118-124). IEEE.

[12] Mittal, S. (2017). A survey of techniques for cache partitioning in multicore processors. ACM Computing Surveys (CSUR), 50 (2), 1-39.