Jan. 31

International Journal of Advanced Studies in Computer Science and Engineering

IJASCSE, Volume 3, Issue 1, 2014

Architectures and Technologies of Cache Memory: A Survey

Jatau Isaac Katuka, Gabriel Lazarus Dams, and Salome Danjuma

[1,2,3]Department of Mathematical Sciences, Kaduna State University.

***Abstract*— Cache is a memory in between the processor and the main memory. It is a smaller temporary memory that manages the main memory locations and access time thereby increases speed during execution time. Cache memory is located on the processor chip which consists of L1 and L2 cache. This paper describes the types of cache and how they can improve the system performance and speed in terms of data accesses. Detailed steps involved in data transfer between the cache and memory were also discussed. We also explored some new cache technologies like: Lower power with higher performance architectures, multi core processors and caching which increases multi-tasking, web cache which store previous responses from web servers, and hybrid cache architecture which deals with integrating different memory types on the same micro-processor thereby improving the performance of microprocessor in terms of power consumption and processing speed. Survey on non-uniform cache architecture concludes the paper which is a future approach to a large number of core processors.**

Keywords- cache; L1; L2; latency; architecture; instruction; data; memory; embedded system; low energy; web; hybrid.

I. INTRODUCTION

The cache memory is a very small but fast kind of memory which serves as an intermediary between the main memory and the processor in an effort to reduce the number of times the main memory is accessed by programs. The reduction of the average time of main memory access is achieved by the cache through storing copies of the data from the most frequently used locations of the main memory. For example, when the need arises for the processor to read from or write to a main memory location, the processor first performs a check to see whether there is a copy of that data in the cache. When the processor finds a copy of the data in the cache, it reads from or writes to the cache immediately which will turn out to be faster (for the processor) than reading from or writing to the main memory directly. Using the cache memory in this manner aids in the reduction of the average latency of memory accesses [1].

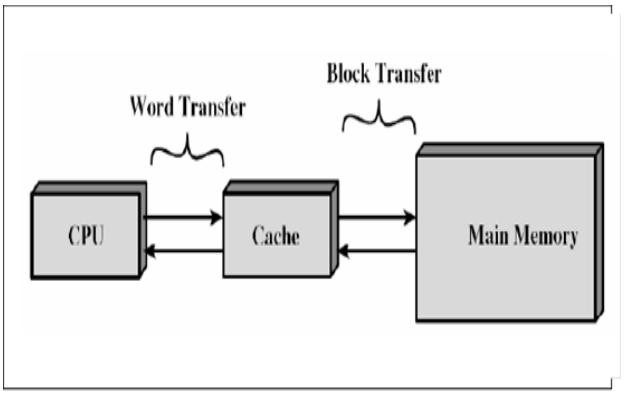

Figure 1: Position of cache

For understanding the structure and working concept of the cache, the Intel Pentium 4 processor can be considered as an example. The processor (Intel Pentium 4) uses two cache levels (L1 and L2). L1 is referred to as the Level 1cache

which is 8KB and 16KB while the L2 is referred to as the Level 2 cache which has a size of 1MB. With the advancement in technology, the sizes are continuously pushed up making it not uncommon to find an L2 size of 4MB. In an effort to separate the data from the instructions in the L1 cache, it would be found that the existence of two L1 caches per processor/core is also quite common. However, the L2's arrangement is in a way that both data and the instructions are together. The make-up of the L2 is known as unified. Modern desktop and servers mostly have at least three independent caches: an instruction cache, a data cache and a translation-look-aside-buffer (TLB).

The instruction cache aids in speeding up the fetched executable instructions; the data cache aids in speeding up the fetching and storing of data and the TLB aids in speeding up virtual-to-physical address translations which are both for the executable instructions and the data.
Higher computing power is increasing with the advancement of information technology (IT) and as such, there is more increase in the on-chip computing resources. To keep up with developments and technological implementations, the size of the on-chip cache memory has been increasing. Despite this increase and advancements, some applications on the contrary may not utilize the full cache capacity but rather would require more computing resources.

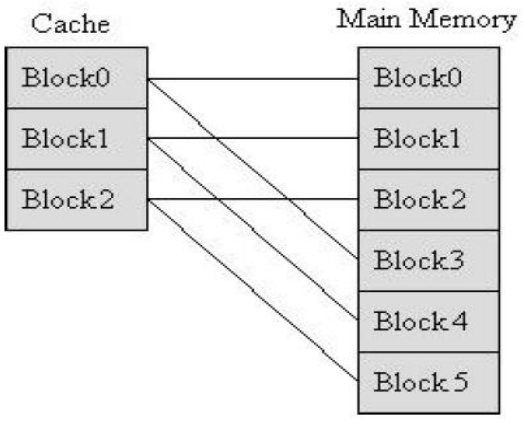

Figure 2: Direct mapped cache, from [3]

II. CACHE ENTRIES

The transfer of data between the main memory and the cache occurs in blocks of fixed sizes which are known as cache lines. Cache entries are created when cache lines are copied from the main memory to the cache which includes the copied data together with the requested memory location. A check for corresponding entry in the cache is first performed by the processor when it needs to read or write a location in the main memory. A check is also performed by the cache for the requested memory contents from any cache lines which contains that address. On finding the memory location in the cache by the processor, the occurrence is referred to as cache hit. However, if the processor doesn't find the location of the memory in the cache, the occurrence is referred to as cache miss. This implies in other words that the processor reads or writes the data in the cache line immediately when there is a cache hit while the allocation of a new entry and copying of data from the main memory by the cache is performed in the occurrence of a cache miss to fulfill the request in the contents of the cache [1] [2].

III. CA CHE PERFORMANCE

The proportion of accesses resulting from a cache hit is referred to as hit rate [2]. The hit rate can be a measure of the cache's effectiveness for a given algorithm or program. When a data transfer is needed from the main memory much more slowly than the cache itself, the read misses delay execution while the write might take place without such penalty, since execution could be continued by the processor as data – in the background – is copied to the main memory.
A datum which is written to the cache by a system has to at certain points write that datum as well to backing store. The "*write policy*" controls the timing of the write. Moreover, the writing has two

basic approaches which are: *write through* and *write-back* or *write-behind*.
The writing in the *write-through* approach is synchronously performed to both the cache and the backing store. The *write-back* which is also known as the *write-behind* approach initially writes to the cache while postponing the backing store writing until new contents are about to modify/replace the cache blocks containing the data.

The write-back cache is complex because it needs a number of things to do in order to be successfully implemented. Things such as tracking the location which it has been written over are performed. The locations which are tracked are then marked as "*dirty*" for later writing which would be performed to the backing store. The only condition for writing back the data to the backing in these locations is when eviction of the locations occurs from the cache; the result of this occurrence is termed as *lazy write*. Consequent to that, two (2) memory accesses to service would often be needed by a read miss located at the write-back cache which is: writing the modified/replaced data from the cache to the backing store and retrieving the datum which is needed. During the writing operations, two (2) approaches exist for write-misses situation (since no actual data is required back):

- **Write allocate** – The write allocate approach is also known as "*Fetch on write*" where datum which is located at the missed-write would be loaded to the cache and afterwards, the write-hit operation will follow. There is similarity between the write misses and the read-misses in this approach.

- **No-write allocate** – In this approach also known as the "*write-no-allocate, write-around*", only the system reads are being cached. More so, the datum located at the missed-write is written directly to the backing store and not loaded to cache.

The write-miss policies according to Hennessy and Patterson [2] and Null and Lobur [3] can be used either by both write-back and write- through policies; however they are usually paired in this way:

- *Write allocate* is used by a write-back cache while anticipation subsequent writes (or even reads) to the same location, which is now cached.

- *No-write allocate* on the other hand is used by a write-through cache. In this policy, the need for direct writing to the backing store make subsequent writes to have no advantage.

IV. ENERGY CONSUMPTION OF CACHE

The major concern of the cache design is the consumption of energy [4]. Several studies have shown that the total energy used in embedded systems (amounting to 50%) is attributed to the cache memory [5, 6]. The application using the cache determines the performance of the cache architecture. Given the current applications, technology and cost, the manufacturers of desktop applications usually set the architecture of the cache as a compromise because the desktop systems accommodate a very wide range of applications.

The design of embedded systems are made in such a way that they run a small range of well-defined applications which are unlike the desktop applications. Cache architecture in this context which is adjusted for that small range of applications can yield both high performance and consumes less energy as well. A novel cache architecture intended for embedded microprocessor platforms was introduced byZhang, F. [4] using the *way concatenation* technique – technique for configuring the cache – configuration of the cache can be done with a software which could be direct-mapped, two way, or four-way set associative having very little performance overhead or size.

Asaduzzaman [7] mentioned that as much as the cache reduces the speed-gap between the CPU and the main memory thereby improving the performance of the cache, the dynamic nature of the cache causes increase in the timings unpredictably. Besides that, significant amount of power is needed by the cache to operate. The presence of multiple levels of caches makes the power consumption and unpredictable timing in multi-core systems worse.

It has been indicated in recent studies [8-10] that the total power consumption in multi-core systems can be decreased while increasing predictability by locking appropriate memory blocks without compromising performance. Selecting the accurate blocks to be locked determines the success of the cache. *Cache locking* techniques should lock the memory blocks with higher cache misses and with the use of BAMI, victim blocks with lower cache misses should be selected through *cache replacement policy* [7].

The approximate consumption of a direct mapped cache (considered to be more efficient in energy consumption per access) is 30% compared to the energy of same sized four-way set associative cache [11]. Accessing only one tag and data array per access in direct mapped cache is the cause of this reduction while four-way cache access is four tag and data array per access. Shorter access time can also take place in direct mapped cache because multiple data arrays do not need to be multiplexed.

V. WEB CACHE

Web pages and images from previous responses of web servers are stored in web caches been employed by web proxy servers and web browsers. The amount of information required to transmit across the network is reduced by web caches as information stored in the cache previously can often be reused. This process results in the reduction of processing requirements and bandwidth of the web server which collectively aids in the improvement of the responsiveness for web users [12].
The technology of web caching has been widely used to decrease the perceived network latencies of users and to improve the web infrastructure performance. The major technique used in web caching which endeavors (from a network of proxies) to serve user web requests between the end users and the web server (that hosts the requested object's original copies) is known as *proxy caching* [13].

The lagging of the internet bandwidth expansion has caused a major bottleneck in the web in terms of performance. Without proper attention given to expansion, the gap which exists between the infrastructure and the demand on the web would continuously exist. Web services, which is also another recent development in web technologies is consuming the network bandwidth through potentially bringing new classes of applications been distributed in large numbers enabling means of communication amongst web users [14].

In order to meet the important web capacity challenges, the web caching has been established as an approach to address the related issues such as latencies of user-perceived network. According to Zeng, Wang [12] web caching has multifold potential benefits which can be viewed from different perspectives. Viewing from the end-users perspective, it can lessen the latency of the user perceived network significantly and make their web experiences improved. Viewing from the internet infrastructure perspective, reducing the web traffic amount can be achieved through web caching thereby improving the overall internet performance and minimizing the congestion on the network. In addition, reduced traffic would imply lowered costs for enterprises which would pay the internet service providers (ISPs) for wide area network bandwidth based on the amount of network traffic.

Viewing from the web server perspective, significant reduction in the server loads can be achieved via caching to improve the responsiveness of the system. Since copies of web objects are kept on the network throughout, even when the original server hosting them are down, users would be able to have access to the web objects. Value-added services such as security, content filtering and advertisement which are not related directly to caching have been hosted using caching servers in recent years [15].

VI. HYBRID CACHE

Performance of the microprocessor in terms of processing speed and consumption of power can be improved using the architecture of hybrid cache. The structure of each memory has the area which at least satisfies the Average Memory Access Time (AMAT) under a given area condition. The architecture of the hybrid cache which compose of the static random access memory (SRAM) and magnetic random access memory (MRAM) shows that 16.9% reduction in AMAT and 15.2% of reduction in power compared with the cache which is composed of the homogenous SRAM. The SRAM and the dynamic random access memory (DRAM) structure represents 33.0% reduction in the consumption of power while that of SRAM and phase change random access memory (PRAM) shows a potential in the reduction of area and consumption of power as a result of the RAM's (SRAM and PRAM) high density [16].

VII. NON-UNIFORM CACHE ARCHITECTURE

The non-uniform cache architecture (NUCA) is a new approach that is being discussed for the future of many-core processors. These NUCA memories are an alternative to increase the performance of processors that have a very large amount of processing cores and require a lot of data bandwidth. In the case of shared memory, one can point the access latency as the main problem for the nowadays uniform cache architectures.

The NUCA memories can reduce the latency of data access, increasing scalability, thus, these memories have a great potential for increasing performance on multi-core and many-core processors. Nowadays, multiple levels of cache memories are organized in a few discrete levels. Typically, each lower level includes and replicates the content of above levels, thereby reducing access time to levels closer to the main memory. For future technologies, large memory cache within the chip, with a single data access time is not appropriate, since problems related to the wire-delay inside the chip are increasing.

This is the main argument for creation of new non-uniform cache architecture. Besides the wire-delay problem, we can cite the important behavior of possible components failures along time, that next generation processors shall have [17]. Thus, some isolation or redundancy on the memory modules and interconnection can ensure that if any failure occurs the other components will keep working.

The main concept involving these new non-uniform cache architectures is that some data will reside closer to the processor, thus, these data will have some fast access times. This way, near data will be accessed faster than data which are distant from the processor. Moreover, problems associated with the number of ports and hardware failure can be partly solved by a non-centralized architecture.

The data mapping policy in NUCA memory according to Sebek and Gustafsson [9] defines the mapping of data into banks and it also defines the banks that the data can reside. Planning for different mapping policies can be done since the bank usage in the NUCA memory is flexible.

Even with the flexibility given by the existence of several memory banks, only one static mapping is

provided for each address by the static non-uniform cache architecture (S-NUCA) which implies that each sub-bank is associated with several addresses as in the uniform cache architecture (UCA) memory. Much of the NUCA memory advantages would be lost with this strategy and more so, performance bottlenecks would be created as some data would always have high access time depending on the static mapping used [18].

CONCLUSIONS

This paper describes the types of cache and how they can improve the system performance and speed in terms of data accesses. Detailed steps involved in data transfer between the cache and memory were also discussed. Cache entries are created when cache lines are copied from the main memory to the cache which includes the copied data together with the requested memory location. The hit rate can be a measure of the cache's effectiveness for a given algorithm or program.

When a data transfer is needed from the main memory much more slowly than the cache itself, the read misses delay execution while the write might take place without such penalty, since execution could be continued by the processor as data – in the background – is copied to the main memory. The major concern of the cache design is the consumption of energy. The application using the cache determines the performance of the cache architecture. Web pages and images from previous responses of web servers are stored in web caches been employed by web proxy servers and web browsers.

The amount of information required to transmit across the network is reduced by web caches as information stored in the cache previously can often be reused. This process results in the reduction of processing requirements and bandwidth of the web server which collectively aids in the improvement of the responsiveness for web users.

REFERENCES

1. Heuring, V.P. and H.F. Jordan, *Computer Systems Design and Architecture (2nd Edition)*. 2003: Prentice-Hall, Inc.
2. Hennessy, J.L. and D.A. Patterson, *Computer Architecture, Fifth Edition: A Quantitative Approach* 5th Edition ed. 2011, USA: Morgan Kaufmann Publishers.
3. Null, L. and J. Lobur, *The Essentials of Computer Organization and Architecture 2nd Edition*. 2006, Canada: Jones and Bartlett Publishers, Inc.
4. Zhang, C., V. F., and N. W. *A highly configurable cache architecture for embedded systems*. in *Computer Architecture, 2003. Proceedings. 30th Annual International Symposium on*. 2003. IEEE.
5. Malik, A., B. Moyer, and D. Cermak. *A low power unified cache architecture providing power and performance flexibility*. in *Low Power Electronics and Design, 2000. ISLPED '00. Proceedings of the 2000 International Symposium on*. 2000. IEEE.
6. Segars, S. *Low power design techniques for microprocessors*. in *International Solid-State Circuits Conference Tutorial*. 2001.
7. Asaduzzaman, A. *An efficient memory block selection strategy to improve the performance of cache memory subsystem*. in *Computer and Information Technology (ICCIT), 2011 14th International Conference on*. 2011. IEEE.
8. Asaduzzaman, A., I. Mahgoub, and F.N. Sibai. *Impact of L1 entire locking and L2*

way locking on the performance, power consumption, and predictability of multicore real-time systems. in *Computer Systems and Applications, 2009. AICCSA 2009. IEEE/ACS International Conference on*. 2009. Morocco: IEEE.

9. Sebek, F. and J. Gustafsson. *Determining the Worst-Case Instruction Cache Miss-Ratio*. 2002. sweden.
10. Molnos, A.M., et al. *Data Cache Optimization in Multimedia Applications*. in *Proceedings of the 14th Annual Workshop on Circuits, Systems and Signal Processing (ProRISC'03)*. 2003. Veldhoven, The Netherlands.
11. Glen, R. and N.P. Jouppi. *CACTI2.0: An Integrated Cache Timing and Power Model*. in *COMPAQ Western Research Lab*. 1999.
12. Zeng, D., F.-Y. Wang, and M. Liu, *Efficient web content delivery using proxy caching techniques.* Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on, 2004. **34**(3): p. pp. 270-280 (3).
13. Rabinovich, M. and O. Spatschek, *Web caching and replication*. 2002: Addison-Wesley Longman Publishing Co., Inc. 361.
14. Takase, T., et al., *A web services cache architecture based on xml canonicalization*, in *In Proceedings 11th Int. World Wide Web Conf. (Poster Paper)*2002, HI: Honolulu.
15. Brooks, C., et al., *Application-specific proxy servers as http stream transducers*, in *In Proceedings 4thWorldWide Conference*1995. p. pp. 539–548.
16. Lee, S., J. Jung, and C.-M. Kyung. *Hybrid cache architecture replacing SRAM cache with future memory technology*. in *Circuits and Systems (ISCAS), 2012 IEEE International Symposium on*. 2012. IEEE.
17. Aggarwal, N., et al., *Isolation in Commodity Multicore Processors.* Computer, 2007. **40 (6)**(6): p. pp. 49-59.
18. Alves, M.A.Z., H.C. Freitas, and P.O.A. Navaux. *Investigation of Shared L2 Cache on Many-Core Processors*. in *Architecture of Computing Systems (ARCS), 2009 22nd International Conference on*. 2009. Berlin: IEEE.