Analyzing and Optimizing Cache Memory and Performance

Sai Keerthan Palavarapu, Divija Morishetty, Sai Vennela Gowreddy and Charitha Tadakanti
Syracuse University, spalavar, dmorishe, sgowredd, ctadakan@syr.edu

Abstract - **Cache memory plays a pivotal role in enhancing the overall performance of computer systems by bridging the speed gap between the high-speed CPU and slower main memory. This paper presents a comprehensive analysis of performance issues related to cache memory in modern computing systems. The study delves into various aspects, including cache organization, replacement policies, and coherence protocols, to identify potential bottlenecks and challenges that impact system efficiency. The analysis begins by examining the fundamental principles of cache design and the trade-offs involved in selecting cache parameters. It explores the impact of cache size, associativity, and block size on hit/miss ratios and their subsequent influence on overall system performance. The investigation extends to different cache replacement policies, evaluating their effectiveness in minimizing cache misses and optimizing data retrieval. Furthermore, the paper addresses architecture circumscribed with three improvement techniques namely victim cache, sub-blocks, and memory bank. These three techniques will be implemented one after other to improve and make the speed and performance of cache comparative to main memory. Moreover, the different variables like miss penalty ratio, access speed of cache and miss rate ratio, which were already in use, are used in this paper to estimate the cache memory performance after implementation of proposed approach. After performance estimation it can be determined that proposed approach at level 1, using Victim Cache technique decreases the rate of misses, at level 2, Subblocks Division technique further reduces the penalty ratio of miss rate and then at level 3 Memory Bank Technique is useful in further decreasing memory access time. Thus, using the suggested approach, performance of Cache Memory can be improved several times.**

Index Terms - Direct Mapping, Miss Penalty Ratio, RAM, Tags and Addresses, Victim Cache.

INTRODUCTION

The computer's cache memory, which has a shorter access time than other memories, is a crucial component that may have an impact on how well an application runs. It approaches the speed of CPU components and is the fastest part of the memory hierarchy. Modern day systems have levels of cache. Generally, there are three levels of cache used in today's system architecture.

Types of Cache Memory:

- **L1**: It is the first level of cache memory, which is called Level 1 cache or L1 cache. In this type of cache memory, a small amount of memory is present inside the CPU itself. The L1 cache further has two types of caches: Instruction cache, which stores instructions required by the CPU, and the data cache that stores the data required by the CPU.
- **L2**: This cache is known as Level 2 cache or L2 cache. This level 2 cache may be inside the CPU or outside the CPU. In terms of speed, they are slower than the L1 cache.
- **L3**: It is known as Level 3 cache or L3 cache. This cache is not present in all the processors; some high-end processors may have this type of cache. This cache is used to enhance the performance of Level 1 and Level 2 cache.

When CPU needs the data, first, it looks inside the L1 cache. If it does not find anything in L1, it looks inside the L2 cache. If again, it does not find the data in L2 cache, it investigates the L3 cache. If data is found in the cache memory, then it is known as a cache hit. On the contrary, if data is not found inside the cache, it is called a cache miss. If data is not available in any of the cache memories, it looks inside the Random Access Memory (RAM). If RAM also does not have the data, then it will get that data from the Hard Disk Drive.

So, when a computer is started for the first time, or an application is opened for the first time, data is not available in cache memory or in RAM. In this case, the CPU gets the data directly from the hard disk drive. Thereafter, when you start your computer or open an application, CPU can get that data from cache memory or RAM.

The HIT ratio closer to one can be taken as on:

- If the memory address is accessed first time, then Misses will take place.
- If 2 blocks are simultaneously mapped on the same address, Misses could occur because of the inadequate size. Misses could occur because of the small size of cache.

Established metrics around miss penalty/access speed ratios benchmark cache gains from the proposed approach. Enhanced on-chip cache retention and intermediate data serving remain essential to realize the full potential of multi-core systems, where cache plays an increasingly crucial role.

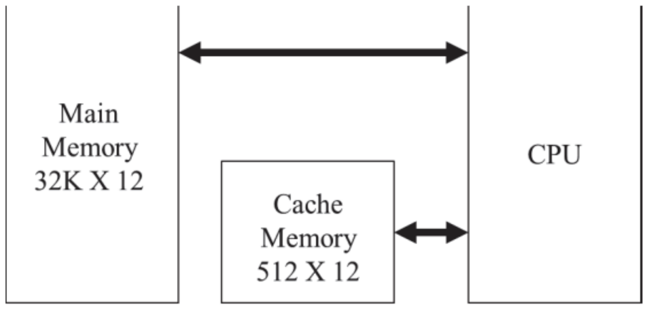

FIGURE I
LOCATION OF CACHE MEMORY

This research investigates boosting cache speeds through three simultaneous enhancement techniques - victim cache, sub-blocks, and memory bank modifications - implemented sequentially at each cache level. The techniques aim to narrow the processor-memory speed gap by aligning accelerated cache memory closer to main memory performance.

Established metrics around miss penalty/access speed ratios benchmark cache gains from the proposed approach. Enhanced on-chip cache retention and intermediate data serving remain essential to realize the full potential of multi-core systems, where cache plays an increasingly crucial role.

BACKGROUND

Cache mapping is the process of deciding where in the cache to store data from the main memory. Different mapping techniques dictate how this decision is made. They are:

- **Direct Mapping** - The Simple Road: Direct mapping is like having a specific parking spot assigned to each block of data. It's a straightforward approach that makes organization simpler.
- **Associative Mapping** - The Flexible Approach: It is like having a very flexible system. In this approach, any block of data from the main memory can be placed in any cache location. It's almost as if each block has a free pass to be stored wherever it wants in the cache.
- **Set-Associative Mapping** - The Balanced Approach: Set Associative Mapping is like finding a middle ground between the flexibility of Associative Mapping and the simplicity of Direct Mapping. It strikes a balance by organizing the cache into sets, and each set contains multiple lines or slots for storing data.

LITERATURE REVIEW

Various researches have shown how to improve cache performance. V.Chaplot[1] has proposed an approach that can provide higher associativity, this enhancement contributes to lowering cache miss rates. In the discussion, two specific aspects of data behavior have been explored: temporal locality, which focuses on how frequently data is accessed over time. Later C.Zhang[2] and A.Agarwal[3] have proposed automatic self-adaptable cache memory models for memory systems by introducing on-chip hardware an efficient cache tuning heuristic that can automatically, transparently, and dynamically tune the cache to an executing program. A number of improvements have been made recently to improve cache memory performance, taking into account variables including hit rate, latency, speed, energy consumption and replacement policies, is presented in [4] and [5]. Cache replacement policy is one of the significant design factors that affects processor performance as a whole. In cache memory, when all the lines in a set of cache become full and a new block from main-memory needs to be placed in cache, then the cache controller has to be discard a line from cache set and replace it with the new block from main-memory. The modern processors employ cache replacement policies such as LRU (Least Recently Used), Random, FIFO (First in First Out), LFU (Least Frequently Used). In all these policies, except Random, determine which block of cache memory to replace by looking only at the past references . The Least Recently Used replacement policy replaces the cache line that has been least recently used (a block in the set that has been in cache for the longest and has no reference to it) with a cache controller by using the access pattern of a program memory. In FIFO objects are added to the queue and are evicted with the same order. It provides a simple and low-cost method to manage the cache as the most used objects are eventually evicted when they're old enough. The LFU algorithm functions similarly to the LRU method, except instead of tracking the frequency of an object's access, it counts the number of times it was accessed.

K. Westerholz[6] has put forward proposals for implementing cache designs and enhancing the efficiency of cache-driven memory management to facilitate communication between the CPU and main memory. Divya [7] has introduced a practical and effective code for optimizing virtual memory [7]. In [8] and [9], an approach to reduce average memory access time is suggested by incorporating a victim cache. Kim and Song have provided a detailed analysis of the impact of cache memory on data storage [10]. Some researchers have explored the performance of cache memory in conjunction with other memory types such as scratchpad memory, split memory, etc. [11], [12]. Kolanski [13] has advocated for the utilization of cache memory in mobile and embedded systems. Smith et al. [14] have proposed a performance-oriented approach for virtual memory using a page-based method, influenced by page fault frequency. Furthermore, Banday and Khan [15] have presented a comprehensive overview and review of the current status of cache memory performance. They have also delved into discussions about cache performance and its adaptability, especially when the system is operating over a network.

SIGNIFICANCE

Cache memory plays a pivotal role in business decision-making processes by significantly enhancing the speed and efficiency of computer systems. In the modern business landscape, characterized by the need for data-driven decisions, cache memory provides distinct benefits that directly influence the decision-making process.

For example, As per “The State of Online Retail Performance” report by Akamai, Delay of 100 milli seconds in load time experienced up to 7% decreased in conversion rates. Also, Delay of 2 seconds in load time, experienced bounce rates by 103%.

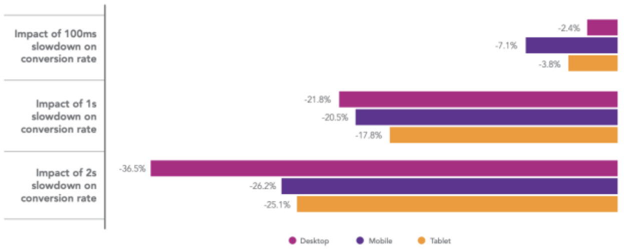

FIGURE II

IMPACT OF PAGE SLOWDOWNS ON CONVERSION RATES (BY DEVICE TYPE)

The above example implies, cache memory plays a crucial role in enhancing customer experience. In businesses offering online services or e-commerce platforms, cache ensures that web pages load swiftly and that customers can access products, services, and information without delay. Providing a smooth user experience is crucial for achieving high customer satisfaction, driving greater sales, and ensuring repeat business - all pivotal elements for a thriving enterprise.

PROBLEM ANALYSIS

In the realm of computing, cache memory stands as a primary source of data retrieval. Yet, we need management strategies to address these issues to optimize system cache performance. Understanding and addressing these challenges are essential for maintaining optimal responsiveness.

Increased Cache Conflicts:

Cache conflicts emerge when multiple memory locations vie for the same cache set, creating contention for limited space. This shared resource leads to conflicts, forcing the cache management system to decide on premature data evictions. Unnecessary evictions result in the removal of valuable information, increasing cache misses. These conflicts introduce inefficiencies, impacting system performance by requiring constant retrieval from slower memory levels, causing higher latency and energy consumption.

Limited Cache Size:

Cache memory has a finite size, and when dealing with large datasets or frequently accessed data, the cache may not be able to accommodate all relevant information. This limitation can lead to a higher likelihood of cache misses, as some data may be constantly evicted to make room for new information.

Cold Start Problem:

This problem occurs during system startup or after a cache flush, resulting in a heightened rate of cache misses due to the initial emptiness of the cache. This can lead to increased latency and slower responsiveness during the initialization phase. The cache requires time to "warm up" by gradually filling with frequently accessed data to become effective. Efficient cache management strategies are crucial to mitigate these initial performance challenges and optimize overall system responsiveness by addressing the impact of delayed access to essential information.

Cache Inconsistency:

In a multi-core or multi-processor system, maintaining cache coherence across different caches can be challenging, leading to inconsistencies in the stored data. Inconsistent data between caches can cause errors and unexpected behavior in programs relying on coherent data, impacting system stability and reliability.

Energy Challenges in Cache Optimization:

Caches, while crucial for enhancing data access speed, present an energy consumption challenge. The frequent access or modification of cache content contributes to elevated power usage. In devices reliant on batteries or designed for energy efficiency, heightened cache activity becomes a critical concern, potentially diminishing overall battery life and escalating operational costs. Balancing the benefits of cache optimization with the imperative to manage energy consumption is paramount for sustaining the efficiency and cost-effectiveness of battery-powered devices and energy-efficient systems.

METHODOLOGY

Comprehensive study is essential to eliminate the speed gap between cache memory and main memory. Consequently, an architecture-based solution to this issue has been proposed in this section.

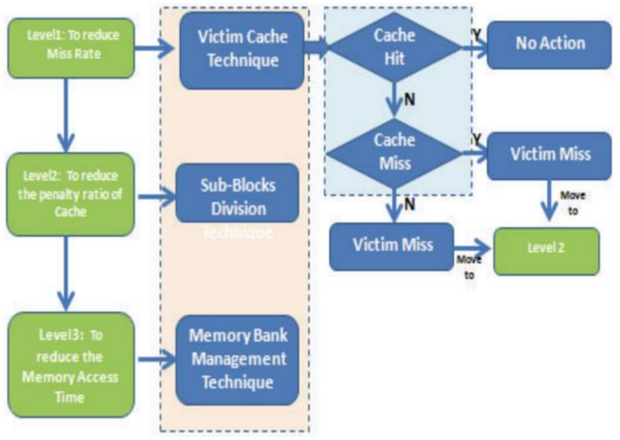
Level1: To reduce Miss Rate
Level2: To reduce the penalty ratio of Cache
Level3: To reduce the Memory Access Time
Victim Cache Technique
Sub-Blocks Division Technique
Memory Bank Management Technique
Cache Hit
Y
No Action
N
Cache Miss
Y
Victim Miss
Move to
N
Victim Miss
Move to
Level 2

FIGURE III

PROPOSED ARCHITECTURE FOR IMPROVING CACHE PERFORMANCE

To overcome the shortcomings of their predecessors, a range of techniques have been methodically removed and integrated into this architectural framework one by one. This incremental architectural approach aims to accelerate cache memory through a complete set of approaches. When considered separately, every approach has an impact on improving memory function.

The flow diagram has been used to demonstrate the architecture, and its implementation.

Step 1: Reducing The Miss Rate using Victim Cache

A victim cache, also known as hardware cache memory, is a tool used to lower the conflict miss rate and increase the hit latency rate for Direct-mapped cache memories. This cache can be placed along the cache refill path that enters at Level 1 so that any cache lines that are discarded from the cache memory are re-cached in the victim cache. As a result, when any type of data is removed from the Level 1 cache, the victim cache is automatically filled. The victim cache will only be considered if Level 1 is missed. The contents of the matching victim cache line and the Level 1 cache-line will be swapped if the access results are coming out as a hit.

Implementation of Victim Cache:

The victim cache behavior in corresponding interaction with the associated level if cache is described below.
Cache Hit: No action
Cache Miss means Victim Hit: In this case, the blocks inside the victim cache and the cache will be swapped. Therefore, the new block which has been stored in victim cache will be considered as the block that has been used most recently.
Cache Miss means Victim Miss: The part from next level will be fetched to cache and the part coming out of cache will get stored in victim cache. This can be explained through an example: Assume a cache of direct-mapped category containing two blocks A, B referring towards the same set of values. Then it is linked to a second entry fully associative victim cache having C, D blocks. The path to be considered is as shown in the FIGURE IV, i.e. A, B, A, B…. and so, on.

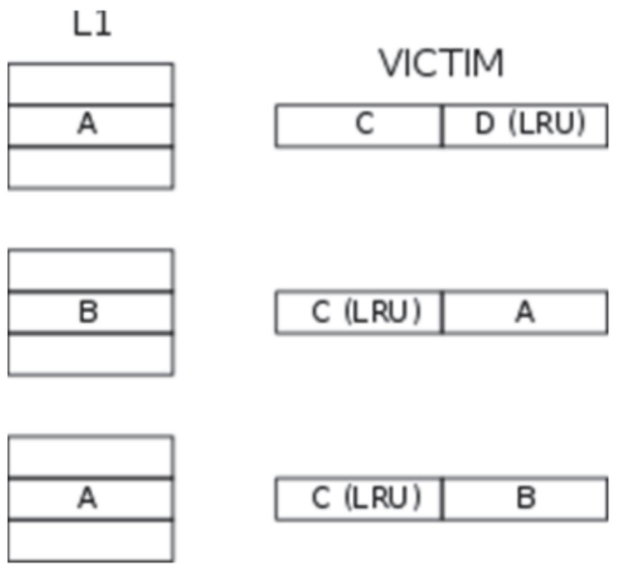

FIGURE IV
IMPLEMENTATION EXAMPLE

As shown in the FIGURE IV, it can be observed that when a victim cache hit occurs, part of blocks A and B will be swapped and the LRU block of victim cache does not change. Thus, we got an illusion of associatively towards the direct-mapped L1 cache which in contrast decreasing the number of conflict misses.

If there are 2 cache memories, Level1 and Level2 with a particular policy (L2 and L1 do not store similar memory locations twice), L2 behaves like victim cache for L1.

Step 2: Reducing the Penalty Ratio of Cache Mis

The performance could be affected by tags at which much space is required, and cache speed is decreased. To decrease the required space of optimizing tags (in addition to making them shorter) on the chip, large blocks are used. Due to reducing the necessary misses, the miss rate might be decreased too. However, because the full block should be transferred between cache and other memories, the miss penalty will be high. To overcome this problem, every block must be divided into sub-blocks as shown in FIGURE V, each one of them has a valid bit. Although the tag is valid for the whole block, just a single block must be read on a miss. So that a smaller miss penalty will be achieved since a block cannot be identified as the minimum unit fetched between cache and memory anymore.

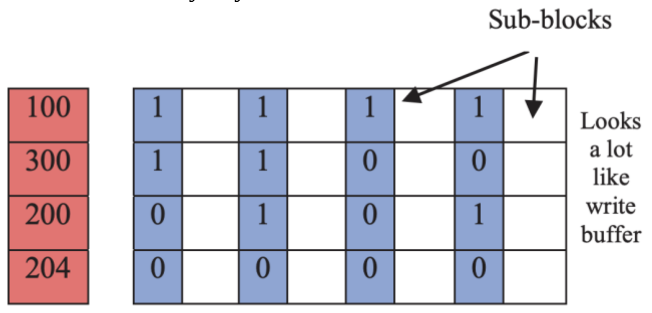

FIGURE V
DIVIDING OF A BLOCK INTO SUB-BLOCKS

Step 3: Decreasing Memory Access Time

Memory bank is a hardware-dependent logical unit of storage in electronics. In a computer, the memory controller with physical organization of the slots of the hardware memory, define the memory bank. A memory bank is a partition of cache memory that is addressed sequentially in the entire set of memory banks. For example, assuming that a(n) is data item which is stored in bank (b), the following data item a(n+1) will be stored in bank (b+1). To avoid the bank cycle time impacts, cache memory will be separated into several banks. Therefore, if data is saved or recovered sequentially each bank will have sufficient time to recover before processing the next request of the same bank. The required number of memory modules to have a similar number of data bits as the bus. A bank can contain many numbers of memory modules as illustrated in FIGURE VI.

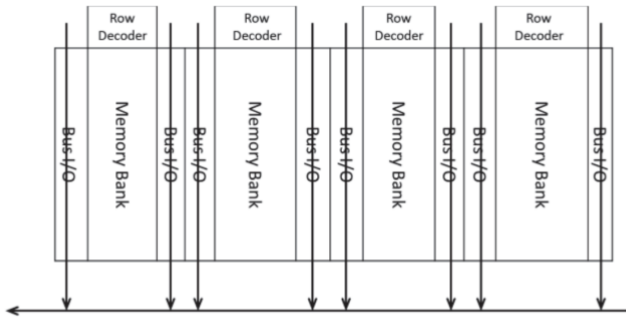

FIGURE VI
MEMORY BANK VOLUMES

RESULTS

The results and outcomes of the analysis on cache memory performance issues reveal significant improvements in overall system efficiency. The implementation of three enhancement techniques—victim cache, sub-blocks, and memory bank—has demonstrated tangible benefits at each level. The evaluation metrics, including miss penalty ratio, cache access speed, and miss rate ratio, provide insights into the effectiveness of the proposed approach. Here are the key findings:

Victim Cache Implementation (Level 1):

Marked reduction in the rate of cache misses. Improved efficiency in handling recently evicted cache lines. Mitigation of latency associated with fetching data from the main memory.

Sub-blocks Division Implementation (Level 2):

Further reduction in the penalty ratio of miss rates. Enhanced cache utilization by optimizing storage for smaller chunks of data. Increased efficiency in handling data granularity mismatches.

Memory Bank Technique Implementation (Level 3):

Substantial decrease in memory access time. Improved parallelism and efficiency in handling multiple memory requests. Optimization of memory banks for concurrent access, contributing to overall system performance.

Cumulative Impact:

Sequential implementation of the three techniques results in a cumulative improvement in cache speed and performance. The proposed approach successfully narrows the speed gap between the processor and main memory, aligning with the evolving demands of faster processors.

Energy Considerations:

Acknowledgment of the energy consumption trade-off associated with on-chip cache placement. Despite increased energy consumption, the benefits in terms of speed and efficiency outweigh the drawbacks.

Comprehensive Evaluation:

The comprehensive evaluation of cache memory performance post-implementation underscores the significance of addressing multiple aspects in enhancing system efficiency.

In conclusion, the results indicate that the proposed architecture, integrating victim cache, sub-blocks, and memory bank techniques, offers a robust solution for optimizing cache memory performance. The findings contribute valuable insights to the field of computer architecture, providing a roadmap for improving cache systems in response to the accelerating speed of processors and the increasing demands on modern computing systems.

CONCLUSION

Several strategies are put forth in this paper to enhance cache memory performance. The benefits and drawbacks of each of the suggested approaches are thoroughly examined. Moreover, their behavior was compared using a number of measures. Thus, the primary conclusions we can get from this research are that increasing block size can reduce the conflict miss rate; however, doing so necessitates increasing cache capacity. Using larger blocks can result in a higher penalty ratio for misses, a shorter hit time, and less power dissipation. Greater cache size leads to slower access times and higher costs. More associative learning, however, yields faster access times and, as a result, shorter cycle times or fewer cycles. Compared to look aside miss cache, victim cache always lowers the rate of misses, albeit at a larger cost.

Future work will involve putting the suggested model into practice. Additionally, many techniques that anticipate data and instruction access in the future will be taken into consideration to recommend even more cache performance improvement.

REFERENCES

[1] V. Chaplot, “Virtual Memory Benefits and Uses,” International Journal of Advance Research in Computer Science and Management Studies, vol. 4, no. 9, 2016.
[2] C. Zhang, F. Vahid, and R. Lysecky, “A Self-Tuning Cache Architecture for Embedded Systems,” ACM Transactions on Embedded Computing Systems, vol. 3, no. 2, pp. 407-425, 2004.
[3] A. Agrawal and S. D. Pudar, “Column-associative Caches: a Technique for Reducing the Miss Rate of Direct-mapped Caches,” in Proc. of the 20th Annual International Symposium on Computer Architecture, 1993, pp. 179-190.
[4] S. Kumar and P. K. Singh, “An overview of modern cache memory and performance analysis of replacement policies,” in Proc. of the IEEE International Conference on Engineering and Technology (ICETECH), 2016, Coimbatore, India, pp. 210-214.
[5] S. S. Omran and I. A. Amory, “Implementation of LRU Replacement Policy for Reconfigurable Cache Memory Using FPGA,” in Proc. of the International Conference on Advanced Science and Engineering, 2018, pp. 13-18.
[6] K. Westerholz, S. Honal, J. Plankl, and C. Hafer, “Improving performance by cache driven memory management,” in Proc. of the 1st IEEE Symposium on High Performance Computer Architecture, 1995, Raleigh, USA, pp. 234-242.
[7] Y. A. Divya, “An Efficient Virtual Memory using Graceful Code,” International Journal of Trend in Scientific Research and Development vol. 3 no. 4, pp. 623-626, 2019.

[8] M. Biglari, K. Barijough, M. Goudarzi, and B. Pourmohseni, “A FineGrained Configurable Cache Architecture for Soft Processors,” in Proc. of the 18th CSI International Symposium on Computer Architecture and Digital Systems (CADS), 2015.

[9] A. Q. Ahmad, M. Masoud, and S. S. Ismail, “Average Memory Access Time ReductionVia Adding Victim Cache,” International Journal of Applied Engineering Research, vol. 11, no. 19, pp. 9767-9771, 2016.

[10] Y. Kim and Y. H. Song, “Impact of processor cache memory on storage performance,” in Proc., of the International SoC Design Conference (ISOCC), 2017, Seoul, pp. 304-305.

[11] W. P. Dias and E. Colonese, “Performance Analysis of Cache and Scratchpad Memory in an Embedded High Performance Processor,” in Proc. of the 5th International Conference on Information Technology: New Generations, 2008, Las Vegas, USA, pp. 657-661.

[12] K. Singh and S. Khanna, “Split Memory Based Memory Architecture with Single-ended High Speed Sensing Circuit to Improve Cache Memory Performance,” in Proc. of the 6th International Conference on Signal Processing and Communication (ICSC), 2020, Noida, India, pp. 188-193.

[13] R. Kolanski, “A logic for virtual memory, “Electronic Notes in Theoretical Computer Science, vol. 217, pp. 61-77, 2008.

[14] S. P. Smith and J. Kuban, “Modelling and Enhancing Virtual Memory Performance in Logic Simulation,” in Proc. of the IEEE International Conference on Computer-Aided Design, January 1988, pp. 264-265.

[15] M. T. Banday and M. Khan, “A study of recent advances in cache memories,” in Proc. of the International Conference on Contemporary Computing and Informatics (IC3I), 2014, Mysore, India, pp. 398-403.