Implementing Multilevel Cache with LRU Replacement Policy for Single Cache Enhancement

Department of Electrical and Computer Engineering, Birzeit University

Birzeit, Palestine

1190768@student.birzeit.edu, 1191519@student.birzeit.edu, 1191636@student.birzeit.edu

***Abstract*—Cache is essential to computer architecture because it helps computers run faster, reduce latency, handle a variety of workloads, preserve data coherence, optimize memory hierarchy, use less power, and be scalable for future developments.**

This paper describes the behavior of cache with different placement algorithms such as direct cache mapping, set-associative mapping and fully associative mapping. In addition to exploring how these algorithms interact with Least Recently Used (LRU) replacement police, with obtaining hit and miss ratios. Moving toward the concept of multilevel caches, to enhance the system performance as multi level caches can do more than a single cache, it can minimize latency, increase hit rates, adjust to different workloads, maximize resource usage, preserve data consistency, facilitate scalability, and improve overall system responsiveness

Keywords—cache, LRU, multi level cache, direct mapping, set-associative, fully associative.

I. INTRODUCTION

In the dynamic world of computing, efficiency and speed are paramount. At the heart of this quest for performance lies the concept of cache, a smaller but faster type of memory that stores copies of frequently accessed data for quick retrieval. This term paper delves into the intricacies of cache management, focusing on placement algorithms which determine where data should be stored in the cache. We also explore replacement policies, the rules that decide which data to discard when the cache is full. Additionally, the paper sheds light on the concept of multi-level cache, a hierarchical approach to caching that further optimizes data access speed. Understanding these components is crucial for enhancing the performance of modern computing systems [1].

II. RELATED WORKS

The author begins by stating the development of a basic analytical model for predicting the performance of on-chip cache hierarchies in chip-multiprocessors (CMP). Validated through extensive full-system simulation scenarios, this model proves effective in assessing the responsiveness of memory hierarchies within complex architectures. The ultimate goal is determining the most effective silicon allocation for each level of the memory hierarchy in future systems. The paper highlights the utility of a predictive model for cache hierarchy behavior in reducing computational burdens of simulations. This established model guides silicon allocation decisions and provides insights into the performance and responsiveness of the CMP cache hierarchy. While it facilitates preliminary evaluations based on real application behavior, it does not aim to replace comprehensive simulations. The paper concludes with recommendations for further research and improvements to enhance the model's accuracy and adaptability across various architectures and applications [2].

The authors present a new metric, "Improvement over LRU Per Kilo Byte" (IPKB), for evaluating cache replacement strategies more comprehensively by accounting for both miss rate improvement and hardware overhead. This addresses the limitations of traditional metrics that neglect the impact of hardware overhead on power consumption. The study includes an evaluation of four different policies using IPKB and analyzes the hardware overhead for various replacement techniques. The paper emphasizes the need to consider both hardware overhead and miss rate improvement in evaluating cache replacement strategies and suggests incorporating energy usage in future evaluations. The introduction of IPKB aims to provide a more accurate assessment of power consumption in cache replacement strategies, highlighting the inadequacies of existing metrics [3].

The authors introduce the Expected Hit Count (EHC), a novel cache replacement technique designed to address inefficiencies in CPU last-level caches. This technique is based on the correlation between a cache block's expected hits and the reciprocal of its reuse distance, aiming to enhance overall performance and cache efficiency. In their evaluation, the EHC policy is compared with existing policies through cycle-accurate full-system simulation. The results show an average speedup of 3.5% on a single-core processor and 12.2% on a multi-core processor when comparing EHC with the state-of-the-art replacement policy, signifying a significant improvement in cache efficiency and speed. The EHC technique is presented as an effective and economical solution for cache replacement challenges [4].

The authors discuss the implementation of the Least Recently Used (LRU) replacement policy and cache mapping functions using FPGA and VHDL in reconfigurable cache memory. Through the study, executed on Xilinx Virtex 7, they compare execution times and hit ratios across various cache associativity scenarios. The paper focuses on the effective demonstration of the LRU policy's implementation and the evaluation of different cache mapping techniques. Results indicate that associative cache memory with an 8-way set performs better in terms of hit rate and execution time. The authors conclude that while the traditional LRU method is simpler for single-unit maintenance, the tree-based pseudo LRU approach proves to be more efficient, requiring a smaller LRU array [5].

The authors examine the optimization of pipelined main cache architecture in high-speed CPUs, focusing on enhancing system performance. They employ a multilayer optimization approach to identify the ideal cache size and pipeline depth, considering both architectural and physical aspects. The study uses timing analysis and trace-driven simulation to assess how cache size and pipeline depth influence CPU cycle time. The results underscore the crucial role of pipelined caches in improving performance, particularly in reducing CPU cycle time and time per instruction for high clock rate CPUs. The paper emphasizes the need for optimizing pipelined main caches to achieve shorter CPU cycle times and larger caches in high-speed computing environments [6].

The authors introduce SF-LRU, a cache replacement strategy that merges Least Recently Used (LRU) and Least Frequently Used (LFU) principles with a second chance concept, aiming to reduce cache misses and battery consumption. The study finds that SF-LRU surpasses traditional LRU and LFU algorithms in miss ratio and power efficiency. This approach effectively tackles cache memory issues and, when combined with advanced cache enhancement methods, improves speed while significantly reducing cache miss ratio and power usage [7].

The authors explore the effects of CPU cycle time, set associativity, block size, and memory speed on cache design and system performance. They emphasize the trade-offs in cache design and the benefits of multi-level cache structures for enhanced performance. Using theoretical and empirical analysis, the paper highlights the importance of balancing various cache design factors, including block and cache size, CPU cycle time, and cache miss penalty, to optimize memory performance with multi-level cache hierarchies [8].

The authors examine hardware techniques like victim caching, stream buffers, and miss caching to enhance cache performance in computer systems. The paper discusses the effectiveness of these strategies in reducing cache misses and improving overall system performance. It also highlights the role of memory hierarchy in system performance and suggests future research in areas like non-unit stride access patterns and second-level caches in numerical applications. The impact of these hardware strategies on lowering cache miss rates and boosting system performance is a key focus of the study [9].

The authors investigate way-prediction and selective direct-mapping techniques to reduce energy use in set-associative caches. The study aims to balance energy savings with maintaining access times, crucial for high-performance cache systems. Comparing these methods with other cache improvement strategies, the paper highlights their effectiveness in lowering processor energy consumption and enhancing cache speed. The results show that these techniques notably improve efficiency and reduce energy consumption, especially in L1 data and instruction caches, as well as in overall processor energy [10].

The authors present the Indirect Index Cache (IIC), a new software-managed secondary cache architecture intended to outperform traditional caches. The IIC uses a generational replacement mechanism and is designed as a direct replacement for a standard 1 MB secondary cache, with the goal of improving cache management and performance. The study uses system traces to demonstrate the advantages of the IIC, such as reduced data locking penalties and flexible data partitioning. The results suggest that fully associative, software-managed secondary caches, like the IIC, offer better performance than conventional caches in on-chip secondary cache design [11].

III. BACKGROUND

A. *Cache Memory*

A computer's cache memory can be described as a small very fast storage space inside the CPU. It stores duplicates of the main memory's frequently accessed data. Consider it the processor's quick-access library, holding important data it needs to work with often. Cache memory speeds up the computer's overall performance by decreasing the time it takes to get data from the bigger, slower main memory by storing this data nearby. Cache memory operates as a high-speed intermediary between the CPU and RAM, facilitating quick retrieval of commonly accessed data and instructions. It essentially functions as a connector, minimizing delays by instantly delivering crucial information to the CPU. Despite being pricier than main or disk memory, cache memory represents a more economical alternative compared to CPU registers. Its primary goal is to optimize and harmonize with a fast CPU, thereby enhancing the overall efficiency of the system. Cache performance is evaluated based on how effectively it retrieves data requested by the processor. When the processor needs data from the main memory, it first looks in the cache. If the data is already stored in the cache (a Cache Hit), it's quickly accessed. However, if the data isn't in the cache (a Cache Miss), the cache fetches it from the main memory and stores a copy for future use. Subsequent requests for the same data benefit from this stored copy. The efficiency of cache memory is commonly measured using its Hit ratio, which indicates the percentage of successful cache hits among all memory accesses.[12].

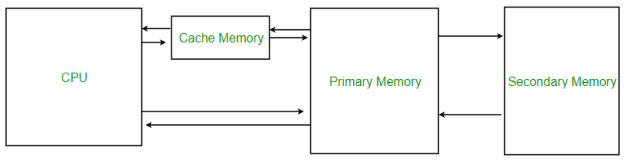


Figure 1: Cache memory

B. *Placement Algorithms*

Placement algorithms in computer architecture refer to methods that are effectively employed in cache management to allocate and organize data in the memory hierarchy. Both cache misses and data access are optimized by implementing these techniques. Cache placement algorithms that use different approaches to deciding where data should be kept, include fully associative, set-associative, and direct-mapped methods.

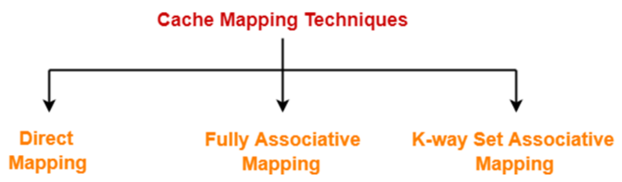


Figure 2: Placement algorithms

1. Direct Mapping Algorithm

Mapping addresses from main memory to designated places (lines or slots) in the cache is made easy and clear with the Direct Mapped cache placement mechanism. All of the main memory blocks are mapped to exactly one place in the cache using this approach. Block, main memory, and cache sizes all affect how many bits are allocated to each component. The tag that offers a unique identification for the memory block and a representation of the high-order bits of the address. The index bits identify the cache line to which the memory block is mapped, while the offset bits identify the data's location inside the line. The four cache lines are represented by the addresses in the main memory as shown in the figure below. This will be done by applying the modulus operation.

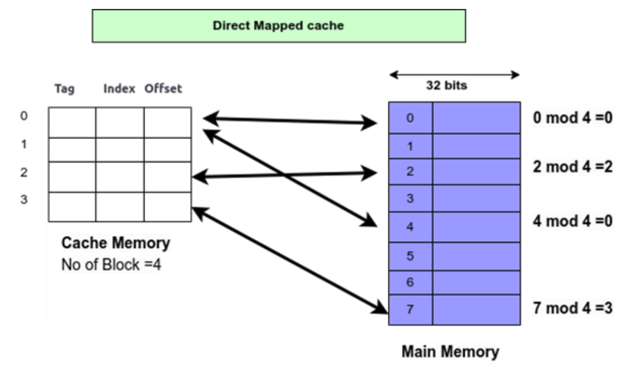


Figure 3: Direct mapping cache

To find a block, the tag generated from the address will be compared with the tag kept in a cache line of the chosen set. A cache hit is produced and the requested data is obtained from the cache if the tags match. A cache miss occurs when the tags are missing. In this instance, the cache does not contain the requested memory block. Data is taken from main memory by the cache controller and placed in the chosen set's cache line. The new block will be used for the old block, which is removed from that cache line. The data is sent back to the requesting processor or core after receiving it from the cache (cache hit) or main memory (cache miss). Because the index bits 000 refer to the first index, the cache controller obtains the tag linked to the cache line there. Next, it contrasts it with the tag—01 in this case—that was obtained from the memory location and is stored in the cache. There is a cache miss since the tags don't match, indicating that the requested memory block isn't in the cache. In order to store the requested data in the cache line that fits the first index, the cache controller first receives it from the main memory address, 01000011. In the case that the cache line already contains data, the new block will be placed in its current location. The offset bits (011) indicate that the controller must

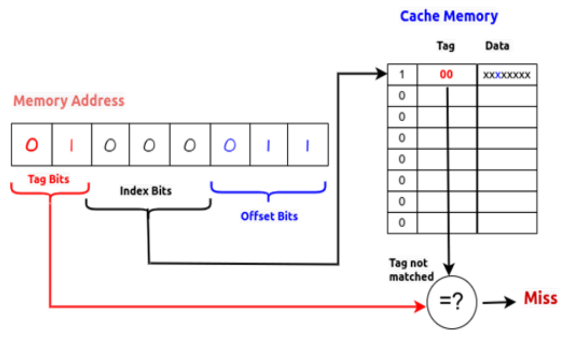


Figure 5: Direct mapping address bits

get the third word from the first cache line if the tags match [13].

1. Set-Associative Mapping

The weaknesses of direct mapping have been eliminated in order to create an improved version of mapping. The set associative approach solves the issue of possible thrashing in the direct mapping technique. To do this, put a few lines together creating a set, rather than having exactly one line that a block can map to in the cache." A memory block can then be mapped to any line within a given set. Due to set-associative mapping, a word in the cache might have two or more alternatives for the same index address in the main memory. The greatest features of both associative and direct cache mapping approaches are combined in set associative cache mapping. The set offset bits provide the index bits in a set associative mapping. In this case, the cache consists of many sets, each of which has a number of lines in it [14].

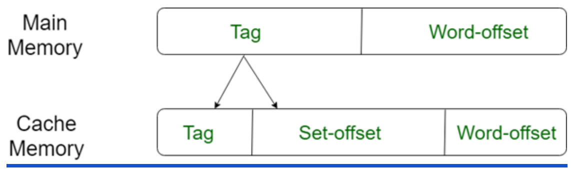
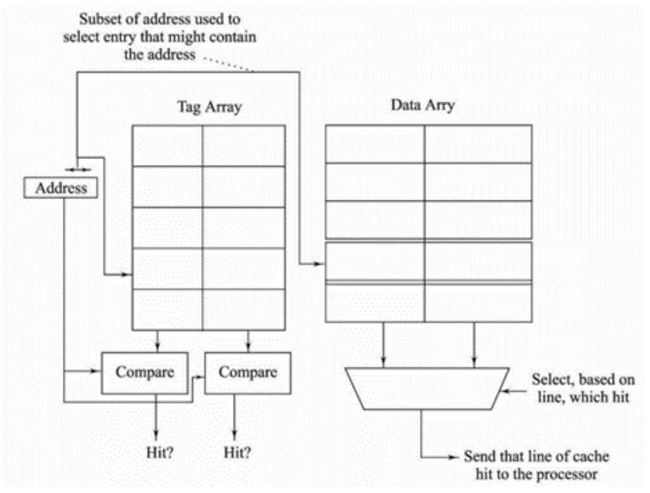


Figure 4: Set associative mapping

Two Way Set Associative Cache Block

A subset of the address bits is used to determine the set that maybe contain the address, the same as in a direct mapped cache.In order to determine if a hit has happened, two tags need to be compared with the memory reference location. A hit has been made if any of the tags matches the address; a suitable line in the data array is then chosen [16]. As shown in the figure below:



Subset of address used to select entry that might contain the address
Tag Array
Data Arry
Address
Compare
Compare
Select, based on line, which hit
Hit?
Hit?
Send that line of cache hit to the processor

Figure 6: 2-Way set associative cache block

Higher than two Way Set Associative Cache Block

- To find if a hit has happened, more comparators will be used.

- In order to choose the set that an address will be stored in, a set-associative cache will use less bits of the address than a direct-mapped cache with the same number of lines.
- To determine how many sets there are in a cache, the number of lines in the cache will be divided by associativity [16].

Four-Way Set Associative

- To find if a hit has happened, the address of the memory reference must be compared to four tags.
- A hit has been made and the matching line in the data array is chosen if one of the tags matches the address [16]. As shown in the figure below:

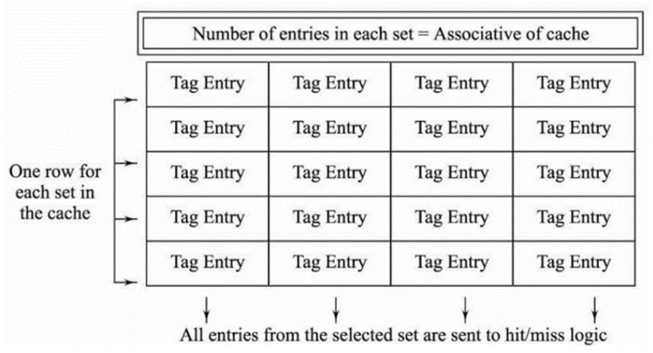

Number of entries in each set = Associative of cache
Tag Entry
One row for each set in the cache
All entries from the selected set are sent to hit/miss logic

Figure 7: 4-Way set associative mapping

2. **Fully Associative Mapping**

A cache mapping technique known as fully associative mapping enables mapping of the main memory block to an open cache line. Data might be stored in any cache block with a completely associative cache. Every memory address would not have to be put into a single specific block. If the information and data are obtained from memory, they might be stored in a cache block that isn't being used. The problem of conflict misses is resolved with the help of the completely associative mapping. This implies that a line of cache memory may easily include any block from the main memory. For example, B0 is easily able to appear in L1, L2, L3, and L4. For every other block, the situation would be the same as well. The probability of a cache hit rises significantly in this way as shown in the figure below:

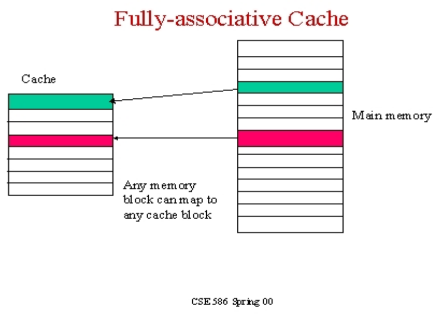


Figure 8: Fully associative mapping

The Cache and the Main Memory are separated into lines and blocks in fully associative mapping. In this algorithm, the comparator's value is set to match the total number of lines in the cache. This is because in a fully associative setup, any large memory block can be placed in any cache line. As a result, the tag of the given address is compared against the tags of all the lines in the cache to determine the appropriate location for the data. The main benefit of a fully associative mapping is that it increases the hit rate and resolves the conflict miss issue. However, the comparison time is extended which is the main weakness. One must compare the tag bit of the looking for block with each and every line tag in order to find any particular block in any cache. A cache hit happens if they match, while a cache miss happens if they don't [17].

*C. **Replacement Policy: Least Recently Used (LRU)***

The idea of the LRU cache policy is that data items with little recent usage are probably not going to be used much in the future. So, the LRU algorithm removes the item that has been used the least when the cache is full and a new item has to be added. To implement this, the cache stores the order in which objects are received. There are several data structures that may be used for this, including arrays and linked lists.

How LRU Works

Access Tracking: A data item in the cache is pushed to the front of the list each time it is requested, signifying that it has been used lately.

Replacement: The item at the bottom of the list (which hasn't been used in the longest) is eliminated to create space when new data has to be added to a full cache.

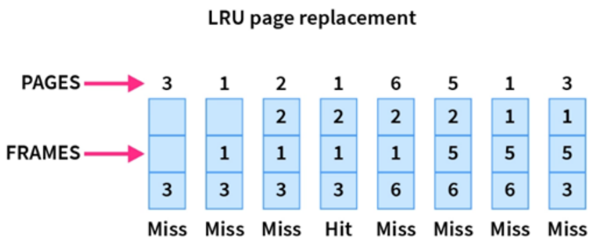


Figure 9: LRU replacement

The Advantages of LRU

Simplicity: LRU is simple to understand and apply. **Efficiency**: By using its past usage data, it successfully predicts which things are likely to be used.

The Disadvantages of LRU

Overhead: Especially in systems with a big cache, keeping track of each item's order of access can be resource-intensive.

Suboptimal in Certain Cases: Not every application type will benefit from LRU, especially if access patterns vary from the common assumption that "recently used items are more likely to be reused [18].

D. Multi-level Cache

In computer architecture, multilevel cache refers to the usage of many cache levels (L1, L2, at times L3) in order to improve system speed. The L1 cache has the lowest latency and the smallest capacity; it is the closest to the CPU. While L3 cache, which is situated further from the CPU and has more capacity, may be shared by several CPU cores, L2 cache is bigger and can be shared by multiple cores. The overall speed and efficiency of the system are increased by this hierarchical architecture, which makes use of the locality principle to optimize the balance between low latency and greater storage capacity.

The use of Multi-Level Caches is a technique designed to improve Cache Performance through a decrease of "MISS PENALTY." When a cache "miss" happens, the term refers to the extra time needed to get data from main memory to the cache.

The purpose of multi-level caches is to fill the gap between slower main memory and higher-capacity but faster caches. Through the setup of a layered data storage model, they optimize memory hierarchies, improving system performance and data access speed.

- The Advantages of Multi-Level Cache

Improved Performance: Through a decrease of memory access latency, Multi-Level Caches greatly improve system performance overall. Because caches are close to the CPU, frequently used data may be accessed more quickly, speeding up program execution.

Optimized Hierarchy: Using the concepts of temporal and spatial locality, multi-level caches allow various access patterns. By ensuring that data is effectively maintained throughout cache levels, this hierarchical strategy maximizes the reuse of data.

Effective Memory Hierarchy: Multi-Level Caches help create a more efficient memory hierarchy by balancing cache size and access speed. They provide processors the ability to quickly access frequently used data without depending completely on slower main memory.

Lower Miss Penalty: The effects of cache "misses" are reduced by using multi-level caches. Lower-level caches may include data that higher-level caches do not, which can minimize the requirement to access main memory as well as the miss penalty.

Versatility: Performance on a variety of devices, including servers, mobile devices, laptops, and desktop computers, can be enhanced by Multi-Level Caches because they can be adjusted to different computing systems and architectures.

- The Disadvantages of Multi-Level Caches

Complexity: The implementation and management of multi-level caches adds complexity to the architecture of the memory hierarchy. This complexity may make cache management more difficult and demand the use of clarify cache replacement strategies.

Cost: An increase in chip area needs caused by multi-level caches may result in higher production costs. Increased hardware resources are required for many cache levels, which impacts manufacturing costs.

Access Time Trade-off: Higher-level caches have faster access times, but because of their bigger sizes, lower-level caches may have a little longer access times. It is important to balance the trade-off between capacity and access time.

Limited Improvement: For activities requiring huge working sets that exceed cache capacity, or for applications with unpredictable memory access patterns, Multi-Level Caches may not provide noticeable increases, despite their advantages [19].

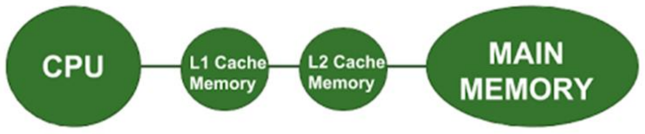


Figure 10: Multi level cache

IV. METHODOLOGY

In our study, we used System Verilog to simulate and evaluate different cache configurations. Our approach began with constructing a main memory model, followed by the development of a single-level cache system. This system was designed to support various cache placement policies, including Direct Mapping, Set Associativity, and Fully Associative caching. We also implemented a Least Recently Used (LRU) algorithm to efficiently handle scenarios where the cache was full, ensuring optimal data management during cache misses.

For each cache configuration, we carefully calculated performance metrics such as hit rates, miss rates, write hit ratios, and write miss ratios. The first phase of our testing focused on the single-level cache, where we analyzed the performance of each placement policy under controlled conditions. This phase allowed us to gather comprehensive data on the effectiveness of each cache type in isolation.

Following the single-level cache analysis, we expanded our study to include multi-level cache systems. These systems were designed with similar functional capabilities to the single-level caches but varied in their configuration parameters. A key aspect of the multi-level cache system was the synchronization mechanism between the two levels of caches, ensuring accurate data retrieval and storage.

Our testing methodology for the multi-level caches involved using random numbers to mimic a range of access patterns. This approach was chosen to test the cache system's responsiveness and efficiency under diverse and unpredictable usage scenarios. We conducted tests across different placement cases for a thorough evaluation of each cache configuration's performance.

Through this systematic and detailed approach using System Verilog, we were able to gain valuable insights into the performance characteristics of single-level and multi-level cache architectures, particularly under varying and random data access conditions.

V. PROPOSED SOLUTION

The design and implementation of a SystemVerilog multilevel cache module with a Least Recently Used (LRU) replacement policy and cache placement algorithms that include Direct Mapped, Set Associative, and Fully Associative is the proposed solution. By effectively supporting a variety of applications with its flexible cache algorithms, this multilevel cache system aims to improve overall system performance. To ensure that useless data is eliminated to increase efficiency and hit rates, cache lines are carefully managed using the LRU replacement strategy.

In order to avoid inconsistencies and ensure that the most current data is used, the cache module is built to handle read and write operations. An efficient cache management strategy reduces memory latency, improves system responsiveness, and speeds up data retrieval from the cache. Interactions between the cache module and main memory are reliable and consistent when cache operations are synchronized with a clock signal.
To maximize memory hierarchy and utilize system resources effectively, a multilevel cache design is suggested. By maximizing the advantages of the memory hierarchy, this strategy hopes to enhance system performance overall. It is suggested that the well thought-out cache module is adaptable and scalable, providing future-proofing for changing technologies and system architectures.
Our system is configurable with user inputted values, means that the user can change any of the values in the configuration file, such as memory size and others.
In conclusion, this cache module's full implementation handles a number of system performance, adaptability, and efficacy issues, making it an important component of building strong and dependable computer systems. The goal of the proposed solution is to offer a flexible and all-around caching system that can handle changing workloads and technological advancements.

VI. EXPERIMENTAL SETUP

In this research, we use the EDAPlaygrounds to test the design codes, testbench, and obtain the waveforms for the results. Cache system and multi-level cache are all test with random values in test benches for all placement algorithms (Direct mapping, Set Associative mapping and Fully Associative mapping)

VII. EXPERIMENTAL RESULTS

In all tables, TH is the Total Hits, TM is the Total misses, TA is the Total Accesses, HR is the Hit Ratio, MR is the Miss Ratio, WHR is the Write Hit Ratio, WMR is the Write Miss Ratio, RHT is the Read Hit Ratio, RMR is the Read Miss Ratio and L is the Level number, all of them are in percentage (%).

A. Single level cache

- Direct mapping:

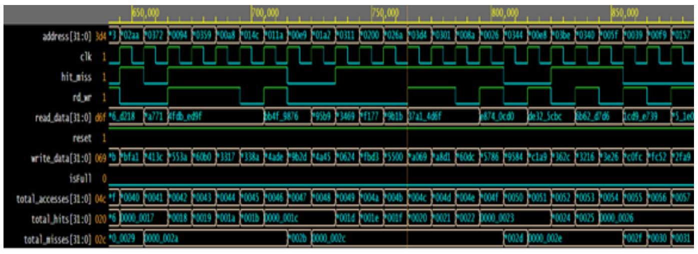


Figure 11: Single level-Direct mapping waveform

Table 1: Single level-Direct mapping results

TH	TM	TA	HR	MR	WHR	WMR	RHR	RMS
140	59	199	70.35	29.65	71.30	28.70	69.32	30.77

- Set Associative mapping:
  2-way set associative

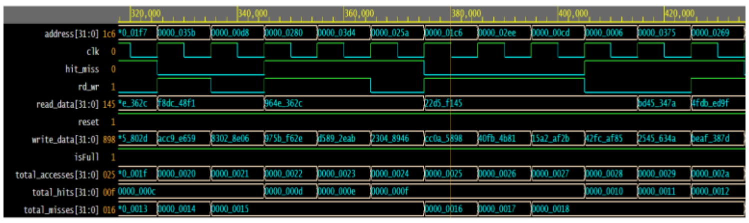


Figure 12: Single level-Set associative mapping waveform

Table 2: Single level-Set associative mapping results

TH	TM	TA	HR	MR	WHR	WMR	RHR	RMS
169	30	199	84.92	15.08	80.56	19.44	90.11	9.89

:

- Fully Associative mapping:

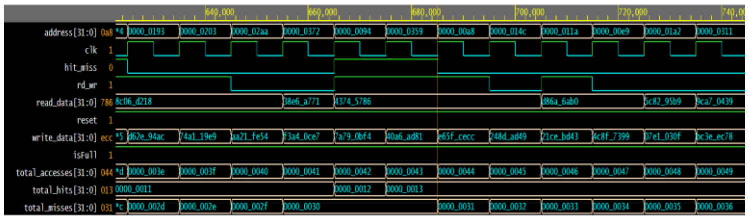


Figure 13: Single level-Fully associative mapping waveform

Table 3: Single level-Fully associative mapping results

TH	TM	TA	HR	MR	WHR	WMR	RHR	RMS
93	106	199	46.73	53.27	47.22	52.78	46.15	53.85

B. Multi-level cache

- Direct mapping:

Table 4: Multi level-Direct mapping results

L	TH	TM	TA	HR	MR	WHR	WMR	RHR	RMS
1	139	59	198	70.2	29.8	71.3	28.7	68.89	31.11
2	0	59	59	0	100	0	100	0	100

- Set Associative mapping:

2-way set associative

Table 5: Multi level-Set associative mapping results

L	TH	TM	TA	HR	MR	WHR	WMR	RHR	RMS
1	167	31	198	84.3 4	15.66	80.56	19.44	88.89	11.11
2	1	30	31	3.23	96.77	0	100	10	90

- Fully Associative mapping:

Table 6: Single level-Fully associative mapping results

L	TH	TM	TA	HR	MR	WHR	WMR	RHR	RMS
1	92	106	198	46.46	53.54	34.23	38.26	45.56	54.44
2	44	62	106	41.51	58.49	40.35	59.65	42.86	57.14

VIII. RESULTS ANALYSIS AND DISCUSSION

The cache performance data from both single-level and multi-level cache configurations reveal critical insights into the efficiency and effectiveness of different cache architectures. This analysis compares the hit rates, miss rates, and read/write ratios across these configurations, providing an understanding of their operational characteristics.

Single-Level Cache Analysis

When we looked at how different single-level caches work, we found some interesting things. The 2-Way Set Associative cache was the best at finding data quickly, with a hit rate of about 85%. This means it's really good at getting the information it needs fast, especially when reading. The Fully Associative cache, which we usually think is pretty good, actually didn't do as well in our tests - it had a hit rate of around 47%. The Direct Mapping cache was pretty good, with a hit rate of 70%. It did a decent job both when reading and writing data. These results show us that the kind of cache you choose can make a big difference. The 2-Way Set Associative cache is a great choice if you need to get data fast, but the best cache for you might depend on what you need it for.

Multi-Level Cache Analysis

In our tests with multi-level caches, we noticed different ways they handle data. For the 2-Way Set Associative cache, the first level was really good at finding data quickly, showing a hit rate of about 84%. Because it did so well, the second level didn't have much to do, with a hit rate of just over 3%. For the Fully Associative cache, both levels were just okay, not too fast or too slow in finding data. The first level had a hit rate of around 46% and the second level was close to 42%. With the Direct Mapping cache, the first level did a good job with a hit rate of 70%, but the second level didn't help at all, with a hit rate of 0%. This tells us that how well the first level of the cache works is really important. If the first level does a great job, like in the 2-Way Set Associative cache, you might not need the second level much. But each type of cache works differently, so the best setup depends on what you need it for.

Compare between single and multilevel

In our study of different types of cache memory, we noticed some clear differences between single-level and multi-level caches. In single-level caches, the 2-Way Set Associative type was really good at quickly finding data, with a high hit rate of 84.92%. This means it was great at getting the data it needed, especially when reading. But when we looked at caches with two levels, the first level's performance was really important. Again, the 2-Way Set Associative cache did well in the first level (hit rate of 84.34%), which meant the second level wasn't used much (hit rate of only 3.23%). On the other hand, the Fully Associative cache didn't do as well in both single-level and multi-level setups, maybe because of how we set up the test or something about how this type of cache works. Our findings show that the kind of cache you use can really change how a system performs. Even though having two levels of cache can be helpful in some situations, a single-level cache, if set up right, can often be good enough and makes things simpler

LIMITATIONS

In this paper, we faced many challenges, including when we used the System C language to build the system. We faced many problems due to our lack of knowledge of the language before, so the choice was to use the System Verilog language. Also, there are not many helpful codes on the Internet for this system.
In addition, the online Background gives us a specific size for the memory. So, we were not able to do a complete test of all the states on the multilevel.

CONCLUSION AND FUTURE WORK

The comprehensive analysis of cache performance, encompassing both single-level and multi-level configurations, provides valuable insights into the operational characteristics of different cache architectures. Single-level cache examinations revealed distinctive behaviors among cache types. Notably, the 2-Way Set Associative cache exhibited remarkable efficiency with an 85% hit rate, making it a standout choice for rapid data retrieval, especially during reading operations. Conversely, the Fully Associative cache demonstrated comparatively lower performance with a 47% hit rate, challenging conventional assumptions. The Direct Mapping cache performed well with a 70% hit rate, demonstrating competence in both reading and writing data. In the realm of multi-level caches, the effectiveness of the first level was emphasized. For instance, the 2-Way Set Associative cache excelled at the first level (84% hit rate), rendering the second level less utilized (3% hit rate). In contrast, the Fully Associative cache exhibited moderate performance in both levels, while the Direct Mapping cache showcased the importance of a proficient first level. Comparisons between single and multi-level caches highlighted the nuanced trade-offs; the 2-Way Set Associative cache, while excelling in single-level configurations, minimized the need for a second level. This study underscores the significant impact of cache selection on system performance, with implications for simplicity and efficiency in various scenarios.
Future investigations could delve deeper into optimizing configurations for multi-level caches, exploring ways to enhance the performance of the second level and understanding its role in specific scenarios. Additionally, considering diverse workloads and varying access patterns in real-world applications could provide a more nuanced understanding of cache behavior. Further research might

explore adaptive cache management strategies that dynamically adjust cache parameters based on the evolving demands of the workload. Additionally, investigating the energy efficiency implications of different cache architectures could contribute to more sustainable computing practices. Finally, exploring emerging technologies and their integration with cache systems, such as non-volatile memory technologies, could present novel opportunities for improving overall system performance and efficiency.

ACKNOWLEDGMENT

We are thankful to Dr. Ayman Hroub, our adviser, for his essential advice and knowledge over the course of the paper.

REFERENCES

[1] cache. (2024, January 16). Merriam-Webster Dictionary. https://www.merriam-webster.com/dictionary/cache

[2] P. Prieto, V. Puente and J. -A. Gregorio, "Multilevel Cache Modeling for Chip-Multiprocessor Systems," in IEEE Computer Architecture Letters, vol. 10, no. 2, pp. 49-52, July-Dec. 2011, doi: 10.1109/L-CA.2011.20.keywords: {Thermal analysis;Cache storage;Complexity theory;Computational modeling;Thermal sensors;Software tools;Multiprocessing systems;Multi-core/single-chip multiprocessors;Memory hierarchy},

[3] A. Haddadi, M. Rezaei and H. Nikmehr, "IPKB: A New Metric for Evaluating Cache Replacement Policies," 2019 27th Iranian Conference on Electrical Engineering (ICEE), Yazd, Iran, 2019, pp. 1940-1945, doi: 10.1109/IranianCEE.2019.8786446. keywords: {Cache storage;Power demand;Memory management;replacement policy;cache;memory systems;memory power consumption;replacement policies' metrics},

[4] A. Vakil-Ghahani, S. Mahdizadeh-Shahri, M. -R. Lotfi-Namin, M. Bakhshalipour, P. Lotfi-Kamran and H. Sarbazi-Azad, "Cache Replacement Policy Based on Expected Hit Count," in IEEE Computer Architecture Letters, vol. 17, no. 1, pp. 64-67, 1 Jan.-June 2018, doi: 10.1109/LCA.2017.2762660.keywords: {Radiation detectors;Proposals;Correlation;History;Multicore processing;Prefetching;Memory system;memory-intensive workload;last-level cache;replacement policy;Belady's MIN},

[5] S. S. Omran and I. A. Amory, "Implementation of LRU Replacement Policy for Reconfigurable Cache Memory Using FPGA," 2018 International Conference on Advanced Science and Engineering (ICOASE), Duhok, Iraq, 2018, pp. 13-18, doi: 10.1109/ICOASE.2018.8548892. keywords: {Cache memory;Arrays;Field programmable gate arrays;Indexes;Hardware;Multiplexing;Distributed databases;Hit ratio;Replacement policies;LRU;FPGA;VHDL},

[6] K. Olukotun, T. N. Mudge and R. B. Brown, "Multilevel optimization of pipelined caches," in IEEE Transactions on Computers, vol. 46, no. 10, pp. 1093-1102, Oct. 1997, doi: 10.1109/12.628394.keywords: {Pipeline processing;Delay effects;Clocks;Timing;Design optimization;Multichip modules;Logic design;Computer architecture;Gallium arsenide;Packaging},

[7] J. Alghazo, A. Akaaboune and N. Botros, "SF-LRU cache replacement algorithm," Records of the 2004 International Workshop on Memory Technology, Design and Testing, 2004., San Jose, CA, USA, 2004, pp. 19-24, doi: 10.1109/MTDT.2004.1327979. keywords: {Cache memory;Energy consumption;Costs;Delay;History;Frequency;Application software;Bridges;System performance;Clocks},

[8] https://dl.acm.org/doi/pdf/10.1145/633625.52433

[9] N. P. Jouppi, "Improving direct-mapped cache performance by the addition of a small fully-associative cache and prefetch buffers," [1990] Proceedings. The 17th Annual International Symposium on Computer Architecture, Seattle, WA, USA, 1990, pp. 364-373, doi: 10.1109/ISCA.1990.134547. keywords: {Costs;Prefetching;Hardware;Reduced instruction set computing;Technology forecasting;Application software;Delay;Performance loss},

[10] M. D. Powell, A. Agarwal, T. N. Vijaykumar, B. Falsafi and K. Roy, "Reducing set-associative cache energy via way-prediction and selective direct-mapping," Proceedings. 34th ACM/IEEE International Symposium on Microarchitecture. MICRO-34, Austin, TX, USA, 2001, pp. 54-65, doi: 10.1109/MICRO.2001.991105. keywords: {Energy dissipation;Probes;Degradation;Power engineering and energy;Computer architecture;Laboratories;Application software;Microprocessors;Costs;Pipelines},

[11] E. G. Hallnor and S. K. Reinhardt, "A fully associative software-managed cache design," Proceedings of 27th International Symposium on Computer Architecture (IEEE Cat. No.RS00201), Vancouver, BC, Canada, 2000, pp. 107-116, doi: 10.1145/339647.339660. keywords: {Random access memory;Delay;Algorithm design and analysis;Microprocessors;Prefetching;Computer architecture;Laboratories;Computer science;Aging;Application software},

[12] https://www.geeksforgeeks.org/cache-memory-in-computer-organization/

[13] Kumar, D., & Kumar, D. (2023, July 29). *What Is a Direct Mapped Cache? | Baeldung on Computer Science*. Baeldung on Computer Science. https://www.baeldung.com/cs/cache-direct-mapping#:~:text=In%20the%20direct%20mapping%20scheme,the%20index%2C%20and%20the%20offset.

[14] Kumar, D., & Kumar, D. (2023, July 29). *What Is a Direct Mapped Cache? | Baeldung on Computer Science*. Baeldung on Computer Science. https://www.baeldung.com/cs/cache-direct-mapping#:~:text=In%20the%20direct%20mapping%20scheme,the%20index%2C%20and%20the%20offset.

[15] https://www.dauniv.ac.in/public/frontassets/coursematerial/computer-architecture/CompArchCh09L03cacheAssociativity.pdf

[16] Kumar, D., & Kumar, D. (2023, July 29). *What Is a Direct Mapped Cache? | Baeldung on Computer Science*. Baeldung on Computer Science. https://www.baeldung.com/cs/cache-direct-mapping#:~:text=In%20the%20direct%20mapping%20scheme,the%20index%2C%20and%20the%20offset.

[17] Kumar, D., & Kumar, D. (2023, July 29). *What Is a Direct Mapped Cache? | Baeldung on Computer Science*. Baeldung on Computer Science. https://www.baeldung.com/cs/cache-direct-mapping#:~:text=In%20the%20direct%20mapping%20scheme,the%20index%2C%20and%20the%20offset.

[18] Kumar, D., & Kumar, D. (2023, July 29). *What Is a Direct Mapped Cache? | Baeldung on Computer Science*. Baeldung on Computer Science. https://www.baeldung.com/cs/cache-direct-mapping#:~:text=In%20the%20direct%20mapping%20scheme,the%20index%2C%20and%20the%20offset.

[19] Kumar, D., & Kumar, D. (2023, July 29). *What Is a Direct Mapped Cache? | Baeldung on Computer Science*. Baeldung on Computer Science. https://www.baeldung.com/cs/cache-direct-mapping#:~:text=In%20the%20direct%20mapping%20scheme,the%20index%2C%20and%20the%20offset.

Appendix

a. Code in Github:
https://github.com/ZakiyaAbuMurra/Cache---Mlulti-Level-?fbclid=IwAR1hdviWRmaz10AAwN0cnGqudJeCqR63Xrtx3xYCxBNNkvLSSe-olyY3qvE

b. Comparison between different placement algorithms

Direct-mapping	**Fully-Associative Mapping**	**Set-Associative Mapping**
Uses a direct formula to obtain the effective cache address, requiring just one comparison.	In order to determine whether a block is in the cache or not, the cache control logic must compare each block's tag with all tag bits concurrently.	Comparing the number of blocks in each set is necessary since a set may include more than one block.
Three fields represent the main memory address: TAG, BLOCK, and WORD. Together, the BLOCK and WORD form an index. The most significant bits are the Tag bits; the BLOCK bits identify a specific block; and the least significant bits, the WORD bits, indicate a unique word inside a block of main memory.	There is just one field in the main memory address: TAG & WORD.	Three fields represent the main memory address: TAG, SET, and WORD.
Each block from main memory can only have one potential position in the cache hierarchy as a result of our fixed formula.	Any cache block can be used to map the main memory block.	Any direct-mapped cache's specific cache block can be used to map the main memory block.
Cache hit ratio decreases when the processor frequently has to access the same memory address from two separate main memory pages.	If the processor needs to access the same memory location from 2 different main memory pages frequently, cache hit ratio has no effect.	The cache hit ratio decreases when there is frequent access to two separate main memory pages.
The reason for the shorter search time is because each block from main memory can only have one possible position in the cache hierarchy.	The cache control mechanism looks for matches in each block's tag, which increases search time.	The number of blocks in each set increases the search time.
The number of blocks in the cache determines the index.	In the case of fully associative mapping, the index is 0.	The amount of cached sets determines the index.
It has the fewest tag bits.	It has the highest quantity of tag bits.	It has more tag bits than direct mapping and less tag bits than fully associative mapping.
Advantages- • The simplest kind of map · Simple and quick because all that's needed to find a term is tag field matching. · In comparison, the cost is lower than using fully associative mapping.	**Advantages-** • It is fast. • Easy to implement	**Advantages-** · Compared to the direct and fully associative mapping strategies, it performs better.

Disadvantages- · • The replacement of the data-tag value results in low performance.	**Disadvantages-** · It is expensive, because the address must be stored with the data.	**Disadvantages-** • It is most expensive as with the increase in set size cost also increases.