Evaluating the Impact of Cache Capacity on AMAT under Sequential and Shuffled Access Patterns

Ebtihal Al-Ajeeli Ali Al-Mrabet
University of Azzawia, Azzawia, Libya
Faculty of Science
Department of Computer Science,
booto199924@gmail.com

Fathi Hamhoum
University of Azzawia, Azzawia, Libya
Faculty of Science
Department of Computer Science
Fathi.hamhoum@zu.edu.ly

Rawad Masoud Salman
University of Azzawia, Azzawia, Libya
Faculty of Science
Department of Computer Science
rawadsalman@zu.edu.ly

ABSTRACT

Cache memory plays an important role in improving computer system performance by reducing memory access time. This study investigates the effect of cache size on memory access performance using a C++-based cache simulator. Four cache capacities (4 KB, 8 KB, 16 KB, and 32 KB) were evaluated using the Least Recently Used (LRU) replacement policy and two memory access patterns: Sequential and Shuffled. The experiments used a 32 KB working set, a cache line size of 64 bytes, and 100,000 memory accesses for each configuration. Performance was evaluated using cache hits, cache misses, hit rate, miss rate, Average Memory Access Time (AMAT), and relative improvement between consecutive cache sizes. The results showed that the 4 KB, 8 KB, and 16 KB caches achieved an AMAT of 101 ns under the Sequential pattern, whereas the 32 KB cache reduced it to 1.51 ns. Under the Shuffled pattern, AMAT decreased from 88.61 ns for the 4 KB cache to 1.51 ns for the 32 KB cache. These results indicate that cache capacity can significantly affect memory access performance when the cache becomes large enough to accommodate the working set. The study provides a practical comparison of cache sizes under consistent simulation conditions.

Keywords: Cache Size, LRU, Sequential Access, Shuffled Access, AMAT.

الملخص .

تلعب الذاكرة المخبئية دورا مهما في تحسين أداء أنظمة الحاسوب من خلال تقليل زمن الوصول إلى الذاكرة. تدرس هذه الدراسة أثر حجم الذاكرة المخبئية على أداء الوصول إلى الذاكرة باستخدام محاكي ذاكرة مخبئية مبني بلغة.C++ تم تقييم أربع سعات للذاكرة المخبئية (4 كيلوبايت، 8 كيلوبايت، 16 كيلوبايت، و32 كيلوبايت) باستخدام سياسة الاستبدال LRU ونمطين للوصول: المتسلسل والمخلوط. استخدمت التجارب مجموعة عمل بحجم 32 كيلوبايت، وحجم خط للذاكرة 64 بايت، و100,000 عملية وصول للذاكرة لكل إعداد. تم تقييم الأداء باستخدام عدد الإصابات، وعدد الإخفاقات، ونسبة الإصابة، ونسبة الإخفاق، وزمن الوصول المتوسطAMAT ، والتحسن النسبي بين السعات المتتالية. أظهرت النتائج أن ذاكرات 4 و 16و كيلوبايت حققت زمن وصول AMAT قدره 101 نانو ثانية في النمط المتسلسل، بينما خفضته ذاكرة 32 كيلوبايت إلى 1.51 نانو ثانية. وفي النمط المخلوط، انخفض AMAT من 88.61 نانو ثانية لذاكرة 4 كيلوبايت إلى 1.51 نانو ثانية لذاكرة 32 كيلوبايت. تشير هذه النتائج إلى أن سعة الذاكرة المخبئية يمكن أن تؤثر بشكل كبير على أداء الوصول إلى الذاكرة عندما تصبح كبيرة بما يكفي لاستيعاب مجموعة العمل. وتقدم الدراسة مقارنة عملية لأحجام الذاكرة المخبئية في ظل ظروف محاكاة ثابتة.

Received: 26-07-2026 **Accepted: 18-08-2026** **Published: 02-09-2026**

Keywords: Cache Size, LRU, Sequential Access, Shuffled Access, AMAT.

الكلمات المفتاحية: سعة الذاكرة المخبئية - سياسة LRU - الوصول المتسلسل - الوصول المخلوط - متوسط زمن الوصول AMAT.

1. Introduction

Cache memory is an essential component of modern computer architecture, designed to reduce the time required to access data from main memory. By storing frequently accessed data in a smaller and faster memory, cache memory helps reduce memory access delays and improve overall system performance. However, the effectiveness of cache memory depends on several factors, including its capacity and the pattern of memory accesses[1],[2].

Cache size is an important factor in determining cache performance. A small cache may not be able to hold enough data to satisfy repeated memory requests, resulting in a higher number of cache misses. In contrast, a larger cache can accommodate more data and may reduce cache misses, particularly when its capacity is

sufficient to hold the working set. Nevertheless, the benefits of increasing cache size may vary depending on the memory access pattern [1],[3],[4].

This study investigates the effect of cache size on memory access performance through a C++-based cache simulator. Four cache capacities (4 KB, 8 KB, 16 KB, and 32 KB) are evaluated using the Least Recently Used (LRU) replacement policy. The experiments consider two memory access patterns, Sequential and Shuffled, while maintaining consistent simulation conditions.

Cache performance is evaluated using the number of cache hits and misses, hit rate, miss rate, Average Memory Access Time (AMAT), and relative improvement between consecutive cache sizes. By comparing these measurements, the study aims to analyze how cache capacity and memory access patterns affect performance and to determine which cache size provides the best results under the experimental conditions.

The findings of this study provide a practical comparison of cache behavior at different capacities and help illustrate the relationship between cache size, cache hit rate, and memory access time.

2. Related Work

Previous studies have investigated cache memory performance from different perspectives, including cache capacity, replacement policies, memory access time, power consumption, and data access behavior.

Akula et al. investigated the effect of varying L1 instruction and data cache sizes on system performance using simulation-based experiments. The study examines cache performance metrics to understand how cache capacity affects memory access behavior[1].

Manohar and Kapoor focused on improving the efficiency of embedded-DRAM caches through dynamic reconfiguration and optimization of memory refresh operations[5].

Agarwalla et al. investigated cache resizing techniques for DRAM-based last-level caches, focusing on resource utilization, performance, and energy efficiency[3].

Alsharef et al. addressed cache memory performance issues and discusses factors that may affect cache efficiency and memory access behavior[2].

Agarwalla et al. focused on resizing DRAM caches while considering physical variations in memory characteristics to improve cache efficiency[6].

Saha et al. compared STT-MRAM and SOT-MRAM technologies for last-level cache applications, considering characteristics such as access latency, energy consumption, and design area[7].

Dave and Kotak examined cache performance in terms of miss rate and power consumption, considering the influence of cache design parameters on overall efficiency[8].

Hasslinger et al. investigated the performance of FIFO, Clock, and LRU replacement policies and examines the accuracy of analytical and approximate methods for evaluating cache behavior[9].

Einziger et al. focused on using access-time measurements to improve cache performance and inform cache management decisions[10].

Wang et al. investigated the influence of different cache capacities on performance using the gem5 simulator, examining cache behavior and program execution performance[4].

Krause et al. explored the use of reuse distance to improve cache efficiency through proactive decisions about data placement and cache utilization[11].

Do presented LWAL, a lightweight adaptive learning-driven cache-bypassing approach for GPUs to improve cache utilization and memory performance[12].

Despite the contributions of previous studies, this study evaluates the effect of cache capacity on memory access performance under Sequential and Shuffled access patterns within a consistent simulation environment. Four cache sizes (4 KB, 8 KB, 16 KB, and 32 KB) are evaluated using a fixed LRU replacement policy. Hit rate, miss rate, and average memory access time (AMAT) are used to analyze performance, while relative AMAT improvement is calculated between consecutive cache sizes.

3. Methodology

3.1 Simulation Environment

A C++-based cache simulator is developed to investigate the effect of cache capacity on memory access performance. All experiments are conducted under consistent simulation conditions to ensure a fair comparison between different cache sizes and memory access patterns. The simulator uses the Least Recently Used (LRU) replacement policy, with a fixed cache line size of 64 bytes.

The working set consists of 512 distinct cache-line addresses, corresponding to a total working-set size of 32 KB. Each experiment performs 100,000 memory accesses. Four cache capacities are evaluated: 4 KB, 8 KB, 16 KB, and 32 KB.

3.2 Cache Configuration and Access Patterns

The experiments evaluate two memory access patterns: Sequential and Shuffled. In the Sequential pattern, memory addresses are accessed in a repeating sequence from 0 to 511. This pattern represents repeated sequential traversal of the working set.

In the Shuffled pattern, the sequential access trace is copied and its complete sequence of 100,000 accesses is shuffled using a fixed seed shuffling algorithm . This approach provides a reproducible reordered access trace while preserving the same access frequencies as the Sequential pattern.

The cache is fully associative, and the cache replacement policy and cache line size remain fixed throughout the experiments. Only the cache capacity and access pattern vary, allowing their effects on cache performance to be evaluated under consistent conditions.

3.3 Proposed Cache Simulation

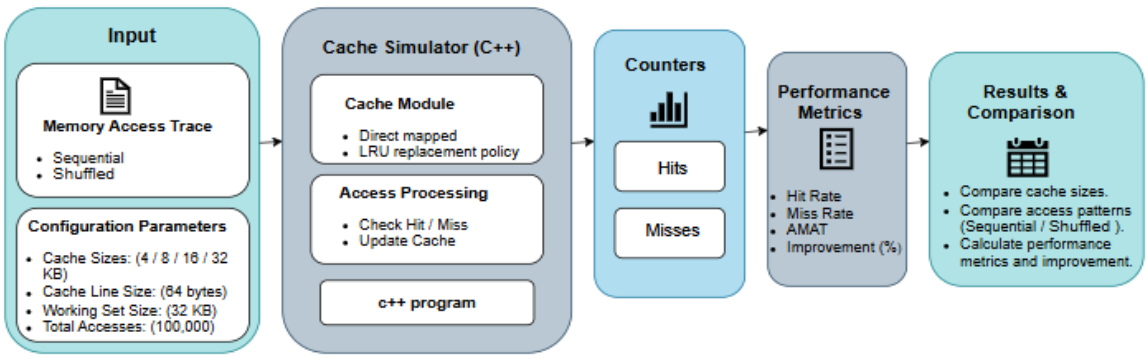


Figure 1: Architecture of the Proposed Cache Simulation Framework

The figure illustrates the overall architecture of the proposed C++ cache simulator. It shows how memory access traces and cache configuration parameters are processed using the LRU replacement policy. The simulator records cache hits and misses, calculates performance metrics, and compares the results across different cache sizes and access patterns.

3.4 Performance Metrics

Four performance metrics are used to evaluate the cache configurations.

3.4.1 Hit Rate

The percentage of memory accesses that find the requested address in the cache.

$$\textbf{Hit Rate}(\%) = \frac{\textbf{Hits}}{\textbf{Total Accesses}} \times 100$$

3.4.2 Miss Rate

The percentage of memory accesses that do not find the requested address in the cache.

$$\textbf{Miss Rate}(\%) = \frac{\textbf{Misses}}{\textbf{Total Accesses}} \times 100$$

3.4.3 Average Memory Access Time (AMAT)

AMAT estimates the average memory access time using an assumed cache hit time of 1 ns and a miss penalty of 100 ns.

$$\textbf{AMAT} = \textbf{Hit Time} + \textbf{Miss Rate} \times \textbf{Miss Penalty}$$

The miss rate is expressed as a decimal fraction when calculating AMAT. These timing values are assumed simulation parameters and do not represent measurements from physical hardware.

3.4.4 Relative AMAT Improvement

This metric measures the percentage reduction in AMAT when moving from a cache size to the next larger size under the same access pattern.

$$Improvement(\%) = \frac{AMAT_{\text{previous}} - AMAT_{\text{current}}}{AMAT_{previous}} \times 100$$

The improvement is calculated separately for each access pattern. The first cache size in each pattern is used as the baseline and therefore has no previous configuration for comparison.

3.5 Experimental Procedure

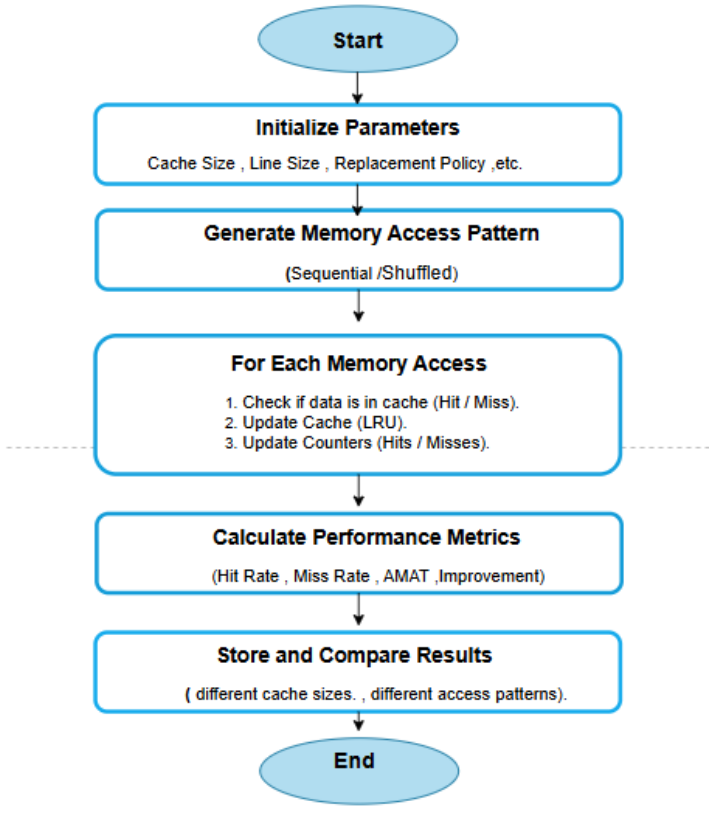


Figure 2: Flowchart of the Simulation Process

The flowchart presents the main steps of the simulation process, starting with parameter initialization and memory access pattern generation. Each memory access is checked to determine whether it results in a cache hit or miss, after which the cache and counters are updated. Finally, performance metrics are calculated, and the results are stored and compared across different cache configurations.

3.6 Simulation Algorithm

Algorithm 1: Cache Simulation Algorithm

Input: Cache sizes (4 KB, 8 KB, 16 KB, and 32 KB), cache line size (64 bytes), and memory access traces.

Output: Hit rate, miss rate, AMAT, and relative AMAT improvement.

1. Generate the Sequential and Shuffled memory access traces.
2. For each cache size, calculate the cache capacity in lines.
3. Initialize the LRU cache.
4. Process all memory accesses and record hits and misses.
5. Calculate the hit rate, miss rate, and AMAT.
6. Calculate the relative AMAT improvement compared with the previous cache size.
7. Compare the results across all cache sizes and access patterns.

4. Implementation

The proposed cache simulator was implemented using C++ to evaluate the effect of cache size on memory access performance. The implementation follows the experimental configuration described in the methodology, ensuring that the cache sizes and access patterns are evaluated under consistent conditions.

The simulator uses the Least Recently Used (LRU) replacement policy to manage cache lines. Each cache line represents 64 bytes, and the working set consists of 512 distinct cache lines, corresponding to a total working-set size of 32 KB. The experiment performs 100,000 memory accesses for each configuration.

Four cache capacities are evaluated: 4 KB, 8 KB, 16 KB, and 32 KB. Two memory access patterns, Sequential and Shuffled, are used to examine cache behavior under different access orders. The Sequential pattern repeatedly accesses the addresses in their original order, whereas the Shuffled pattern uses a shuffled version of the same access sequence with a fixed shuffled seed to ensure reproducibility.

During execution, the simulator records the number of cache hits and misses for each configuration. These values are then used to calculate the hit rate, miss rate, and Average Memory Access Time (AMAT). The AMAT is calculated using an assumed hit time of 1 ns and a miss penalty of 100 ns. The relative improvement in AMAT is also calculated when comparing consecutive cache sizes.

Finally, the simulation results are organized in a table to facilitate comparison between cache capacities and access patterns. These results provide the basis for analyzing the relationship between cache size and memory access performance in the following section.

5. Results and Discussion

5.1 Simulation Results

The simulation results were obtained using a C++ cache simulator to investigate the effect of cache size on memory access performance. Four cache capacities (4 KB, 8 KB, 16 KB, and 32 KB) were evaluated using the Least Recently Used (LRU) replacement policy, a cache line size of 64 bytes, a working set of 32 KB, and 100,000 memory accesses for each experiment.

Two memory access patterns, Sequential and Shuffled, were examined. The collected measurements include cache hits, cache misses, hit rate, miss rate, Average Memory Access Time (AMAT), and relative improvement. The results are summarized in Table 1.

Table 1. Simulation Results for Different Cache Sizes and Access Patterns

Cache Size (KB)	Pattern	Hits	Misses	Hit Rate (%)	Miss Rate (%)	AMAT (ns)	Improvement (%)
4	Sequential	0	100,000	0.00	100.00	101.00	N/A
4	Shuffled	12,387	87,613	12.39	87.61	88.61	N/A
8	Sequential	0	100,000	0.00	100.00	101.00	0.00%
8	Shuffled	24,931	75,069	24.93	75.07	76.07	14.16%
16	Sequential	0	100,000	0.00	100.00	101.00	0.00%
16	Shuffled	49,752	50,248	49.75	50.25	51.25	32.63%
32	Sequential	99,488	512	99.49	0.51	1.51	98.50%
32	Shuffled	99,488	512	99.49	0.51	1.51	97.05%

Table 1 presents the simulation results for the four cache sizes under Sequential and Shuffled access patterns. The results include cache hits, cache misses, hit rate, miss rate, AMAT, and relative improvement. These measurements provide an overview of how cache capacity affects memory access performance under the tested conditions.

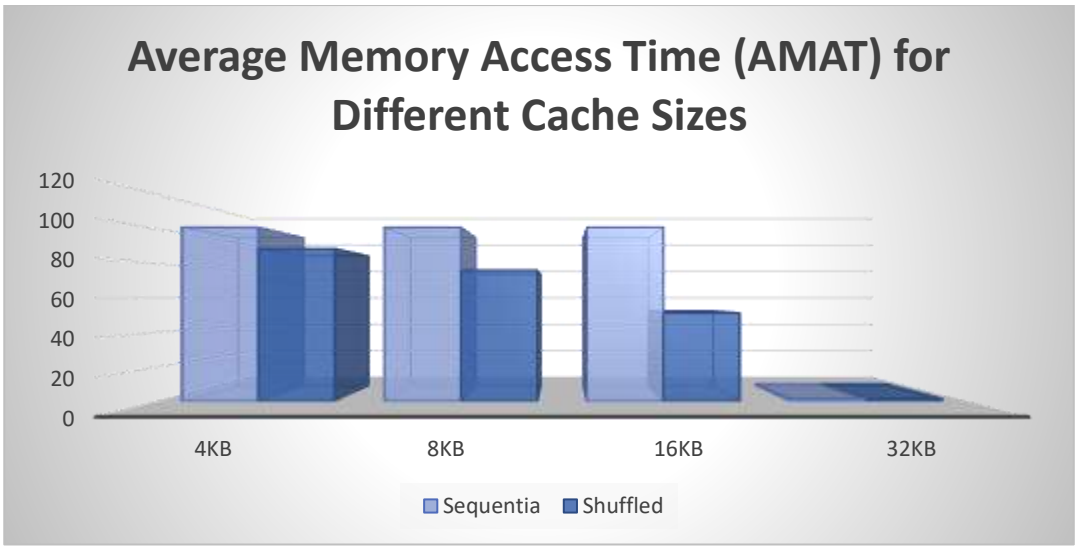


Figure 3: Average Memory Access Time (AMAT) for Different Cache Sizes

5.2 Discussion

The results show that cache size has an important effect on memory access performance. As the cache size increases, the Shuffled access pattern shows a gradual improvement in hit rate and a reduction in AMAT. The hit rate increases from 12.39% for the 4 KB cache to 49.75% for the 16 KB cache, while AMAT decreases from 88.61 ns to 51.25 ns. At 32 KB, both access patterns achieve a hit rate of 99.49% and an AMAT of 1.51 ns.

Under the Sequential access pattern, the 4 KB, 8 KB, and 16 KB caches achieved a hit rate of 0%. This is because the working set consists of 512 cache lines (32 KB), which is larger than the capacity of these caches. Since the cache cannot hold the entire working set, the LRU replacement policy evicts previously accessed cache lines as new lines are loaded. When the access sequence repeats, the required lines have already been evicted before they can be accessed again, resulting in a cache miss for every access. In contrast, the 32 KB cache can accommodate the entire working set, allowing previously accessed lines to be reused and achieving a hit rate of 99.49%.

These findings indicate that a cache capacity equal to the working-set size can significantly reduce cache misses and memory access time under the tested conditions.

6. Conclusion and Future Work

This study investigated the effect of cache size on memory access performance using a C++-based cache simulator. Four cache capacities (4 KB, 8 KB, 16 KB, and 32 KB) were evaluated using the Least Recently Used (LRU) replacement policy under Sequential and Shuffled memory access patterns. The evaluation was based on cache hits, cache misses, hit rate, miss rate, Average Memory Access Time (AMAT), and relative improvement.

The results showed that increasing cache capacity improved performance when the cache was large enough to accommodate the working set. The 4 KB, 8 KB, and 16 KB configurations recorded an AMAT of 101 ns for the Sequential pattern, while the 32 KB cache achieved an AMAT of 1.51 ns. For the Shuffled pattern, AMAT

decreased from 88.61 ns with a 4 KB cache to 1.51 ns with a 32 KB cache. These findings demonstrate that cache capacity and memory access patterns can significantly affect cache performance under the tested conditions.

For future work, the simulator could be extended to evaluate additional cache sizes, different cache line sizes, and alternative replacement policies, such as FIFO and LFU. Further experiments could also investigate different working-set sizes and memory access patterns. In addition, the simulation could be extended to study multilevel cache architectures and energy consumption to provide a broader understanding of cache performance.

7. References

[1] R. Akula, K. Jain, and D. J. Kotecha, "System Performance with varying L1 Instruction and Data Cache Sizes: An Empirical Analysis," Nov. 26, 2019, *arXiv*. arXiv:1911.11642. doi: 10.48550/arXiv.1911.11642.

[2] A. Alsharef, P. Jain, M. Arora, S. R. Zahra, and G. Gupta, "Cache memory: an analysis on performance issues," in *2021 8th international conference on computing for sustainable global development (INDIACom)*, IEEE, 2021, pp. 184—188. Accessed: Oct. 11, 2026. [Online]. Available: https://www.researchgate.net/profile/Sonia-Garg-2/publication/352825606_Cache_Memory_An_Analysis_on_Performance_Issues/links/61a74c0e85c5ea51abc25042/Cache-Memory-An-Analysis-on-Performance-Issues.pdf

[3] B. Agarwalla, S. Das, and N. Sahu, "Efficient cache resizing policy for DRAM-based LLCs in ChipMultiprocessors," *Journal of Systems Architecture*, vol. 113, p. 101886, 2021, Accessed: Oct. 11, 2026. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1383762120301582

[4] R. Wang, L. Yu, and W. Zhuang, "Research on the impact of Gem5-based cache capacity on cache performance," in *2023 4th International Conference on Machine Learning and Computer Application*, Hangzhou China: ACM, Oct. 2023, pp. 392—395. doi: 10.1145/3650215.3650284.

[5] S. S. Manohar and H. K. Kapoor, "Dynamic reconfiguration of embedded-DRAM caches employing zero data detection based refresh optimisation," *Journal of Systems Architecture*, vol. 100, p. 101648, 2019,

Accessed: Oct. 11, 2026. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1383762119304552

[6] B. Agarwalla, S. Das, and N. Sahu, "Process variation aware DRAM-Cache resizing," *Journal of Systems Architecture*, vol. 123, p. 102364, 2022, Accessed: Oct. 11, 2026. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1383762121002538

[7] R. Saha, Y. P. Pundir, and P. K. Pal, "Comparative analysis of STT and SOT based MRAMs for last level caches," *Journal of magnetism and magnetic materials*, vol. 551, p. 169161, 2022, Accessed: Oct. 11, 2026. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0304885322001275

[8] H. V. Dave and N. A. Kotak, "Critical analysis of cache memory performance concerning miss rate and power consumption," *IJES*, vol. 15, no. 6, p. 516, 2022, doi: 10.1504/IJES.2022.129810.

[9] G. Hasslinger, K. Ntougias, F. Hasslinger, and O. Hohlfeld, "Scope and accuracy of analytic and approximate results for FIFO, clock-based and LRU caching performance," *Future Internet*, vol. 15, no. 3, p. 91, 2023, Accessed: Oct. 11, 2026. [Online]. Available: https://www.mdpi.com/1999-5903/15/3/91

[10] G. Einziger, O. Himelbrand, and E. Waisbard, "Boosting Cache Performance by Access Time Measurements," *ACM Trans. Storage*, vol. 19, no. 1, pp. 1–29, Feb. 2023, doi: 10.1145/3572778.

[11] A. M. Krause, P. C. Santos, A. F. Lorenzon, and P. O. Navaux, "HBPB, applying reuse distance to improve cache efficiency proactively," *Journal of Parallel and Distributed Computing*, vol. 191, p. 104919, 2024, Accessed: Oct. 11, 2026. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0743731524000832

[12] C. T. Do, "LWAL: Lightweight Adaptive Learning-Driven Cache Bypassing for GPUs," *IEEE Computer Architecture Letters*, 2026, Accessed: Oct. 11, 2026. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/11430541/