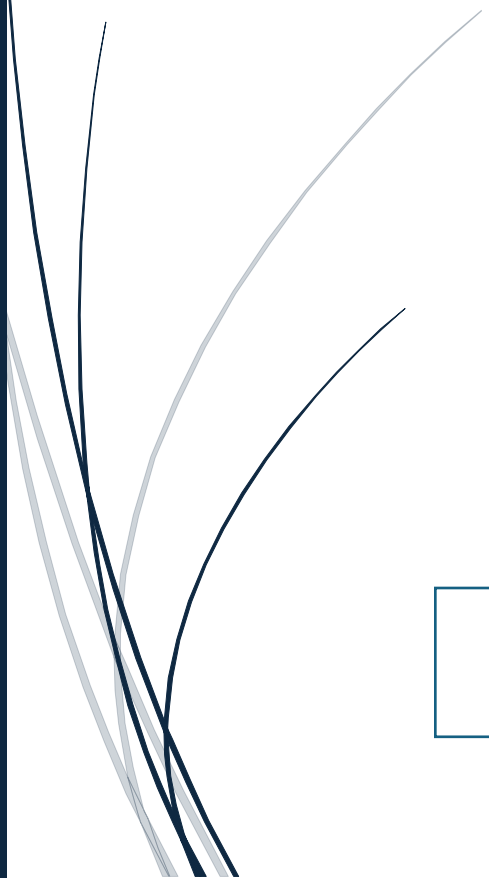
The American University in Cairo
School of Science and Engineering

Summer 2025

Cache Simulator Report

Computer Organization and Assembly Language

Prepared by
Habeba Saad
Doha Nour El-Din
Kareem Rashed
Yousef Elmenshawi



Supervisor:

Dr. Mohamed Shalan

1. Introduction

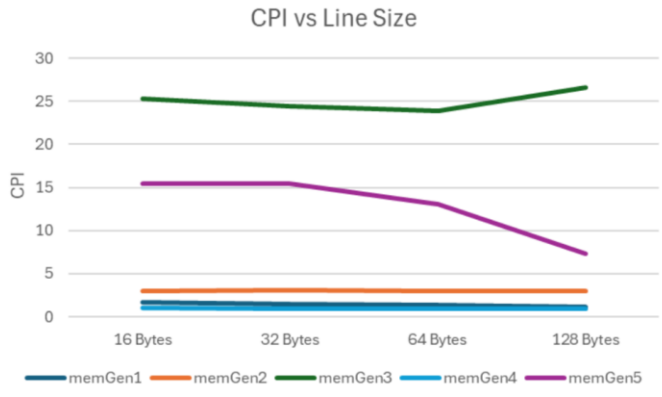
The project is built to minimize memory latency through multi-level cache hierarchies. The primary objective of this project is to simulate a two-level data cache system and evaluate its performance under varying workloads and configurations. Specifically, the simulation measures the effective Cycles Per Instruction (CPI) for a system with a 16KB L1 cache and a 128KB L2 cache using a write-back policy and random replacement. The goal is to:

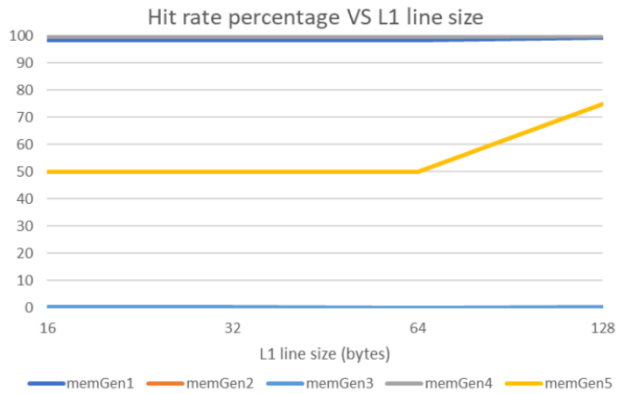
- Observe how different L1 cache line sizes (16, 32, 64, and 128 bytes) influence performance across a variety of memory access patterns, generated by five synthetic memory generators (memGen1 to memGen5).

This report documents the simulation methodology, presents the collected data, analyzes the simulation process, and examines how cache parameters affect performance. The primary performance is effective CPI, derived from simulating one million instructions, 35% of which are memory-related.

2. The Analysis of CPI & Hit Ratio vs. Line Size

This experiment analyzes cache performance using different memory access generators. The CPI is measured by changing the L1 cache line sizes of 16, 32, 64, and 128 bytes.



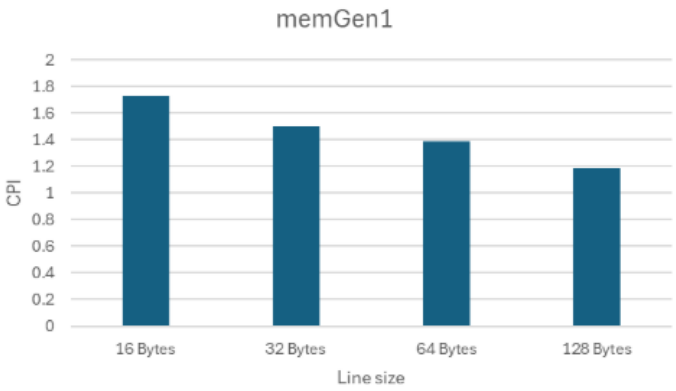


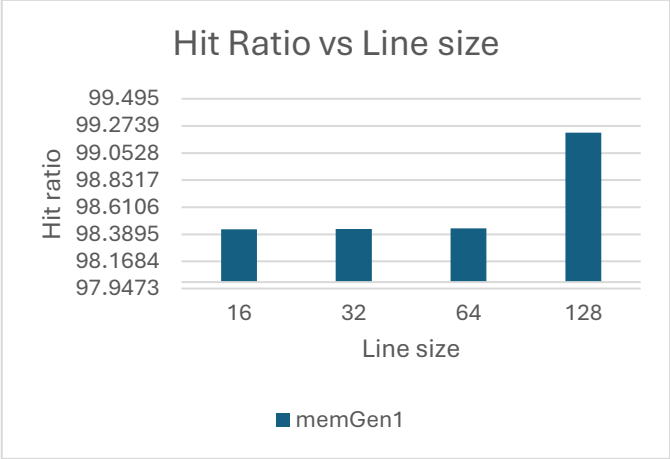
The graph clearly shows that the memory generator with the best performance is memGen4, while the one with the worst is memGen3. That is clearly due to the high CPI for memGen3 and the low CPI for memGen4. Consequently, the hit ratio for memGen4 is the best among the other functions, and memGen5 performs the worst hit ratio. This means there is an inverse relation between the hit ratio and CPI performance. The reasons for this difference will be clearly elaborated through the following analysis for each generator.

memGen1: Sequential Access Pattern

The following table and graph display the data collected from varying L1 line sizes and their impact on effective CPI.

L1 Line Size	Effective CPI	Hit Ratio
16 Bytes	1.73	98.4301
32 Bytes	1.50	98.4341
64 Bytes	1.39	98.4374
128 Bytes	1.19	99.2186





Observation

- memGen1() generates **sequential memory accesses**: 0, 1, 2, 3, ..., which shows **strong spatial locality**.

- As **L1 cache line size increases**, each cache miss brings in **more adjacent words** since:
  - 16B line = 4 words
  - 128B line = 32 words
- This means **more upcoming accesses hit the cache**, reducing the number of costly L2 or DRAM accesses.
- Effective CPI drops steadily with increasing line size, as shown in the graph.
- Performance improves significantly due to fewer L1 misses, especially for a workload with such predictable access.
- The best CPI occurs at **128B**, confirming that **larger cache lines are highly effective** for sequential access patterns like memGen1().

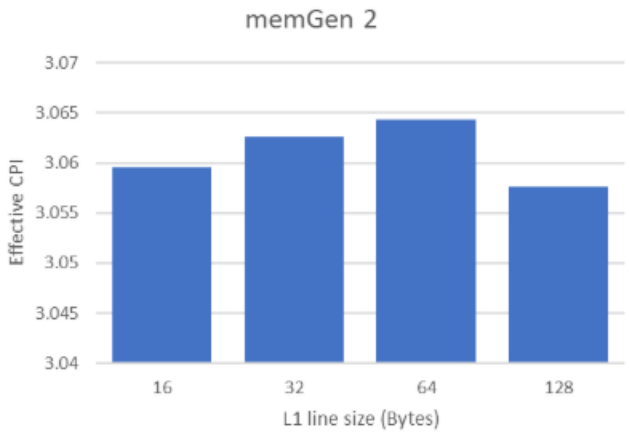
Detailed Analysis

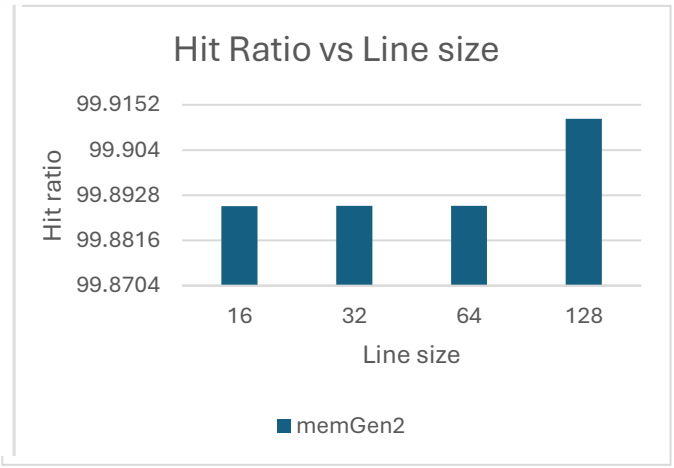
The memGen1() function generates sequential memory accesses 0, 1, 2, 3, and so on within a DRAM size of 64 MB. This access pattern shows strong spatial locality, making it a better test for how the cache line size impacts performance. As the L1 cache size increases, each cache miss results in a larger block of adjacent memory being stored in the cache. For example, a 16-byte line size brings 4 adjacent words into the cache, while a 128-byte line brings 32 words into the cache. This increases the probability of the upcoming memory accesses hitting the cache, which, as a result, reduces the number of expensive accesses to L2 cache or DRAM. Consequently, the effective CPI drops from 1.73 (16-byte line size) to 1.19 (128-byte line size), which demonstrates a steady improvement in performance as line size increases. On the other hand, temporal locality is not the best because the address generator function is traversing 64 MB (the whole DRAM memory), which differentiates this model from memGen4, which has a better range (4 KB), which in turn improves temporal locality and CPI.

memGen2: Random Access, Small Range

The following table and graph display the data collected from varying L1 line sizes and their impact on effective CPI.

L1 Line Size	Effective CPI	Hit Ratio
16 Bytes	3.0595	99.8901
32 Bytes	3.0626	99.8902
64 Bytes	3.0643	99.8902
128 Bytes	3.0576	99.9117





Analysis

memGen2() generates memory addresses randomly but restricts them to a small range (0–24 KB). Although the addresses are selected randomly, the limited range increases temporal locality such that the same addresses are likely to be reused over time. However, spatial locality remains low because accesses within that range are still randomly scattered, making it unlikely for consecutive addresses to be fetched together in a single cache line. This moderate temporal locality helps limit the CPI increase compared to fully random patterns over a wider range, like in memGen3().

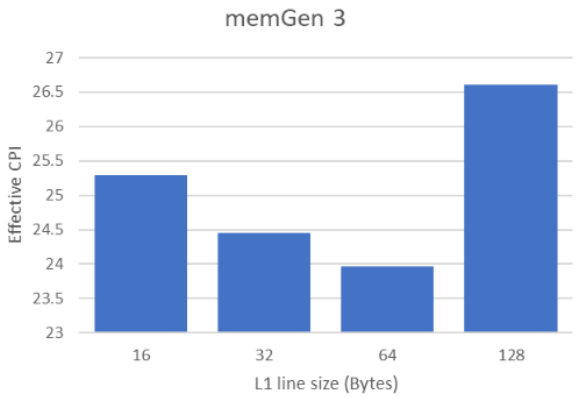
When comparing CPI concerning the L1 line size, we observe that CPI increases from 16 to 64 bytes, then slightly drops at 128 bytes. In the smaller line sizes (16–64 bytes), the low spatial locality means that larger lines do not improve performance and may even increase CPI due to higher miss penalties as more data is fetched per miss. At 128 bytes, the cache line covers a larger memory span, and due to the limited address range, this starts to improve the effective spatial coverage. Also, the temporal locality increases more, resulting in a slight CPI improvement. This results in extremely high hit ratios across all line sizes, consistently around 99.89%, with only a marginal increase at 128 bytes due to slightly better spatial coverage. However, the hit ratio is the same in the sizes (16-64B) regardless of the small change of CPI, which does not affect the hit rate.

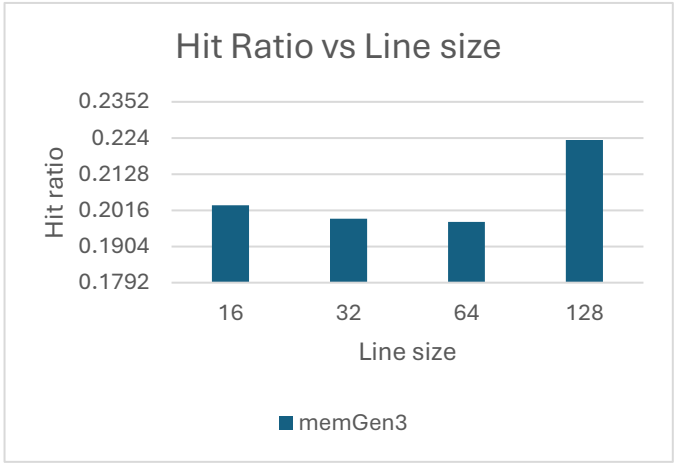
Randomizing the memory address in this way still produces all three types of cache misses, specifically compulsory and conflict misses, which decrease the performance compared to the sequential memory address. However, because the working set is small, the CPI remains relatively low compared to generators with a larger or fully random working set like memGen3(). Hence, memGen2() generates poor spatial locality but maintains decent temporal locality due to the small range. As a result, performance is acceptable, and line size has a limited effect, except at very high sizes, where a bit more spatial locality is captured, in addition to the highest hit ratio that is inconsistent with the CPI values.

memGen3: Random Access, Large Range

The following table and graph display the data collected from varying L1 line sizes and their impact on effective CPI and hit ratio.

L1 Line Size	Effective CPI	Hit Ratio
16 Bytes	25.2993	0.2032
32 Bytes	24.4529	0.199
64 Bytes	23.9685	0.198
128 Bytes	26.6034	0.2234





Observation

- The CPI Performance is the worst in this generation since it is very high.
- The hit rate is very low despite the change in line size.
- The cache fails in accessing the random and large patterns.

Analysis

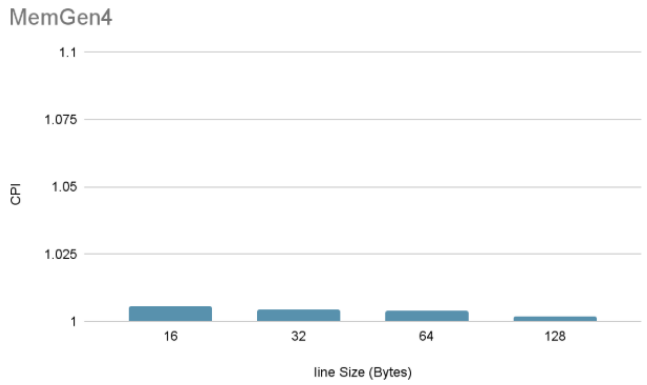
memGen3() generates the memory addresses **randomly**. Unlike memGen2(), this memory generator has a wide range for generating any memory address across all DRAM. Compared to all other memory generators, this one has the worst CPI since it has the highest CPI. This higher CPI can result from a large number of misses, including all three types of misses. This is shown clearly from the low hit ratio, as shown in the graph. The largest value for the hit ratio is almost 0.2%. This large number of misses results from randomization in a very large range of addresses, resulting in a very poor spatial and temporal locality.

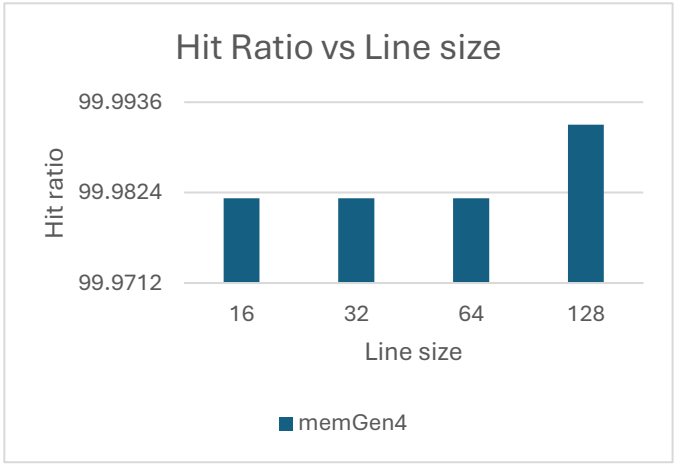
The cache in this memory generator has the least effect on accessing the data due to randomization, resulting in accessing memory more. Unlike the sequential memory access generators, there is no clear relation between the CPI and L1 line size, since the data are completely random and the range is too big. The high effective CPI performance is along L1 sizes 16B and 128B, which emphasizes the randomness and non-relationship with the line size. That also results in a random hit ratio since the hit ratio reaches its highest rate at the same time the CPI performance is the highest in line size 128B. Hence, memGen3() has the worst performance due to randomizing the memory address over a very wide range.

memGen4: Sequential Access, Wraps around 4KB

The following table and graph display the data collected from varying L1 line sizes and their impact on effective CPI and hit ratio.

L1 Line Size	Effective CPI	Hit Ratio
16 Bytes	1.0058	99.9817
32 Bytes	1.0045	99.9817
64 Bytes	1.0038	99.9817
128 Bytes	1.0019	99.9908





Observation

- The 4KB wrap-around range aligns perfectly with the L1 cache size, effectively eliminating capacity and conflict misses after the initial warm-up.
- Larger line sizes reduce the number of compulsory misses at the beginning by pulling in more bytes per miss, covering the 4KB range faster, hence the hit ratio is effectively increasing.
- The strictly sequential nature of accesses makes this the most cache-friendly pattern, maximizing both spatial and temporal locality.
- Once the first 4KB are loaded, the system experiences near-constant hits for the rest of the simulation, resulting in a CPI that stays very close to 1.
- L2 and DRAM are effectively bypassed after the initial load, playing no role in the performance for this access pattern.
- As line size increases, the CPI quickly converges to the theoretical ideal, highlighting the benefits of good locality with sufficient cache size.
- This pattern can be considered the ideal or best-case scenario for cache behavior, useful for benchmarking other, less efficient patterns.

Analysis

MemGen4 is quite similar in nature to MemGen1, the key difference lies within the fact that in this address generating function, the sequential access wraps around 4KB only, not the entire DRAM size. This makes a huge difference in Temporal and Spatial Locality, since our L1 cache size alone is 16KB, meaning that we would have no capacity nor conflict misses, significantly reducing CPI.

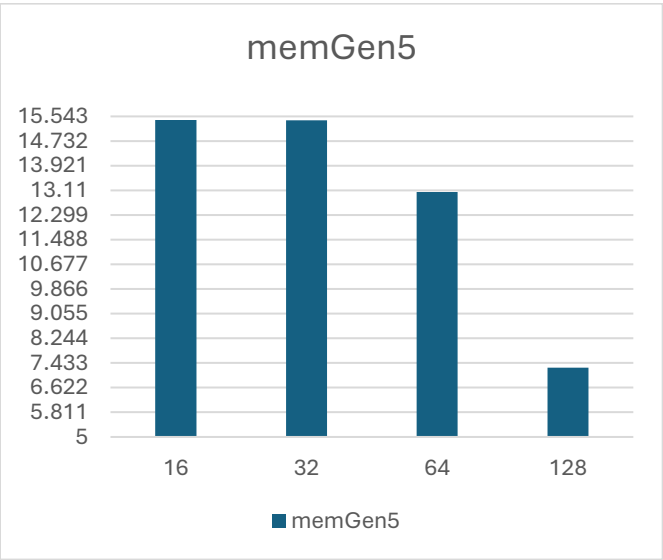
The graph, with the Y-Axis being reduced to a range from 1 to 1.1 to show the very small difference, still shows a steady decrease in CPI as Line Size increases. This happens for the exact same reason as in MemGen1; greater line size leads to fetching more adjacent bytes, which will be accessed sequentially soon.

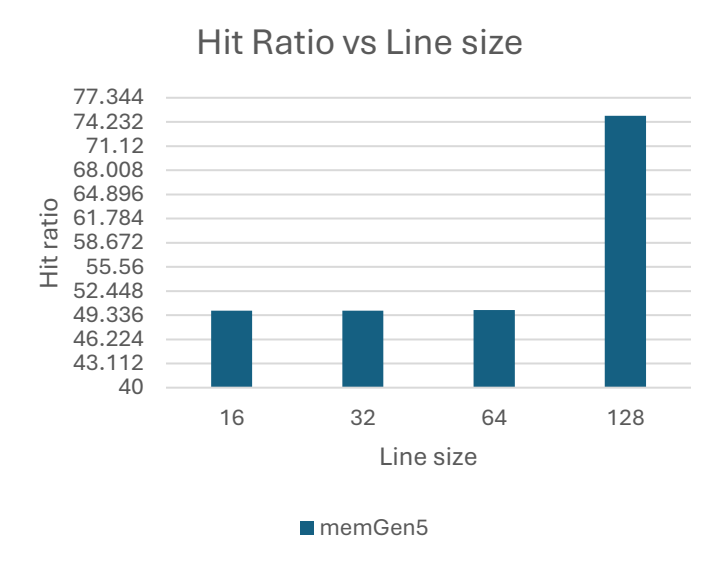
Temporal Locality is also much greater than MemGen1 since the entire 4KB are already inside the cache, so once the address generator wraps around 4KB the first time, it will only hit for the remainder of the 1 Million accesses, this means that the only misses we encounter here are the cold start compulsory misses before each line is fetched for the first 4KB. Also, notice that L2 is not being accessed at all at any point since our L1 cache size is greater than 4KB. Hence, this address generating function is the most ideal possible scenario for our simulation, where the only misses we encounter are at the very beginning due to cold start. Then, everything remains in the cache.

memGen5: Sequential Access (every 32B), Wraps around 1MB

The following table and graph display the data collected from varying L1 line sizes and their impact on effective CPI and hit ratio.

L1 Line Size	Effective CPI	Hit Ratio
16 Bytes	15.4238	49.9273
32 Bytes	15.4188	49.9277
64 Bytes	13.0599	50.0057
128 Bytes	7.2817	75.0024





Observation

- Initially, the CPI values are high in the line sizes (16–32B) with minimum difference, and is gradually declining in the size 64B. Suddenly, it dropped to almost half its value in the line size 128B.
- However, the hit ratio did not change its value except for the size 128B as it rose to 75%.

- The access pattern is deterministic but skips every 32B, so the data with small sizes is missing, which emphasizes why the 128B size has a high hit rate because of its capacity. That shows the benefits of spatial locality.
- DRAM and L2 are frequently accessed at lower line sizes, resulting in high penalties and poor CPI.
- As line size increases, L1 cache becomes more effective, reducing reliance on lower levels of memory.

Analysis

memGen5() generator simulates a sequential access pattern in which the memory is accessed every 32B and wraps around 1MB. This makes the special locality not accessible for small sizes, which results in a 50% missing ratio for the data in L1, as shown in the graph. For that reason, there is a penalty cost on L2 and the DRAM with a high value for the CPI in the sizes (16-32B). The changes are slightly improved in the size 64B where the CPI has decreased. The significant change of the improvement occurred in the size 128B since the CPI dropped to almost 7, which is half of the first two sizes, and the hit ratio jumped to 75%.

This significant change in 128B occurred because of fetching 4 consecutive 32B strides for a line (Miss, Hit, Hit, Hit). Because of having multiple accessing operations in a single cache, the miss rate decreases and triggers the write back every time to access the data. For that reason, the special locality is improved by increasing the line size.

Moreover, the conflict, CPI values, and miss rate increase while the L1 size is less than 32B, which specifies a specific range for the hit ratios, unlike memGen4, which fits the entire L1 sizes. memGen5 focuses on the larger set sizes for the function to have effective results.

3. Conclusion

The aim of the project is to measure the CPI and hit rate changes in the memory at different levels and functions. The L1 plays a crucial role in the performance of the memory function, as shown in the different tests and also by varying its line sizes (16B, 32B, 64B, and 128B). In addition, the spatial locality behaviors have effects on some functions like memGen4 and memGen1, which also affect the hit ratio and CPI. Strong spatial locality involves lower CPI and a higher hit ratio. In addition, the cache line size affects some memory functions and has no effect on others; it is necessary to have a larger size in memGen5 function to have effective results, while increasing/ decreasing the size will not affect the random access patterns wisely because of generating random data.

Moreover, the temporal locality can be a solution for accessing more data in a hit rate like memGen2 as a slight CPI improvement resulted from increasing the temporal locality. This results in extremely high hit ratios across all line sizes, consistently around 99.89%. On the other side, the uniform random access works as the worst pattern because it destroyed both spatial and temporal locality which results in high CPI values and low hit rate regardless the line sizes.

Overall, the cache memory design is workload-dependent since there are different behaviors that affect its performance, and there is a unique performance that results from each pattern. On the other hand, the random access patterns should have techniques and algorithms to reduce the randomness. Consequently, this simulation served as a functional testbed to validate cache logic and memory behavior and also supports both theoretical understanding and practical performance tuning for hierarchical memory systems.