Two-Level Cache Simulator Performance Report

Computer Org. & Assembly Lang. - Project 2 Report - CSCE 2303
The American University in Cairo
Instructor: Dr. Mohammed Shalan

Ahmed Elzahaby - Youssef Hawash - Moaz El Damery

July 22, 2025

1. Introduction

In modern computer architectures, memory latency presents a critical performance bottleneck. While processors have significantly increased in speed, accessing data from the main memory remains relatively slow. Therefore, in an attempt to mitigate this disparity, modern CPUs have employed multi-level cache hierarchies that store frequently accessed data closer to the processor.

This project aims to investigate the performance of a two-level cache hierarchy using a custom-built simulator which models a set-associative, write-back caching system with configurable L1 line sizes. Thus, by incorporating realistic memory access patterns and penalties for cache misses, we aim to analyze how variations in cache line size affect the overall system performance, measured in terms of effective cycles per instruction (CPI), and analyze CPI results across different memory access generators and line sizes, ultimately helping to determine optimal cache configurations under realistic workloads.

2. Cache Simulator Design

2.1 Memory Hierarchy Description

The simulator models a two-level cache hierarchy alongside main memory. The specifications for each level are as follows:

- **L1 Cache**
  - Size: 16 KB
  - Line size: Variable (16 B, 32 B, 64 B, 128 B)
  - Associativity: 4-way set associative
  - Hit time: 1 cycle
- **L2 Cache**
  - Size: 128 KB
  - Line size: Fixed (64 B)
  - Associativity: 8-way set associative
  - Hit time: 10 cycles
- **Main Memory (DRAM)**
  - Size: 64 GB
  - Access penalty: 50 cycles

2.2 System Assumptions

- 35% of the executed instructions are memory operations (loads/stores).
- 50% of the memory accesses are reads, and 50% are writes.
- Instruction fetches are handled by an ideal memory and do not affect CPI.
- Write-back cache policy is employed.
- Random replacement policy is used.
- The ideal CPI (100% L1 hit rate) is 1.

2.3 Simulator Algorithm Overview

The simulator runs a loop for one million instruction cycles. The structure is as follows:

1. Generate a random number p, ranging from 0 to 1.
2. If $p \leq 0.35$, simulate a memory instruction:
   - Generate a memory address via one of five memory generators.
   - Generate a random number $rdwr$, ranging from 0 to 1.
     - If $rdwr$ is less than 0.5, then READ.
     - Else, WRITE
   - Pass the address to the L1 cache:
     - On L1 hit: $cycles++$
     - On L1 miss: $cycles+=1+10$ (Penalty of accessing the L2 cache)
   - If L1 miss, pass the address to the L2 cache:
     - On L2 hit: $cycles++$
     - On L2 miss: $cycles+=1+10+50$ (Penalty of accessing the DRAM)
   - If L2 miss, pass the address to the DRAM.
3. If $p > 0.35$, count as a non-memory instruction: $cycles++$

The CPI is calculated as:

$$\text{CPI} = \frac{\text{Total Cycles}}{\text{1,000,000}}$$

3. Implementation Plan

The simulator is implemented in a modular architecture using *TBD*. Key components include:

- **CacheLine** is a structure which holds the tag, valid, and dirty bits.
- **The cache vectors** are 2D Array meant to simulate the behavior of a real-life cache.
- **Memory generator modules** to simulate the different access patterns.
- **Instrumentation** for tracking total cycles and hit/miss statistics.

The simulator decodes memory addresses based on cache configuration (line size and associativity) and applies random replacement on cache misses. The write-back and dirty-bit logic are enforced for memory consistency.

4. Experimental Setup

The simulation plan includes:

- **5 memory generators**: Each with a distinct pattern (e.g., random, sequential).
- **4 L1 line sizes**: 16 B, 32 B, 64 B, 128 B.
- **20 total simulations**: Each combination of generator and line size.

Each simulation will report the effective CPI with the final results being visualized using line graphs that plots the CPI vs line size for each generator.

5. Simulation Results

In this section, each generator will be analyzed in its own subsection, along with its corresponding graph and table. Below is a graph displaying the CPI, on the Y-axis, vs the L1 line size, on the X-axis, with all five curves representing the five memory generators:

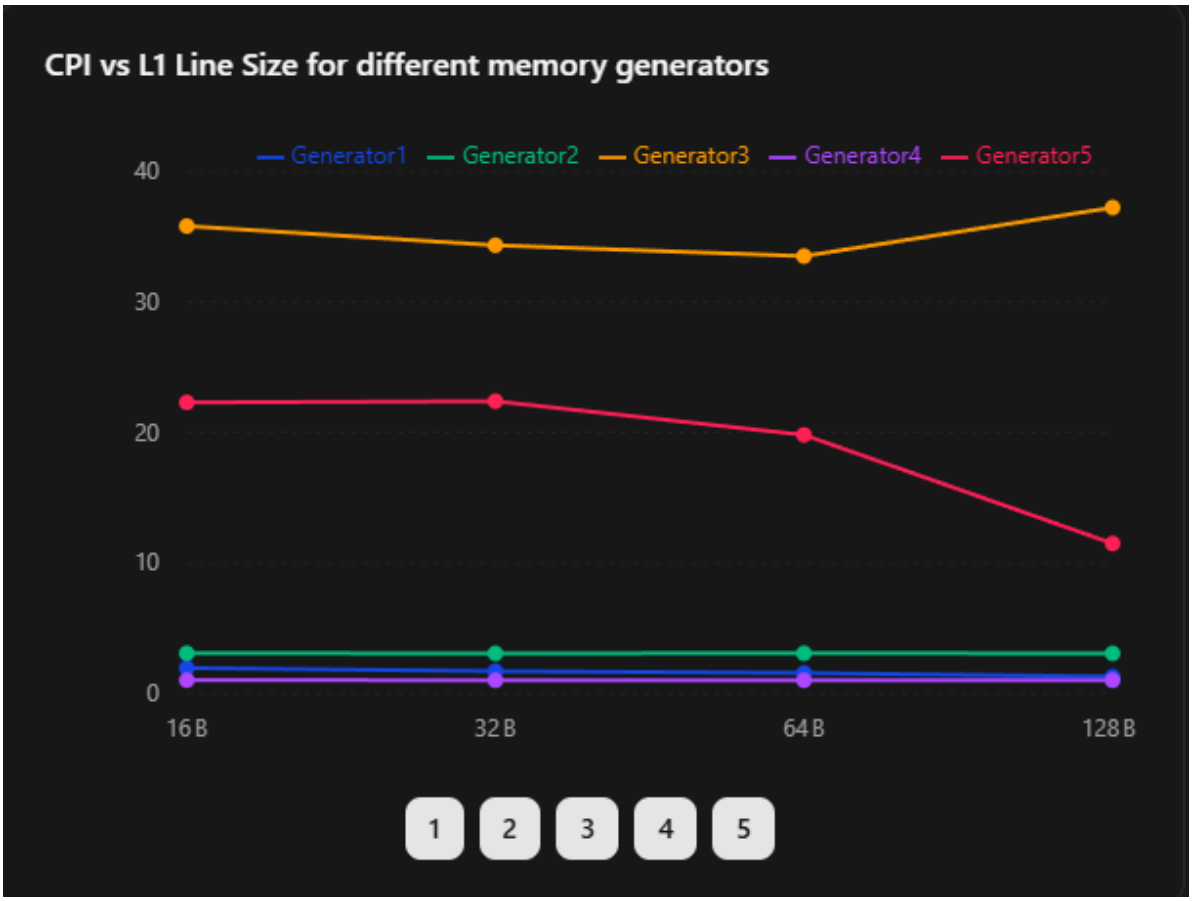


Figure 1: All generators graph

5.1 Generator 1 Analysis

Figure 2: Generator 1 Graph

CPI	Generator	LineSize	L1_Hit	L1_Miss	L2_Hit	L2_Miss	L1_WriteBacks	L2_WriteBacks	L1_MissRate	L2_MissRate
1.94	1	16	328180	21879	36530	6204	20855	4097	0.0625	0.1452
1.7	1	32	339303	10946	15480	5900	10434	3734	0.0313	0.276
1.57	1	64	345242	5481	4965	5741	5225	3419	0.0156	0.5362
1.28	1	128	346732	2731	2455	2879	2603	1698	0.0078	0.5397

Figure 3: Generator 1 Table

After analyzing the results from generator 1, the following has been observed: Because Generator 1 issues strictly sequential addresses, it maximizes spatial locality: each cold miss brings in a contiguous block of data that the very next accesses will use. As you increase the line size from 16 B to 128 B, each miss fetches more useful bytes, driving the L1 miss rate down (from 6.25 % at 16 B to 0.78 % at 128 B). The result is a smooth, monotonic decline in CPI, falling from 1.94 at 16 B to 1.28 at 128 B.

5.2 Generator 2 Analysis

Figure 4: Generator 2 Graph

CPI	Generator	LineSize	L1_Hit	L1_Miss	L2_Hit	L2_Miss	L1_WriteBacks	L2_WriteBacks	L1_MissRate	L2_MissRate
3.07558	2	16	232852	117994	205254	384	87644	0	0.3363	0.0019
3.05957	2	32	233577	116903	203653	384	87134	0	0.3336	0.0019
3.06463	2	64	233286	117016	204159	384	87527	0	0.334	0.0019
3.05897	2	128	233571	116658	203593	384	87319	0	0.3331	0.0019

Figure 5: Generator 2 Table

After analyzing the results from generator 2, the following has been observed: Generator 2 issues uniform random accesses within a 24 KB region. Because the working set (24 KB) exceeds the 16 KB L1 cache, L1 sees a steady miss rate of about 33 % regardless of line size (e.g. 0.336 at 16 B $\rightarrow$ 0.332 at 128 B). However, that entire 24 KB region still comfortably fits in the 128 KB L2 cache, so nearly every L1 miss is satisfied by an L2 hit (L2 miss rate 0.0033).

5.3 Generator 3 Analysis

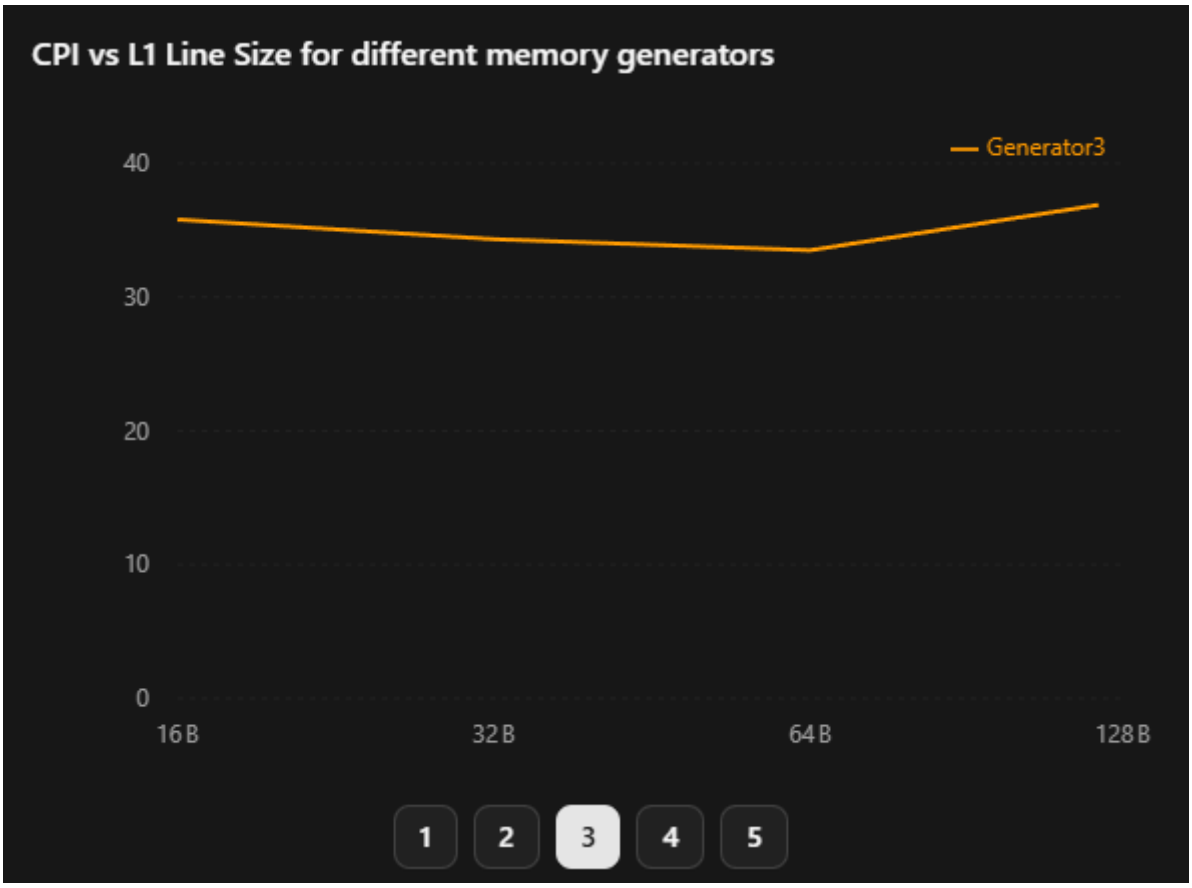


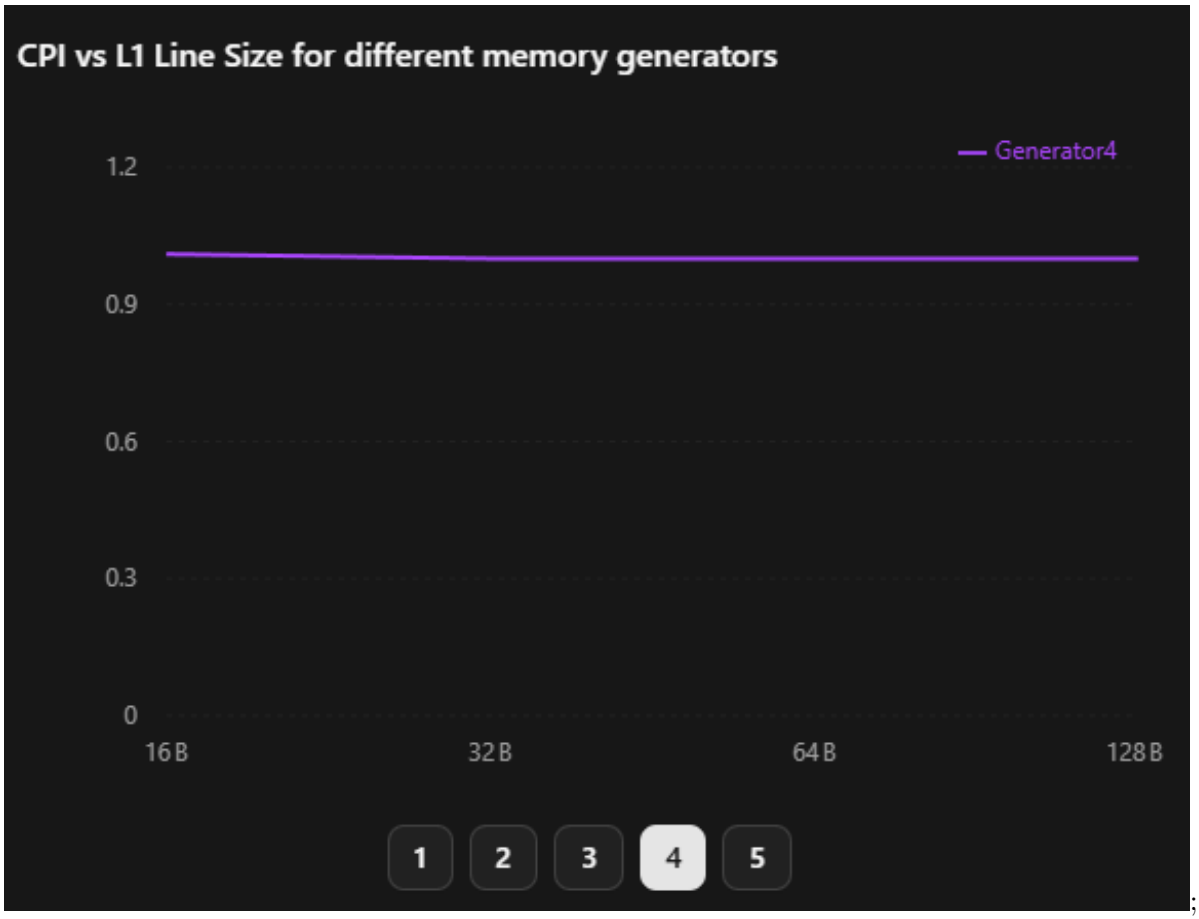
Figure 6: Generator 3 graph

CPI	Generator	LineSize	L1_Hit	L1_Miss	L2_Hit	L2_Miss	L1_WriteBacks	L2_WriteBacks	L1_MissRate	L2_MissRate
35.82055	3	16	2	350123	108210	417080	175167	174273	1	0.794
34.44494	3	32	4	350327	135420	389844	174937	174002	1	0.7422
33.49827	3	64	3	349977	153300	371337	174660	173701	1	0.7078
37.19938	3	128	3	349761	79690	444908	174837	174160	1	0.8481

Figure 7: Generator 3 Table

After analyzing the results from generator 3, the following has been observed: Generator3 drives a purely random stream over a 1MB region, so it completely destroys spatial locality and incurs very high CPI. You see a downward slope as line size grows, as larger lines bring in more bytes per miss, slightly boosting the chance that a subsequent random access hits data already in the cache. However, as soon as the L1 line size exceeds the L2 line size, each L1 block spans *two* L2 blocks. In that matter every L1 write-back populates only the first half of the corresponding L2 block, effectively "demolishing" half of each cached line. The result is a sudden jump in miss penalties (and hence CPI) once L1's line size outstrips L2's. That misalignment continues to worsen as you further increase L1's block size, so the CPI curve rises sharply beyond the optimal match point.

5.4 Generator 4 Analysis



;

Figure 8: Generator 4 graph

CPI	Generator	LineSize	L1_Hit	L1_Miss	L2_Hit	L2_Miss	L1_WriteBacks	L2_WriteBacks	L1_MissRate	L2_MissRate
1.00576	4	16	349681	256	192	64	0	0	0.0007	0.25
1.00448	4	32	350386	128	64	64	0	0	0.0004	0.5
1.00384	4	64	350133	64	0	64	0	0	0.0002	1
1.00192	4	128	349864	32	0	32	0	0	0.0001	1

Figure 9: Generator 4 Table

After analyzing the results from generator 4, the following has been observed: this pattern walks through a fixed 4 KB buffer sequentially, the entire working set fits in the 16 KB L1 cache after just one pass. So it shows a steady rate of CPI = 1.

5.5 Generator 5 Analysis

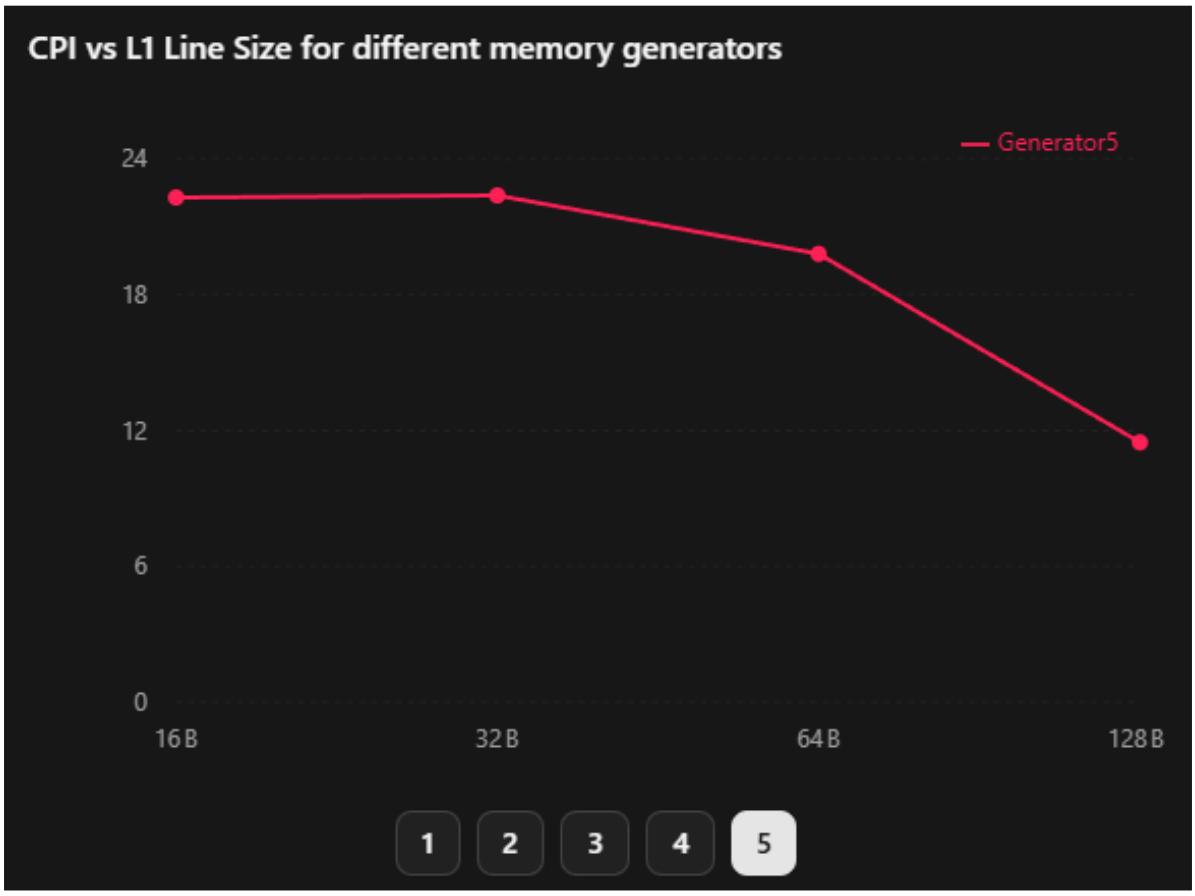


Figure 10: Generator 5 graph

CPI	Generator	LineSize	L1_Hit	L1_Miss	L2_Hit	L2_Miss	L1_WriteBacks	L2_WriteBacks	L1_MissRate	L2_MissRate
22.29504	5	16	0	349739	336308	187576	174145	133548	1	0.358
22.33883	5	32	0	350143	336583	187955	174395	133914	1	0.3583
19.83161	5	64	175206	175207	121157	185394	131344	129928	0.5	0.6048
11.46128	5	128	262543	87515	75282	94211	81978	81116	0.25	0.5558

Figure 11: Generator 5 Table

After analyzing the results from generator 5, the following has been observed: For Generator 5, it implements a stride access every 32 addresses, hence its access pattern is 0,32,64,96,128,..... So if L1 line size is 32 or below all address will miss with a cold start as we can notice in the table, but when the size increases we can see how the miss rate decreases significantly.

6. Conclusion

In conclusion, this project explored how the cache line size affects the memory system performance within a two-level cache hierarchy. By simulating realistic memory access patterns using five distinct generators, we were able to quantify the impact of varying L1 cache line sizes on the effective

cycles per instruction (CPI).

The results demonstrated that there is no universally optimal line size; with the performance being highly dependent on the access patterns. For example, generators with high spatial locality (such as Generator 4) benefited from larger line sizes, exhibiting minimal miss rates and near-ideal CPI values. In contrast, access patterns with poor spatial locality (such as Generator 3) suffered from high miss rates and CPI values regardless of line size, emphasizing the limitations of large cache lines under random or sparse access conditions.

Generator 1 and 2, on the other hand, showed moderate performance gains with increased line size, but their relatively high L1 miss rates suggest that their patterns did not fully utilize the spatial advantages of larger lines. Meanwhile, Generator 5 illustrated the trade-offs between L1 and L2 performances: decreasing L1 miss rates at the cost of increasing L2 traffic.

Ultimately, the simulator affirmed the critical role of cache configuration in system performance while highlighting how architectural decisions, such as cache line sizing, must be tailored to the expected memory access behavior of the workloads. Additionally, alternative replacement policies, prefetching logic, or simulating multi-threaded execution may be incorporated in the future to further enrich the analysis.

7. Future Results

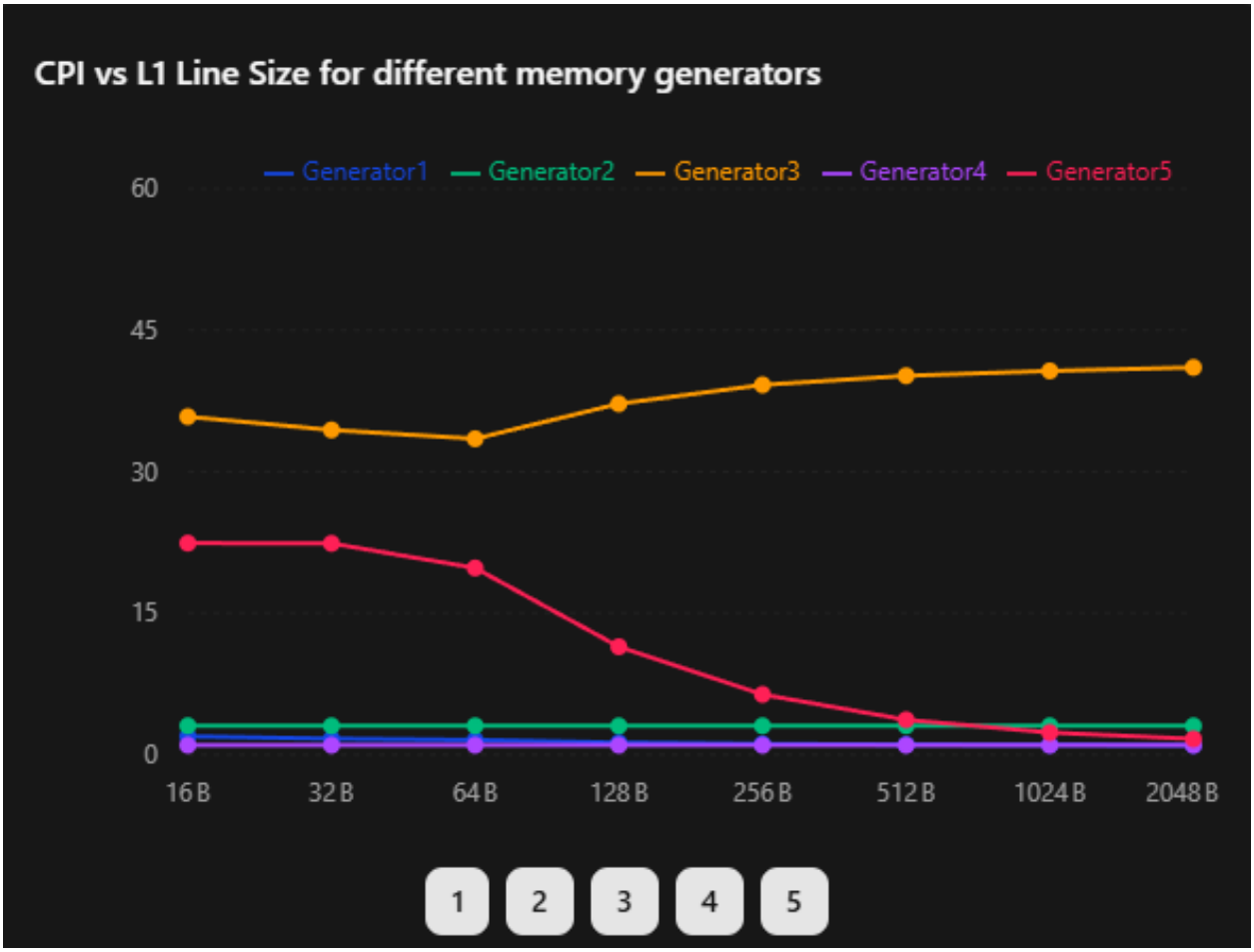


Figure 12: Advanced Simulator