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:

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

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:

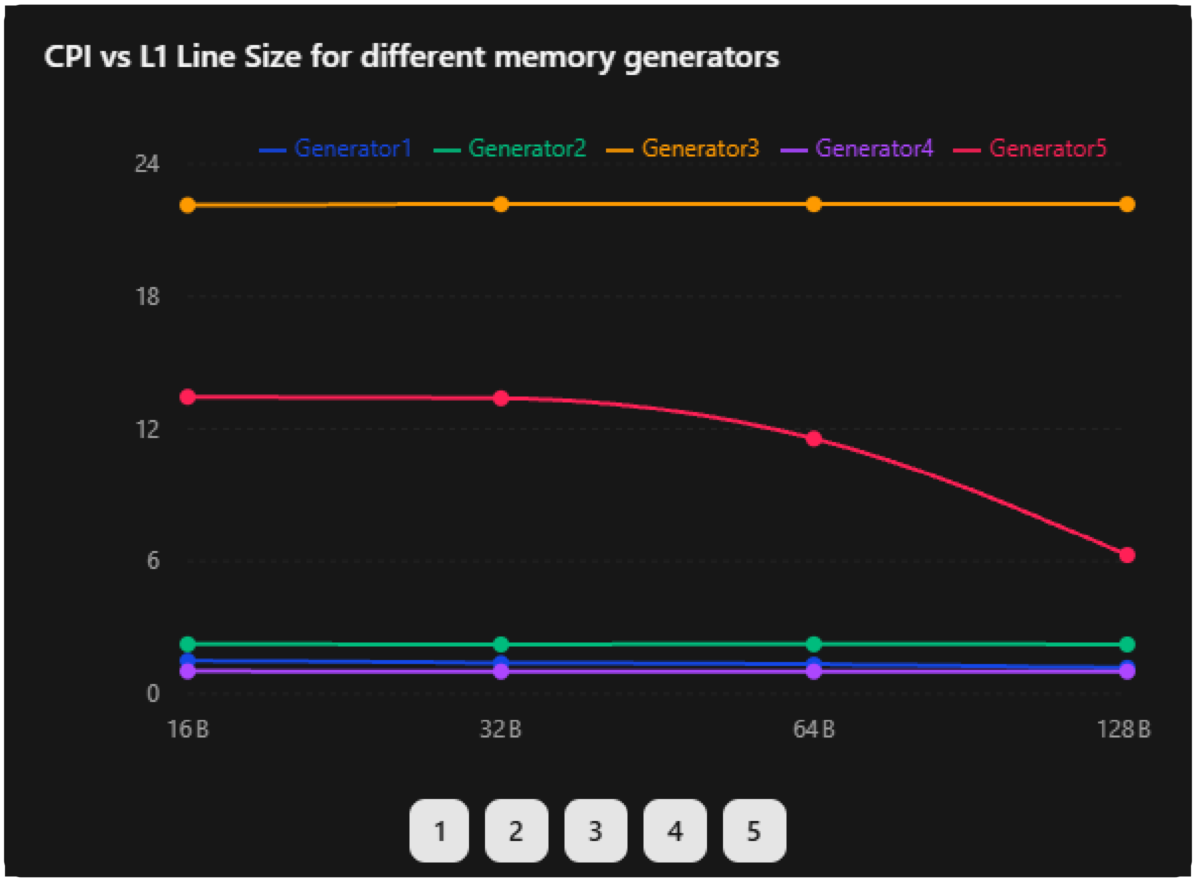


Figure 1: All generators graph

5.1 Generator 1 Analysis

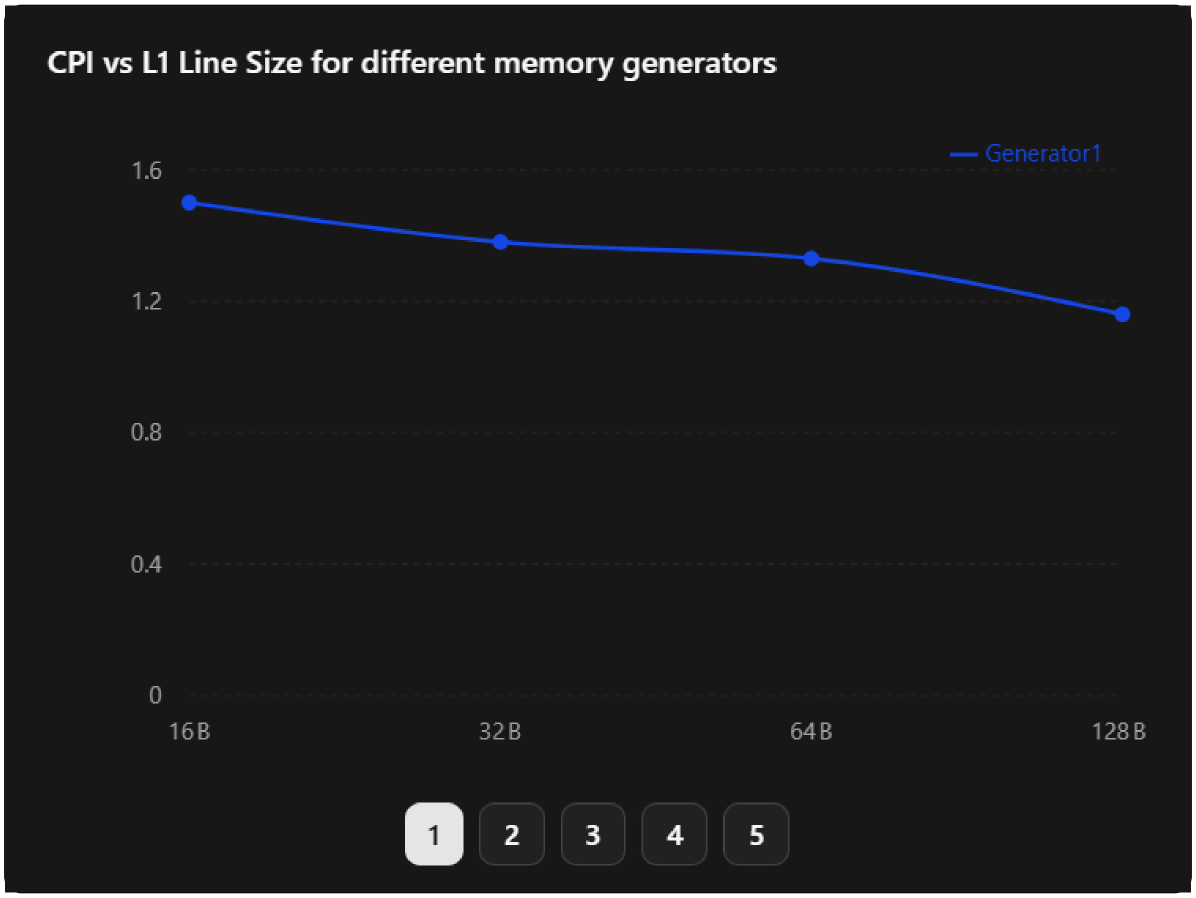
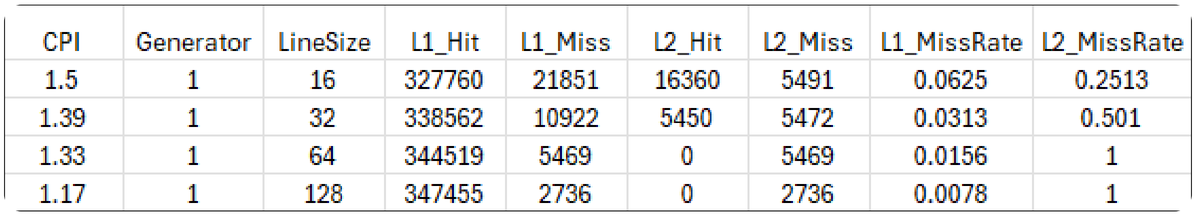


Figure 2: Generator 1 graph



CPI	Generator	LineSize	L1_Hit	L1_Miss	L2_Hit	L2_Miss	L1_MissRate	L2_MissRate
1.5	1	16	327760	21851	16360	5491	0.0625	0.2513
1.39	1	32	338562	10922	5450	5472	0.0313	0.501
1.33	1	64	344519	5469	0	5469	0.0156	1
1.17	1	128	347455	2736	0	2736	0.0078	1

Figure 3: Generator 1 table

After analyzing the results from generator 1, the following has been observed:

- **Point 1** Continue here
- **Point 2** Continue here
- **Point 3** Continue here

5.2 Generator 2 Analysis

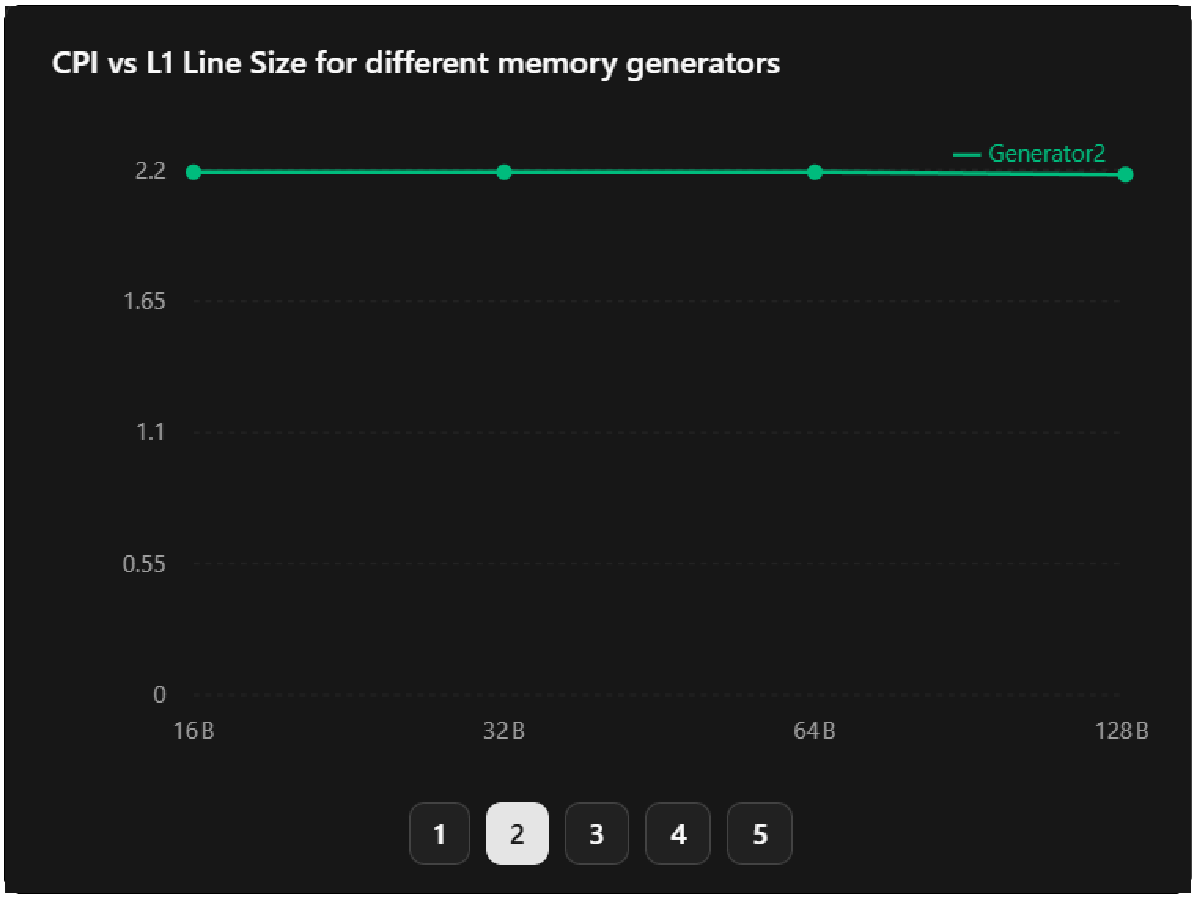
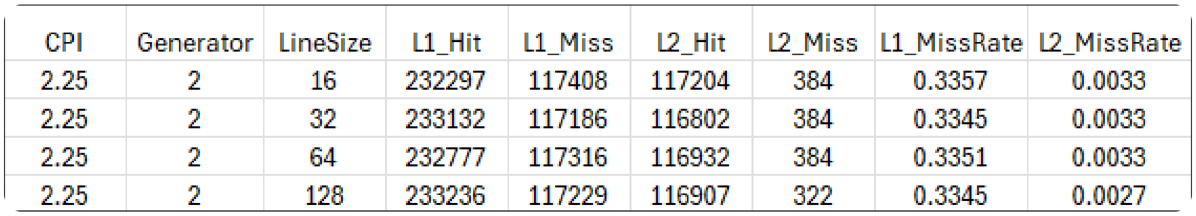


Figure 4: Generator 2 graph



CPI	Generator	LineSize	L1_Hit	L1_Miss	L2_Hit	L2_Miss	L1_MissRate	L2_MissRate
2.25	2	16	232297	117408	117204	384	0.3357	0.0033
2.25	2	32	233132	117186	116802	384	0.3345	0.0033
2.25	2	64	232777	117316	116932	384	0.3351	0.0033
2.25	2	128	233236	117229	116907	322	0.3345	0.0027

Figure 5: Generator 2 table

After analyzing the results from generator 2, the following has been observed:

- **Point 1** Continue here
- **Point 2** Continue here
- **Point 3** Continue here

5.3 Generator 3 Analysis

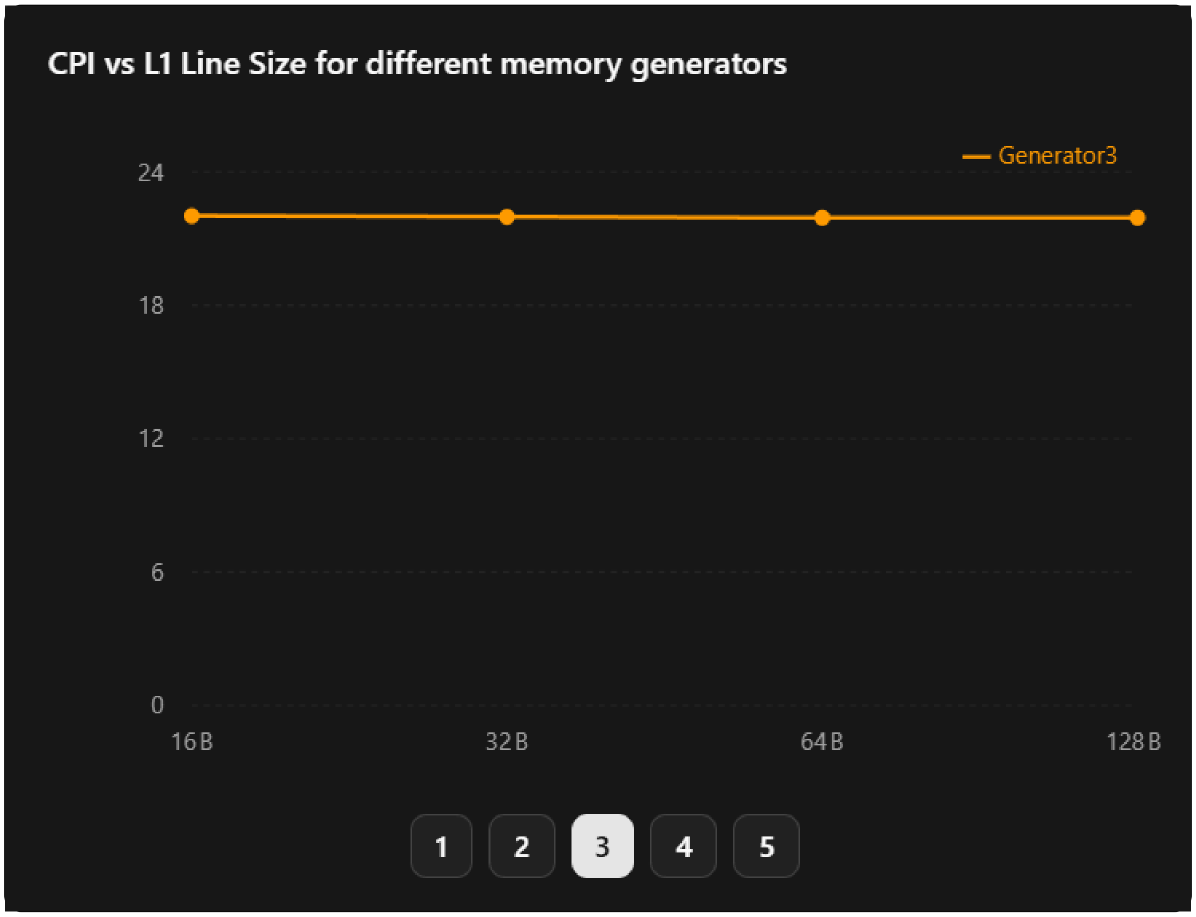
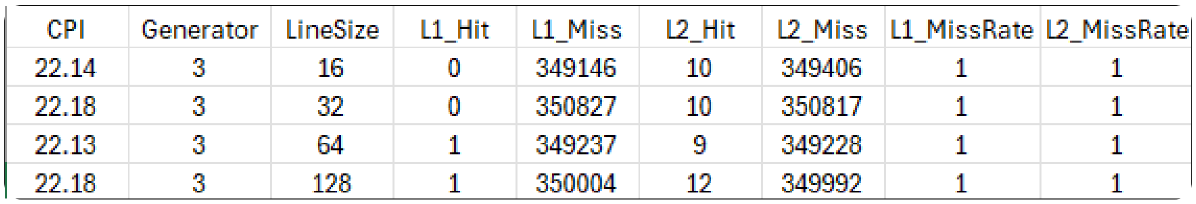


Figure 6: Generator 3 graph



CPI	Generator	LineSize	L1_Hit	L1_Miss	L2_Hit	L2_Miss	L1_MissRate	L2_MissRate
22.14	3	16	0	349146	10	349406	1	1
22.18	3	32	0	350827	10	350817	1	1
22.13	3	64	1	349237	9	349228	1	1
22.18	3	128	1	350004	12	349992	1	1

Figure 7: Generator 3 table

After analyzing the results from generator 3, the following has been observed:

- **Point 1** Continue here
- **Point 2** Continue here
- **Point 3** Continue here

5.4 Generator 4 Analysis

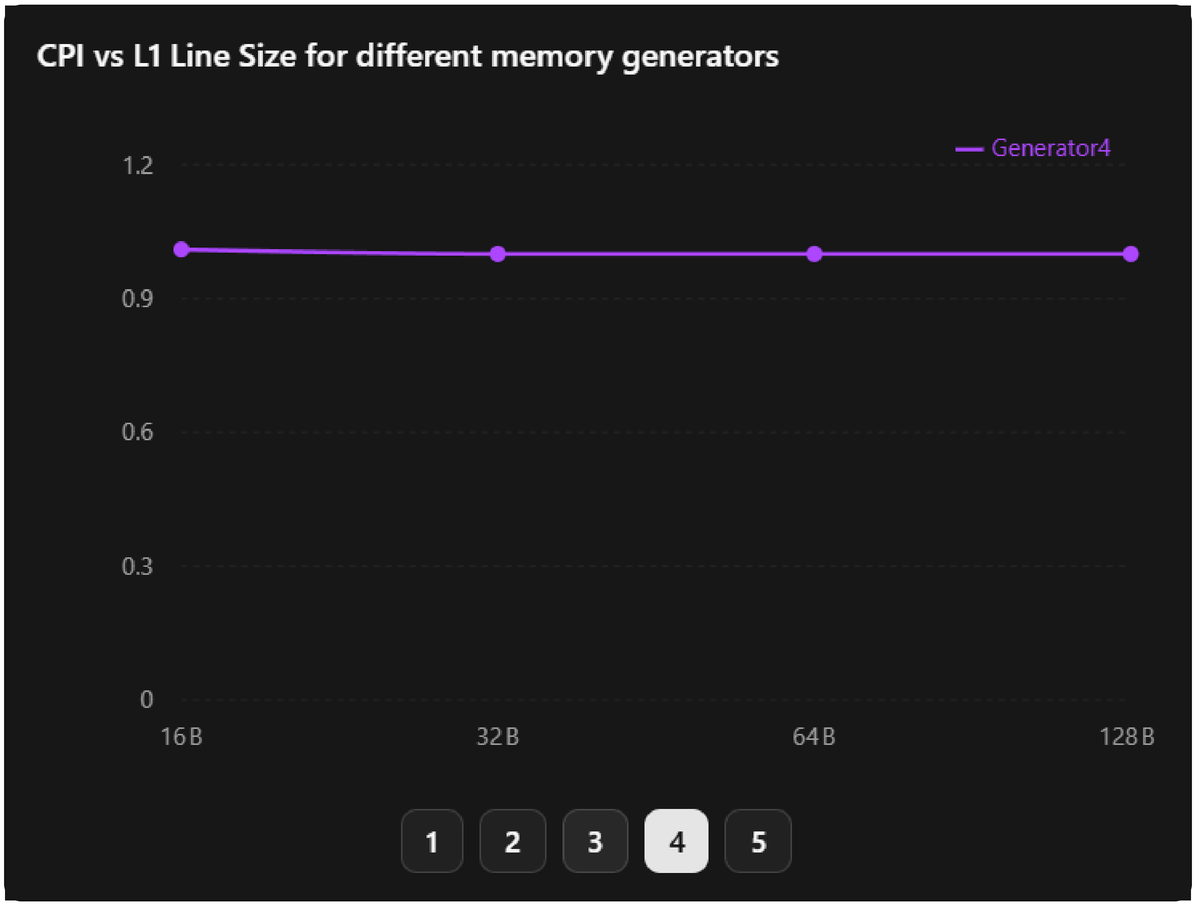
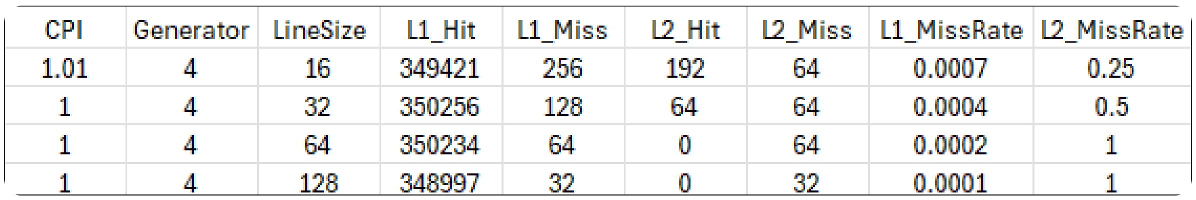


Figure 8: Generator 4 graph



CPI	Generator	LineSize	L1_Hit	L1_Miss	L2_Hit	L2_Miss	L1_MissRate	L2_MissRate
1.01	4	16	349421	256	192	64	0.0007	0.25
1	4	32	350256	128	64	64	0.0004	0.5
1	4	64	350234	64	0	64	0.0002	1
1	4	128	348997	32	0	32	0.0001	1

Figure 9: Generator 4 table

After analyzing the results from generator 4, the following has been observed:

- **Point 1** Continue here
- **Point 2** Continue here
- **Point 3** Continue here

5.5 Generator 5 Analysis

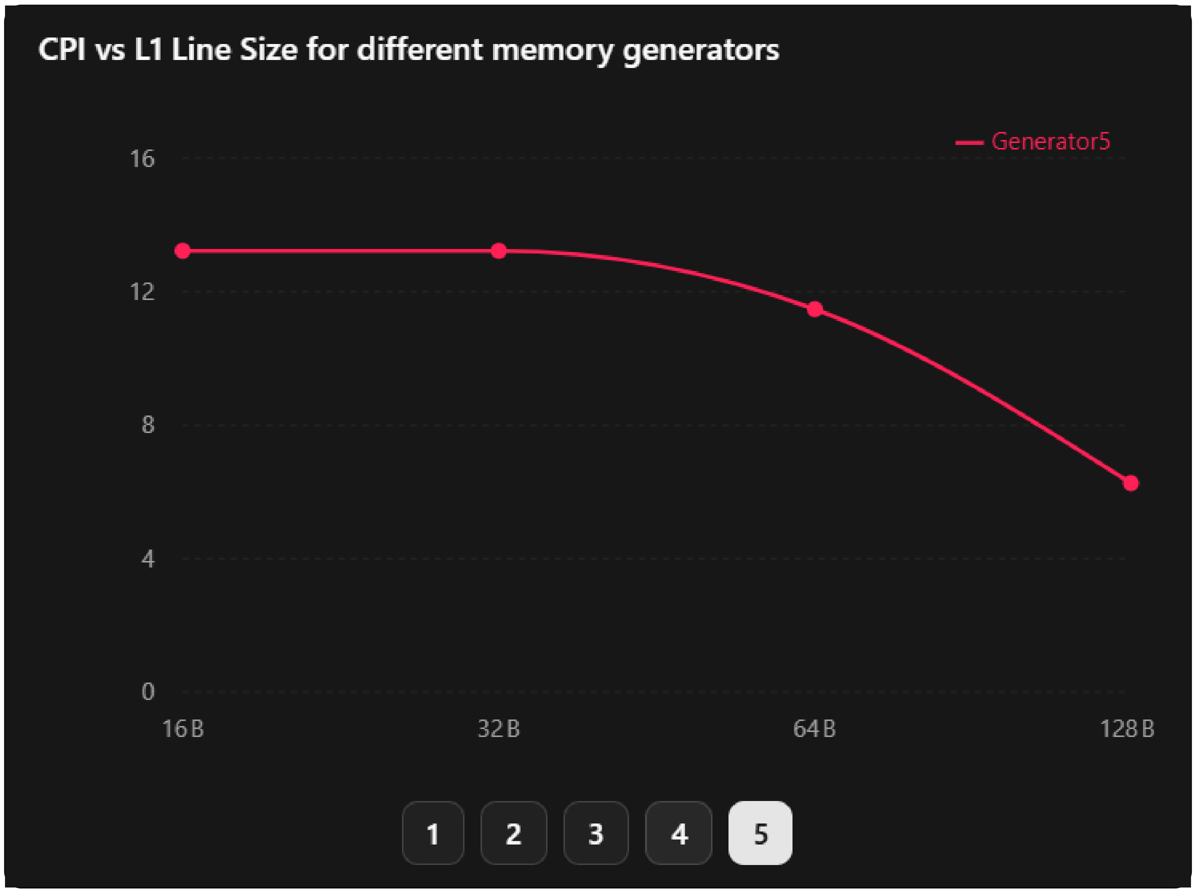
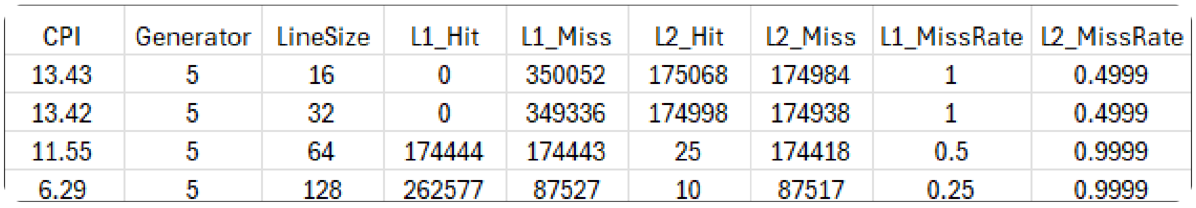


Figure 10: Generator 5 graph



CPI	Generator	LineSize	L1_Hit	L1_Miss	L2_Hit	L2_Miss	L1_MissRate	L2_MissRate
13.43	5	16	0	350052	175068	174984	1	0.4999
13.42	5	32	0	349336	174998	174938	1	0.4999
11.55	5	64	174444	174443	25	174418	0.5	0.9999
6.29	5	128	262577	87527	10	87517	0.25	0.9999

Figure 11: Generator 5 table

After analyzing the results from generator 5, the following has been observed:

- **Point 1** Continue here
- **Point 2** Continue here
- **Point 3** Continue here

6. Challenges

(To be completed).

7. 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.