Cache Design Project Report

Team NINE

September 6, 2024

Team Members

- Dhruv Ramesh Joshi (IMT2023032)
- Arnav Oruganty (IMT2023078)
- Vaishak Prasad Bhat (IMT2023085)

1 Introduction

In this report, we simulate a set-associative cache system and analyze its performance by varying three key parameters: cache size, block size, and associativity. The purpose of this simulation is to study how different configurations impact the hit and miss rates for various memory trace files. Our goal is to identify optimal configurations that maximize cache performance while minimizing cache misses.

Cache memory is a fundamental component in modern CPU architecture, acting as a temporary storage space that improves the speed of memory access by storing frequently used data closer to the processor. The performance of the cache is often gauged by hit rates (successful data retrievals from the cache) and miss rates (failures requiring data retrieval from the slower main memory).

To better understand these trade-offs, we implemented a Python-based cache simulation, which we used to analyze the impact of different cache configurations.

2 Code Design

The simulation was developed in Python and is structured with the following key components:

- **Cache Class:** Models the behavior of the cache, accepting parameters for cache size, block size, and associativity.
- **Block Class:** Represents individual cache blocks containing a tag, a valid bit, and an LRU (Least Recently Used) counter to handle cache replacements.
- **Functions:**
  - `Part-A`: Simulates a 4-way set associative cache with a fixed cache size of 1024KB and a block size of 4 bytes.
  - `Part-B`: Varies cache sizes from 128KB to 4096KB to observe changes in hit/miss rates.
  - `Part-C`: Varies block sizes from 1 byte to 128 bytes while keeping the cache size constant at 1024KB.
  - `Part-D`: Varies associativity from 1-way to 64-way with a fixed cache size of 1024KB.

3 Methodology

In this study, we used memory trace files representing various workloads to evaluate the performance of different cache configurations. The parameters varied include:

- Cache Size: The total amount of memory available in the cache (from 128KB to 4096KB).
- Block Size: The amount of data retrieved in a single access (from 1 byte to 128 bytes).
- Associativity: The number of cache lines available in each set (from 1-way to 64-way).

To simulate the performance, the Python implementation measured hit and miss rates for different trace files, providing insights into how the cache parameters affect the overall performance.

4 Results and Observations

4.1 Part A: Fixed Cache Size (1024KB) and Block Size (4 Bytes)

Cache Lines Calculation

The formula to calculate the number of cache lines is given by:

$$\text{Cache Lines} = \frac{\text{Cache Size}}{\text{Block Size} \times \text{Number of Ways}}$$

Given:

- Cache Size $= 1024$ kB $= 1024 \times 1024$ Bytes
- Block Size $= 4$ Bytes
- Number of Ways $= 4$

Calculation

First, convert the cache size to bytes:

$$\text{Cache Size} = 1024 \times 1024 = 1048576 \text{ Bytes}$$

Next, apply the formula:

$$\text{Cache Lines} = \frac{1048576}{4 \times 4}$$

$$\text{Cache Lines} = \frac{1048576}{16}$$

$$\text{Cache Lines} = 65536$$

Therefore, the number of cache lines is **65536.**

```
○ arnavoruganty@Arnavs-MacBook-Air CA % python3 -u "/Users/arnavoruganty/Desktop/CA/cache.py"
---------------Cache Simulator---------------
Enter the integer to view the corresponding answer of the question:
    1) Question A
    2) Question B
    3) Question C
    4) Question D
    -1) Exit
Enter the integer: 1


Answer to Question A:
These are the hit rate, miss rate and hit/miss rate for the given Cache.

+-------+------------+-------------+------------------+
|       |   Hit Rate |   Miss Rate |   Hit/Miss Ratio |
+=======+============+=============+==================+
| gcc   |    93.8356 |     6.16445 |       15.2221    |
+-------+------------+-------------+------------------+
| gzip  |    66.7055 |    33.2945  |        2.0035    |
+-------+------------+-------------+------------------+
| mcf   |     1.03241 |    98.9676  |        0.0104318 |
+-------+------------+-------------+------------------+
| swim  |    92.6225 |     7.37748 |       12.5548    |
+-------+------------+-------------+------------------+
| twolf |    98.7615 |     1.23855 |       79.7398    |
+-------+------------+-------------+------------------+
```

Figure 1: Output for Part A

4.2 Part B: Varying Cache Size (128KB to 4096KB)

By varying the cache size, we observed that the miss rates generally decrease as the cache size increases. However, beyond a cache size of 1024KB, the improvement in hit rate becomes marginal, indicating diminishing returns.

Cache Size (kB)	gcc	gzip	mcf	swim	twolf
128	6.19%	33.29%	98.97%	7.38%	1.24%
256	6.17%	33.29%	98.97%	7.38%	1.24%
512	6.16%	33.29%	98.97%	7.38%	1.24%
1024	6.16%	33.29%	98.97%	7.38%	1.24%
2048	6.16%	33.29%	98.97%	7.38%	1.24%
4096	6.16%	33.29%	98.95%	7.38%	1.24%

Table 1: Miss Rates for Different Cache Sizes

Cache Size (kB)	gcc	gzip	mcf	swim	twolf
128	93.81%	66.71%	1.03%	92.62%	98.76%
256	93.83%	66.71%	1.03%	92.62%	98.76%
512	93.84%	66.71%	1.03%	92.62%	98.76%
1024	93.84%	66.71%	1.03%	92.62%	98.76%
2048	93.84%	66.71%	1.03%	92.62%	98.76%
4096	93.84%	66.71%	1.05%	92.62%	98.76%

Table 2: Hit Rates for Different Cache Sizes

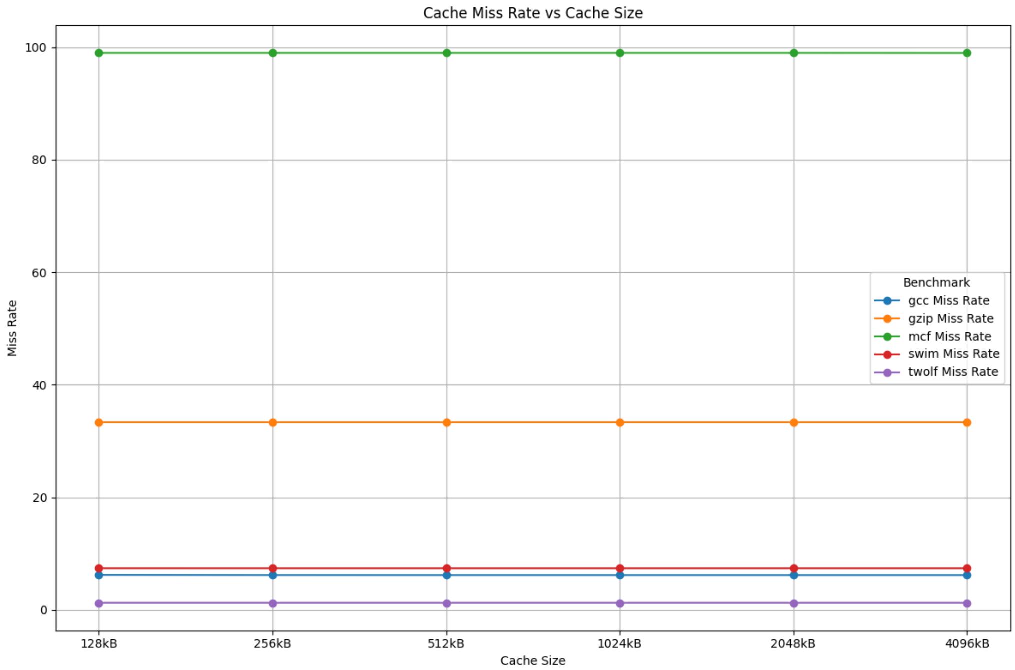

Figure 2: Miss Rate vs Cache Size

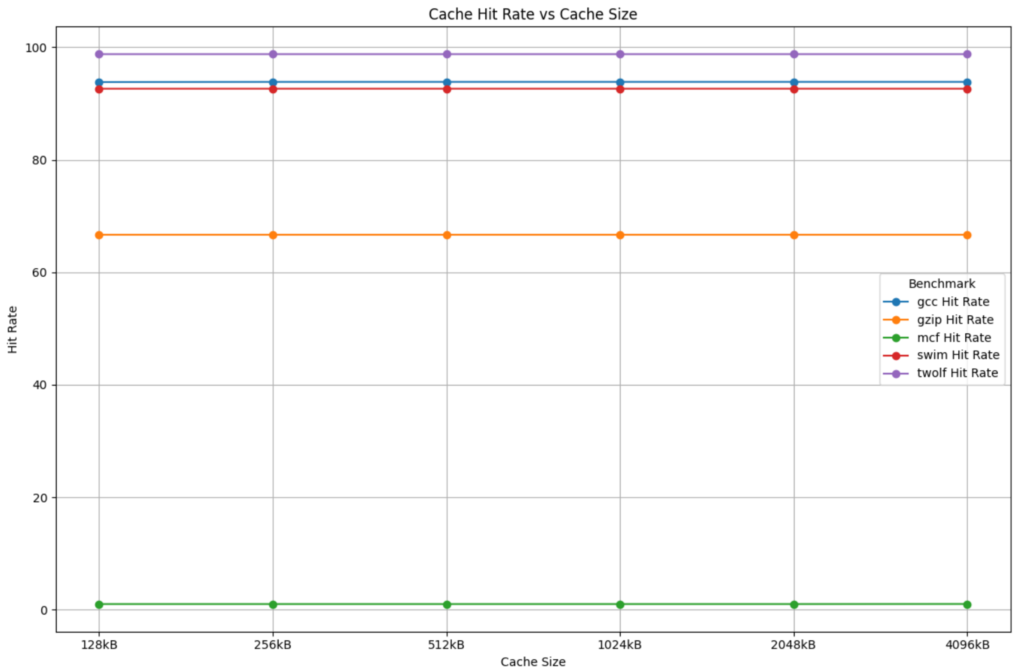

Figure 3: Hit Rate vs Cache Size

Answer to Question B:
These are the hit rate, miss rate and hit/miss rate for the given Cache when the cache size is varied.

Hit Rate:

	128kB	256kB	512kB	1024kB	2048kB	4096kB
gcc	93.8016	93.8311	93.8354	93.8356	93.8356	93.8356
gzip	66.7055	66.7055	66.7055	66.7055	66.7055	66.7055
mcf	1.03241	1.03241	1.03241	1.03241	1.03241	1.04547
swim	92.6199	92.6225	92.6225	92.6225	92.6225	92.6225
twolf	98.7612	98.7615	98.7615	98.7615	98.7615	98.7615

Miss Rate:

	128kB	256kB	512kB	1024kB	2048kB	4096kB
gcc	6.19838	6.16891	6.16464	6.16445	6.16445	6.16445
gzip	33.2945	33.2945	33.2945	33.2945	33.2945	33.2945
mcf	98.9676	98.9676	98.9676	98.9676	98.9676	98.9545
swim	7.38012	7.37748	7.37748	7.37748	7.37748	7.37748
twolf	1.23875	1.23855	1.23855	1.23855	1.23855	1.23855

Hit/Miss Ratio:

	128kB	256kB	512kB	1024kB	2048kB	4096kB
gcc	15.1332	15.2103	15.2215	15.2221	15.2221	15.2221
gzip	2.0035	2.0035	2.0035	2.0035	2.0035	2.0035
mcf	0.0104318	0.0104318	0.0104318	0.0104318	0.0104318	0.0105652
swim	12.5499	12.5548	12.5548	12.5548	12.5548	12.5548
twolf	79.7263	79.7398	79.7398	79.7398	79.7398	79.7398

Figure 4: Output for Part B

Cache Size Variation and Its Impact on Hit Rates

Cache size ranges from 128KiB to 4096KiB
Block size: 4 Bytes
Associativity: 4-Way

Observations:

- **gcc.trace**: There is a slight increase in hit rate when the cache size is increased from 128 to 1024 KiB. Beyond 1024 KiB, there is no further increase in the number of hits. This is likely because, with a larger cache size, the number of sets increases, and at 1024 KiB, all addresses in the trace can fit within the cache without evicting one another, leading to no additional increase beyond 1024 KiB.
- **gzip.trace**: The cache size does not affect the hit rates, likely because, similar to gcc.trace, at 1024 KiB and beyond, all addresses can be present simultaneously without evicting one another.
- **mcf.trace**: There is no increase in hit rate until the cache size reaches 4096 KiB. At 4096 KiB, we observe a slight increase in hit rate.
- **swim.trace**: The hit rate increases when the cache size is increased from 128 to 256 KiB, but there is no further improvement with additional cache expansion. This is likely for the same reason that gcc.trace and gzip.trace hit rates saturate.
- **twolf.trace**: Similar to swim.trace, there is a slight improvement when increasing the cache size from 128 to 256 KiB, but no further increase beyond that.

4.3 Part C: Varying Block Size (1 Byte to 128 Bytes)

When varying the block size, we noticed that larger block sizes initially reduce the miss rate. However, beyond a certain point (32 bytes), further increases in block size degrade performance as the number of cache lines decreases.

Block Size (Bytes)	gcc	gzip	mcf	swim	twolf
1	6.80%	33.30%	98.98%	7.46%	1.52%
2	6.38%	33.30%	98.97%	7.40%	1.34%
4	6.16%	33.29%	98.97%	7.38%	1.24%
8	4.07%	33.29%	98.96%	6.54%	1.14%
16	2.17%	33.21%	49.50%	3.77%	0.61%
32	1.17%	33.17%	24.76%	2.11%	0.34%
64	0.65%	33.15%	12.39%	1.14%	0.20%
128	0.38%	33.14%	6.20%	0.60%	0.12%

Table 3: Miss Rates for Different Block Sizes

Block Size (Bytes)	gcc	gzip	mcf	swim	twolf
1	93.20%	66.70%	1.02%	92.54%	98.48%
2	93.62%	66.70%	1.03%	92.60%	98.66%
4	93.84%	66.71%	1.03%	92.62%	98.76%
8	95.93%	66.71%	1.04%	93.46%	98.86%
16	97.83%	66.79%	50.50%	96.23%	99.39%
32	98.83%	66.83%	75.24%	97.89%	99.66%
64	99.35%	66.85%	87.61%	98.86%	99.80%
128	99.62%	66.86%	93.80%	99.40%	99.88%

Table 4: Hit Rates for Different Block Sizes

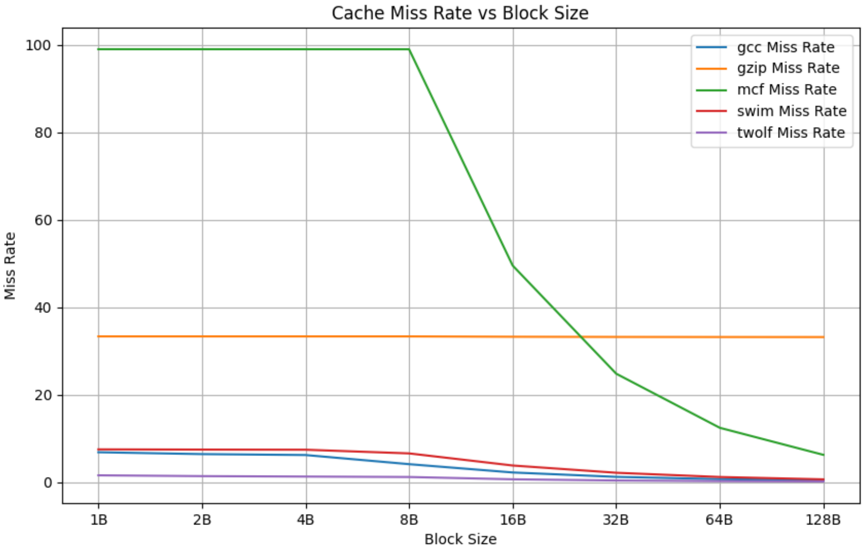

Figure 5: Miss Rate vs Block Size

Impact of Varying Block Sizes on Hit Rates

Cache size: 1024 KiB
Block size: 1B to 128B
Associativity: 4-Way

Observations

- **gcc.trace**: Hit rates increase continuously as block sizes are increased, due to the effective use of spatial locality. However, the benefits diminish with further increases in block size.
- **gzip.trace**: There are steady and minimal increments in hit rates, presumably for similar reasons as gcc.trace.
- **mcf.trace**: Hit rates show a slight increase until the block size reaches 8B. Beyond this, there is a substantial increase in hit rates with each increment, because the addresses accessed by this trace file exploit spatial locality effectively after a certain point.
- **swim.trace**: Performance is similar to gcc.trace, with hit rates increasing as block sizes are increased.
- **twolf.trace**: Similar to gzip.trace, there are steady and minimal increments in hit rates.

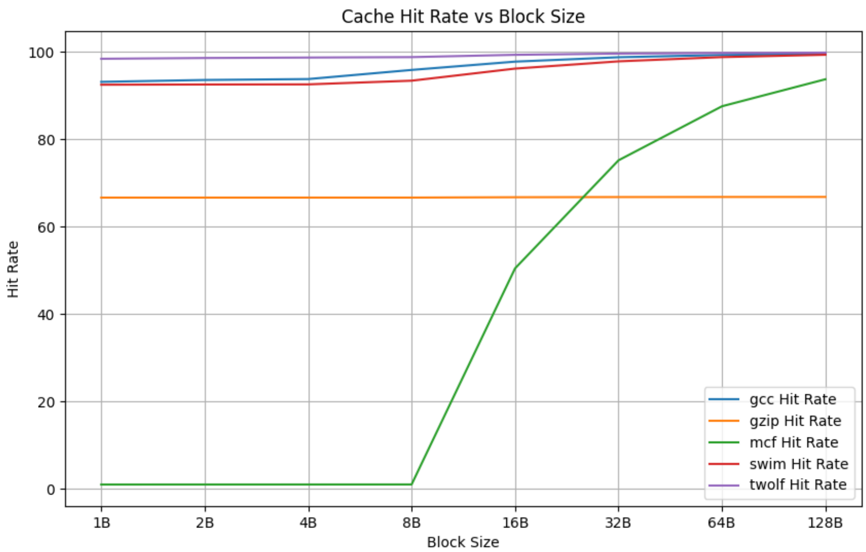

Figure 6: Hit Rate vs Block Size

Observation: The hit rate improves up to a block size of 32 bytes, after which it starts to decline as the cache begins to lose capacity for unique blocks.

Additional Observation: Applications that exhibit strong spatial locality benefit from larger block sizes, but for those with random access patterns, smaller block sizes may yield better performance.

Answer to Question C:
These are the hit rate, miss rate, and hit/miss rate for the given Cache when the block size is varied.

Hit Rate:

	1B	2B	4B	8B	16B	32B	64B	128B
gcc	93.1989	93.6248	93.8356	95.9266	97.825	98.8289	99.3459	99.6209
gzip	66.7039	66.7041	66.7055	66.7072	66.7856	66.8253	66.8461	66.8565
mcf	1.02457	1.0287	1.03241	1.03832	50.503	75.2378	87.608	93.7955
swim	92.5444	92.5935	92.6225	93.4642	96.2324	97.8905	98.8611	99.3977
twolf	98.4769	98.6608	98.7615	98.8598	99.388	99.6599	99.8024	99.8809

Miss Rate:

	1B	2B	4B	8B	16B	32B	64B	128B
gcc	6.80108	6.37523	6.16445	4.07343	2.17498	1.17107	0.654084	0.379109
gzip	33.2961	33.2959	33.2945	33.2928	33.2144	33.1747	33.1539	33.1435
mcf	98.9754	98.9713	98.9676	98.9617	49.497	24.7622	12.392	6.2045
swim	7.45565	7.4065	7.37748	6.53577	3.76757	2.10955	1.13888	0.602257
twolf	1.52312	1.3392	1.23855	1.14017	0.612024	0.340083	0.197588	0.119091

Hit/Miss Rate:

	1B	2B	4B	8B	16B	32B	64B	128B
gcc	13.7036	14.6857	15.2221	23.5493	44.9774	84.3921	151.886	262.776
gzip	2.00335	2.00337	2.0035	2.00365	2.01074	2.01434	2.01623	2.01718
mcf	0.0103518	0.0103939	0.0104318	0.0104922	1.02032	3.03842	7.06975	15.1173
swim	12.4127	12.5016	12.5548	14.3004	25.5423	46.4035	86.8057	165.042
twolf	64.6546	73.6712	79.7398	86.7064	162.392	293.046	505.105	838.694

Figure 7: Output for Part C

4.4 Part D: Varying Associativity (1-Way to 64-Way)

Increasing the associativity generally improves hit rates as it reduces conflicts between cache lines. However, beyond 16-way associativity, the benefits diminish, indicating that higher associativity does not always lead to better performance for all workloads.

Associativity	gcc	gzip	mcf	swim	twol
1-way	6.17%	33.29%	98.97%	7.38%	1.25%
2-way	6.17%	33.29%	98.97%	7.38%	1.24%
4-way	6.16%	33.29%	98.97%	7.38%	1.24%
8-way	6.16%	33.29%	98.97%	7.38%	1.24%
16-way	6.16%	33.29%	98.97%	7.38%	1.24%
32-way	6.16%	33.29%	98.97%	7.38%	1.24%
64-way	6.16%	33.29%	98.97%	7.38%	1.24%

Table 5: Miss Rates for Different Associativities

Associativity	gcc	gzip	mcf	swim	twol
1-way	93.83%	66.71%	1.03%	92.62%	98.75%
2-way	93.83%	66.71%	1.03%	92.62%	98.76%
4-way	93.84%	66.71%	1.03%	92.62%	98.76%
8-way	93.84%	66.71%	1.03%	92.62%	98.76%
16-way	93.84%	66.71%	1.03%	92.62%	98.76%
32-way	93.84%	66.71%	1.03%	92.62%	98.76%
64-way	93.84%	66.71%	1.03%	92.62%	98.76%

Table 6: Hit Rates for Different Associativities

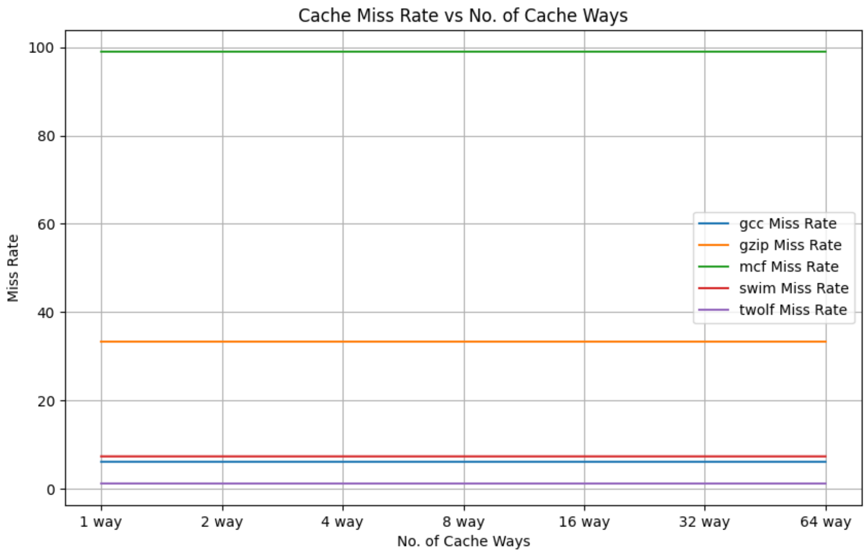

Figure 8: Miss Rate vs Associativity

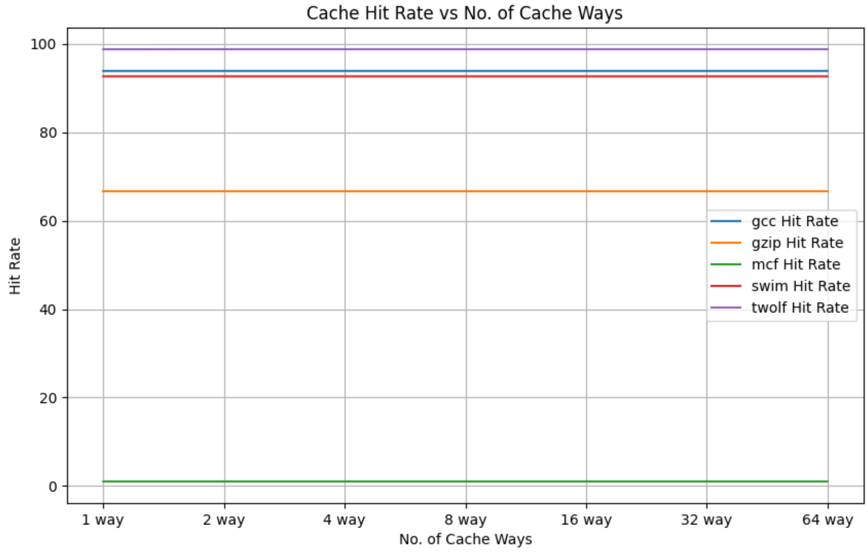

Figure 9: Hit Rate vs Associativity

Observation: Increasing number of ways from 1 to 2 or 4 gives a very slight increase in hit rate but anything beyond that seems to generally give negligible benefits.

Answer to Question D:
These are the hit rate, miss rate and hit/miss rate for the given Cache when the no. of cache ways is varied.

Hit Rate:

	1 way	2 way	4 way	8 way	16 way	32 way	64 way
gcc	93.8305	93.8348	93.8356	93.8356	93.8357	93.8359	93.8359
gzip	66.7055	66.7055	66.7055	66.7055	66.7055	66.7055	66.7055
mcf	1.032	1.03227	1.03241	1.03241	1.03241	1.03241	1.03241
swim	92.6205	92.6225	92.6225	92.6225	92.6225	92.6225	92.6225
twolf	98.7463	98.7608	98.7615	98.7615	98.7615	98.7615	98.7615

Miss Rate:

	1 way	2 way	4 way	8 way	16 way	32 way	64 way
gcc	6.16949	6.16522	6.16445	6.16445	6.16425	6.16406	6.16406
gzip	33.2945	33.2945	33.2945	33.2945	33.2945	33.2945	33.2945
mcf	98.968	98.9677	98.9676	98.9676	98.9676	98.9676	98.9676
swim	7.37946	7.37748	7.37748	7.37748	7.37748	7.37748	7.37748
twolf	1.25367	1.23917	1.23855	1.23855	1.23855	1.23855	1.23855

Hit/Miss Rate:

	1 way	2 way	4 way	8 way	16 way	32 way	64 way
gcc	15.2088	15.22	15.2221	15.2221	15.2226	15.2231	15.2231
gzip	2.0035	2.0035	2.0035	2.0035	2.0035	2.0035	2.0035
mcf	0.0104276	0.0104304	0.0104318	0.0104318	0.0104318	0.0104318	0.0104318
swim	12.5511	12.5548	12.5548	12.5548	12.5548	12.5548	12.5548
twolf	78.7661	79.6993	79.7398	79.7398	79.7398	79.7398	79.7398

Figure 10: Output for Part D

5 Conclusion

This project demonstrates how cache performance is affected by key parameters such as cache size, block size, and associativity. Larger cache sizes and higher associativity generally improve hit rates, but with diminishing returns. Varying the block size shows that a balance must be struck between exploiting spatial locality and maintaining enough cache lines to store unique blocks. Overall, our experiments highlight the importance of tuning cache parameters to match workload-specific characteristics for optimal performance.