COL216 Cache Simulation : Report

Abhijeet Choudhary

2022CS11104

Introduction

This report examines the effectiveness of various cache configurations through simulations to determine their impact on hit rates, miss penalties, cache size, and overall system performance. Utilizing a custom-built cache simulator, we explore different cache structures and policies to identify optimal settings for maximizing computational efficiency.

Analysis:

1. Varying the Number of Sets

Changing the number of sets in a cache directly impacts its structure and performance, particularly in terms of hit rate, miss penalty, and the complexity of the cache.

- **Impact on Hit Rate:**

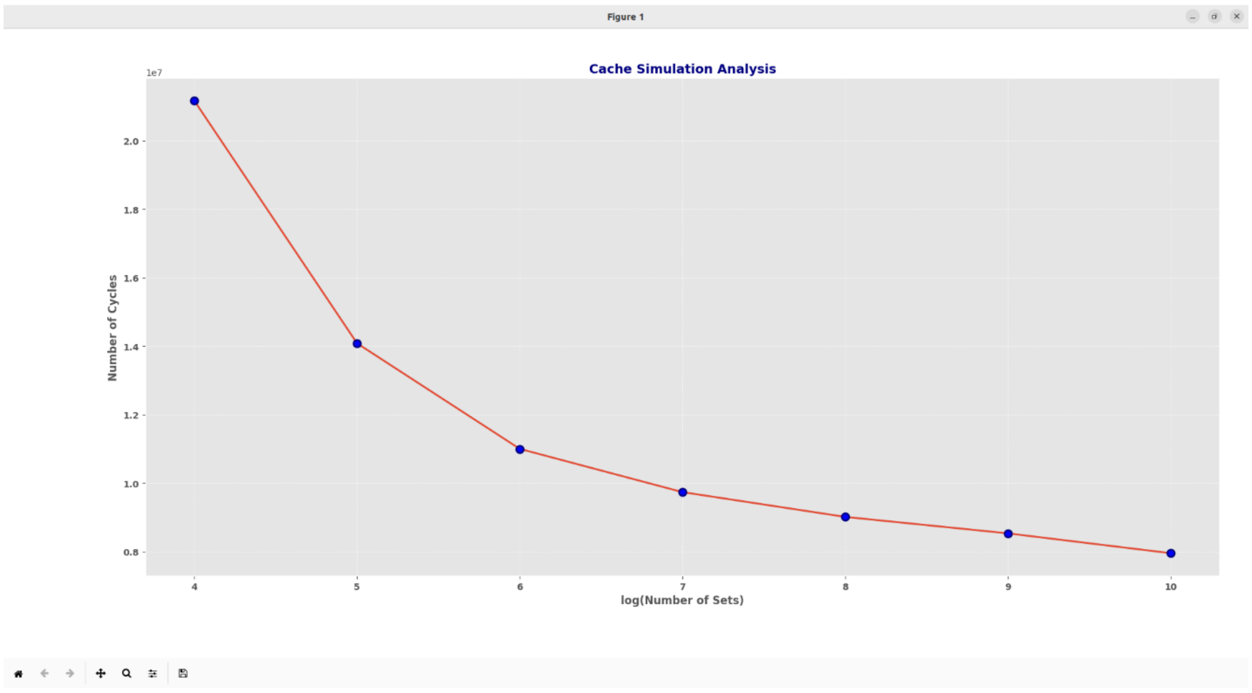
Increasing the Number of Sets: Typically reduces conflict misses in set-associative and direct-mapped caches, as each set can hold more unique data blocks. This generally improves the hit rate, especially in workloads with diverse memory access patterns that might otherwise lead to cache line collisions.

- **Impact on Miss Penalty:**

Increased Sets: Reduces miss penalties by lowering conflict misses, which means that the cache more effectively provides data, reducing the need to access slower main memory.

- **Impact on Cache Complexity and Size:**

More Sets: Increases the complexity of the cache controller, as it needs to manage more sets, which can involve more complex indexing mechanisms and potentially larger tag fields in the cache metadata.



We can see that increasing the number of sets increases the hit rate. Now, commenting on the number of cycles is not trivial, as when the number of sets increase, the hit time also increases, hence we cannot comment on the total number of cycles taken.

2. Varying Block Size

The size of the blocks stored in each cache line significantly influences cache performance. Larger blocks can capitalize on spatial locality, while smaller blocks may reduce the amount of data loaded that is not used.

- **Impact on Hit Rate:**

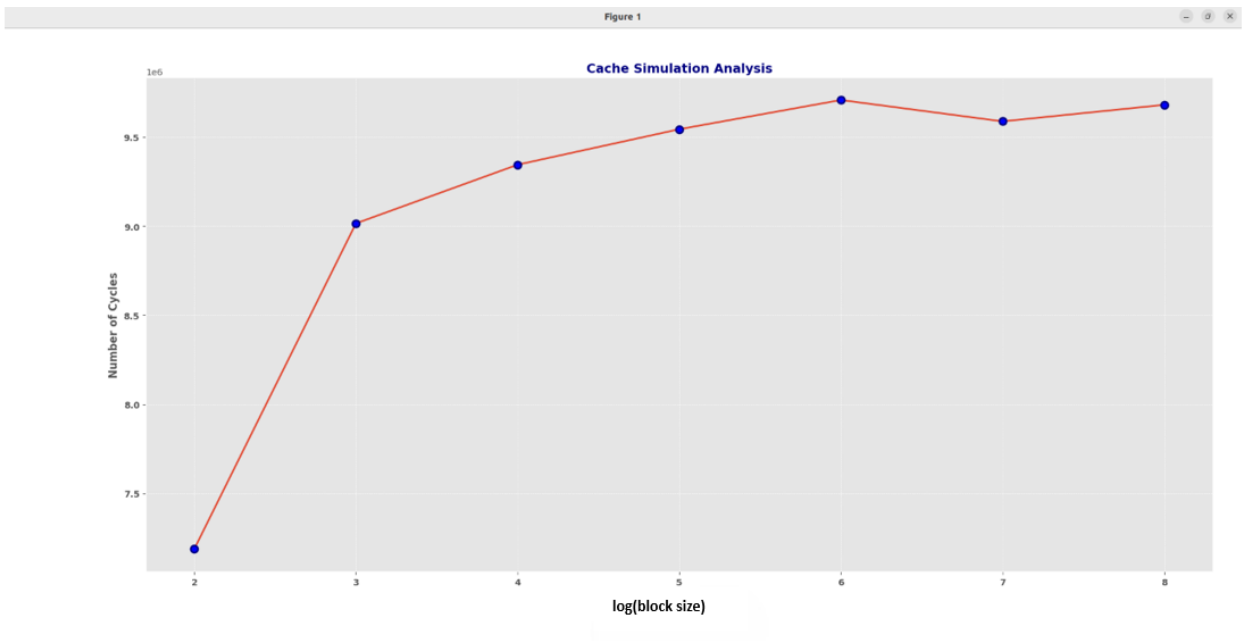
Larger Block Sizes: Improve the hit rate when there is significant spatial locality in the access patterns, as more data is brought into the cache with each memory fetch. However, if the spatial locality is low, larger blocks might lead to loading data that is never accessed, which can reduce the effective cache capacity for useful data.

- Larger **Impact on Miss Penalty:**

Blocks: Increase the miss penalty, as each miss incurs the cost of loading a larger amount of data from main memory, which takes more time.

- **Impact on Cache Complexity and Size:**

Larger Block Sizes: Can simplify cache design in terms of fewer blocks to manage and potentially simpler indexing. However, it increases the memory bandwidth requirements to fill a block.



3. *Cache Configuration*

Direct-Mapped vs. Set-Associative vs. Fully Associative

- **Hit Rate:**

Direct-Mapped: Offers the fastest access time among cache types but may suffer from higher miss rates due to conflict misses.

Set-Associative: Typically has a higher hit rate than direct-mapped because multiple blocks can reside at the same index, reducing conflict misses.

Fully Associative: Typically has the highest hit rate since any block can be placed in any line in the cache, virtually eliminating conflict misses.

- **Miss Penalty:**

Direct-Mapped: Simpler hardware means faster access times but potentially higher miss penalties if conflict misses are frequent.

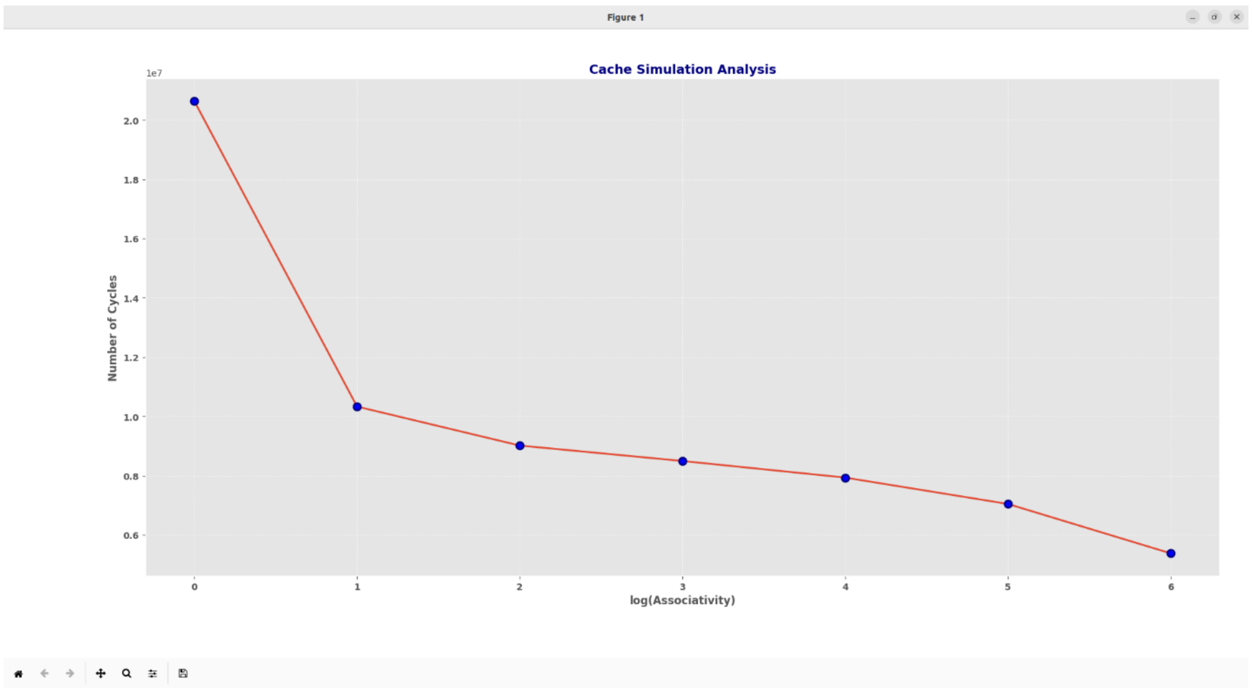
Set-Associative and Fully Associative: More complex hardware can slow down access times slightly due to the need to search through multiple blocks, potentially increasing the miss penalty.

- **Cache Size and Complexity:**

Direct-Mapped: Least complex, requires less hardware for indexing and tag comparison.

Set-Associative: More complex than direct-mapped, needing additional logic to handle multiple blocks per set.

Fully Associative: Most complex and costly in terms of hardware because it requires a search through the entire cache for every access.



4. *Write Policy*

Write-Back vs. Write-Through

- **Hit Rate:**

Not directly affected by the write policy, though write-back may indirectly improve hit rates by reducing the frequency of main memory accesses, which can keep the cache filled with useful data.

- **Miss Penalty:**

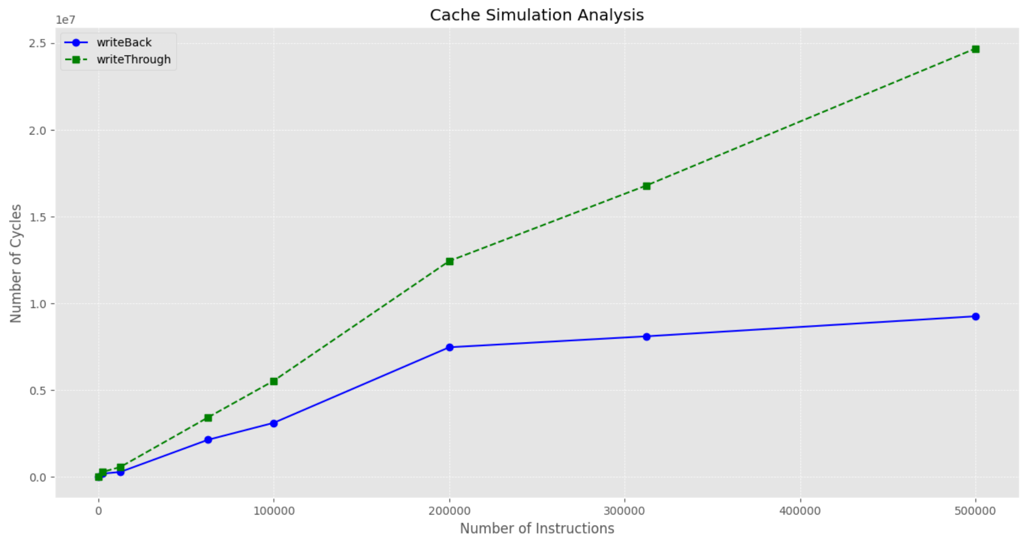
Write-Back: Miss penalty can be higher because dirty blocks require writing back to main memory on eviction.

Write-Through: Generally, has a lower miss penalty for writes because data is immediately written to main memory, ensuring consistency but increasing memory traffic.

- **Cache Size and Complexity:**

Write-Back: Requires mechanisms to track which blocks are dirty.

Write-Through: Simpler as it doesn't need dirty bits, but requires a high-bandwidth connection to main memory to handle frequent writes.



5. *Allocation Policy*

Write-Allocate vs. No-Write-Allocate

- **Hit Rate:**

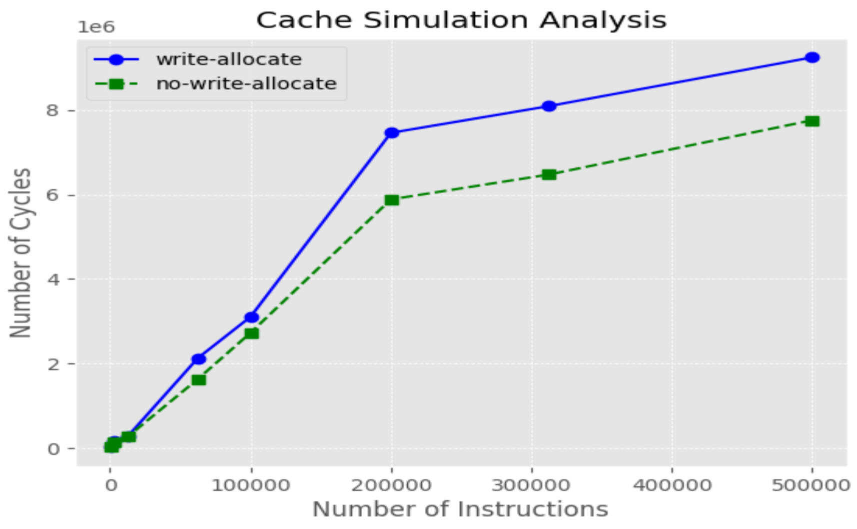
Write-Allocate: Potentially increases the hit rate for subsequent accesses to the newly written data by ensuring it resides in the cache.

No-Write-Allocate: May suffer from lower hit rates for write accesses since data is not cached on a miss, potentially leading to another miss if the data is accessed again soon.

- **Miss Penalty:**

Write-Allocate: Can increase the miss penalty because bringing a block into the cache on a write miss involves reading from main memory.

No-Write-Allocate: Reduces miss penalty for writes as it writes directly to main memory, bypassing the cache.



6. Replacement Policy

LRU vs. FIFO

- **Hit Rate:**

LRU (Least Recently Used): Generally, offers a better hit rate in environments with strong locality of reference, as it keeps recently used items in the cache.

FIFO (First In, First Out): Can perform worse in terms of hit rate because it might evict blocks that are frequently accessed but were loaded early into the cache.

- **Miss Penalty:**

Both LRU and FIFO have similar miss penalties, but the efficiency of LRU in keeping relevant data can reduce the frequency of cache misses.

- **Cache Size and Complexity:**

LRU: More complex to implement because it must constantly update the usage status of cache blocks.

FIFO: Simpler to implement as it just evicts blocks in the order they were inserted, without considering their usage frequency.

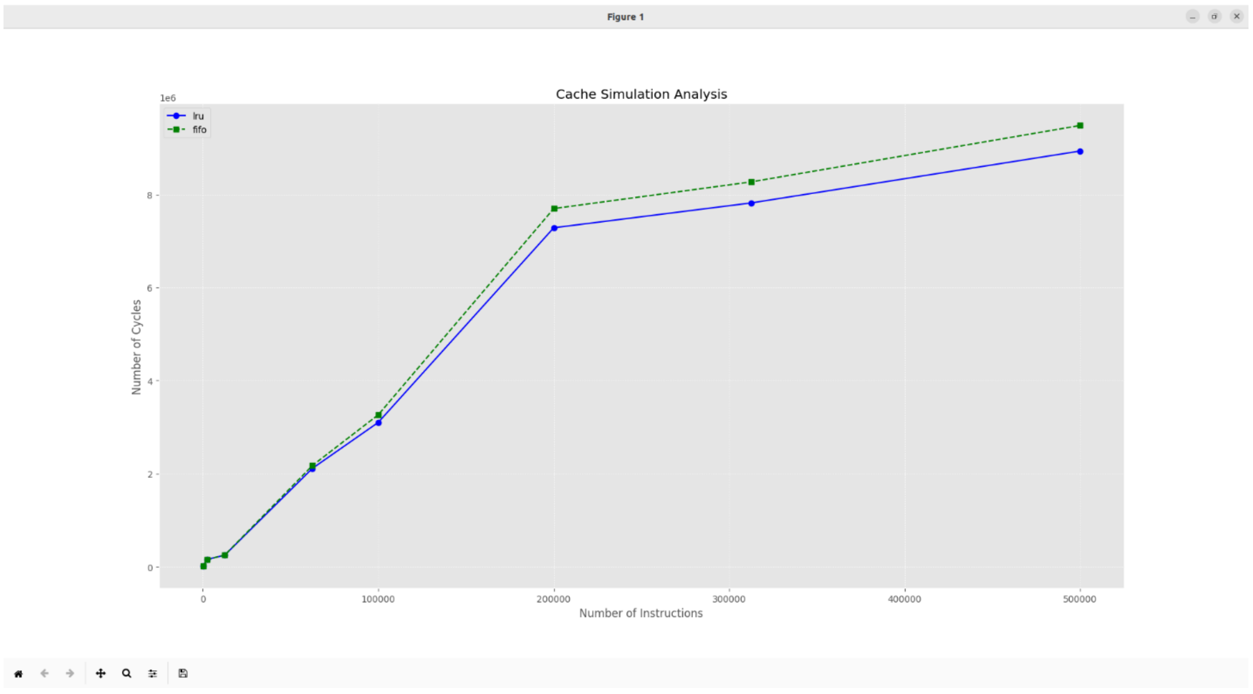


Figure 1
Cache Simulation Analysis
1e6
lru
fifo
Number of Cycles
Number of Instructions