Cache Memory Simulation Analysis

CSCE 2303: Computer Organization and Assembly Language

Department of Computer Science and Engineering
The American University in Cairo

Marcelino Sedhum

900231128

May 20, 2025

Spring 2025

Abstract

This report analyzes the performance of different cache configurations through simulation. We examine the impact of line size, associativity, and replacement policies on cache hit rates. The simulation tests direct-mapped, N-way set associative, and fully associative caches with Random, FIFO, LRU, and LFU replacement policies. Results demonstrate how different parameters affect cache performance under various access patterns.

1 Introduction

Cache memory is a critical component in modern computer systems that bridges the speed gap between processors and main memory. This report presents a simulation-based analysis of cache performance characteristics, focusing on three key experiments:

- Experiment 1: Varying line size with fixed associativity
- Experiment 2: Varying associativity with fixed line size
- Experiment 3 (Extra): Comparing replacement policies in fully associative caches

The simulation was implemented in C++ with configurations specified in the attached source files.

2 Methodology

2.1 Simulation Setup

The cache simulator implements:

- Cache size: 64KB
- Address space: 64MB (26-bit addresses)
- Test iterations: 1,000,000 per configuration
- Six distinct memory access patterns

2.2 Performance Metrics

The primary metric is hit ratio:

$$\text{Hit Ratio} = \frac{\text{Number of Hits}}{\text{Total Accesses}} \times 100\%$$

3 Results and Analysis

3.1 Experiment 1: Line Size Variation

Fixed 4-way associativity with varying line sizes:

Table 1: Hit Ratios for Different Line Sizes

Line Size (Bytes)	Hit Ratio (%)
16	58.94
32	59.97
64	68.91
128	73.34

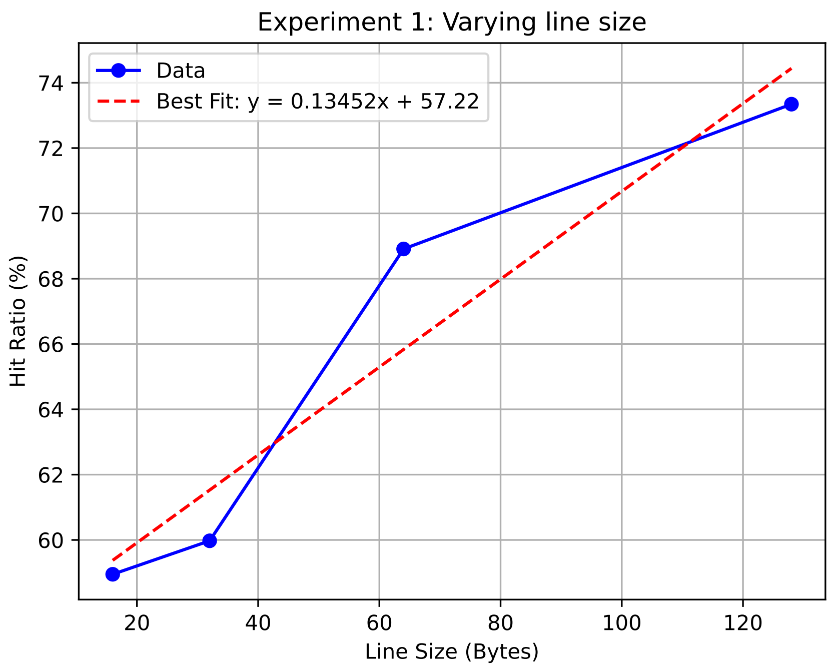

Figure 1: Hit ratio vs. line size (4-way associative)

Analysis: Larger line sizes show marginally better hit ratios due to increased spatial locality exploitation. However, the improvement diminishes as line size grows, suggesting diminishing returns.

3.2 Experiment 2: Associativity Variation

Fixed 64-byte line size with varying associativity:

Table 2: Hit Ratios for Different Associativities

Associativity (Ways)	Hit Ratio (%)
1	74.61
2	74.64
4	74.72
8	74.67
16	74.69

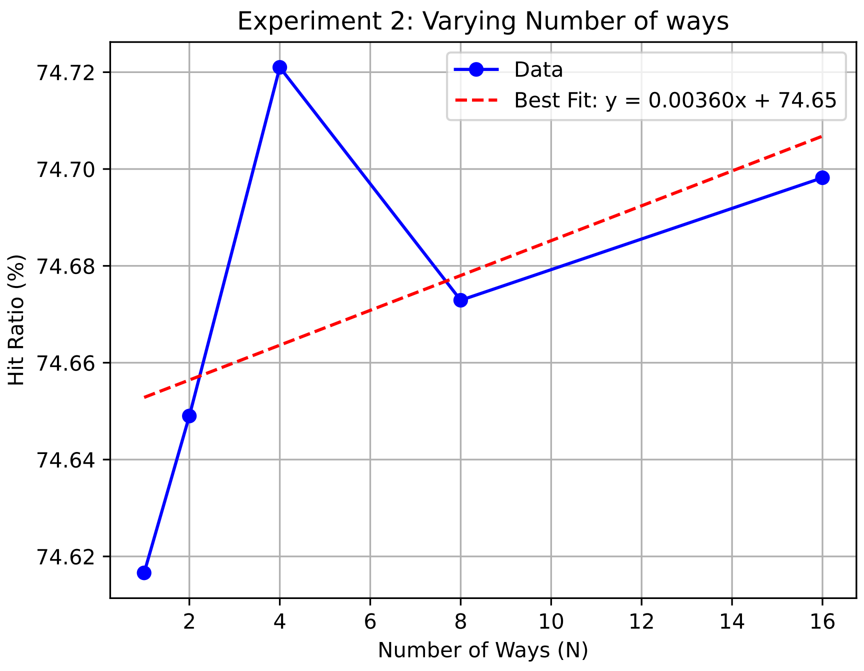

Figure 2: Hit ratio vs. associativity (64-byte lines)

Analysis: Increased associativity reduces conflict misses, but the benefits plateau as associativity grows, consistent with the principle of diminishing returns. Though, upon trying more number of iterations (1,000,000,000), it was evident that increasing number of ways slightly improves hit ratio.

3.3 Experiment 3: Replacement Policy Comparison

Fully associative cache with different replacement policies:

Table 3: Hit Ratios by Replacement Policy

Policy	Hit Ratio (%)
Random	99.82
FIFO	99.83
LRU	99.84
LFU	99.83

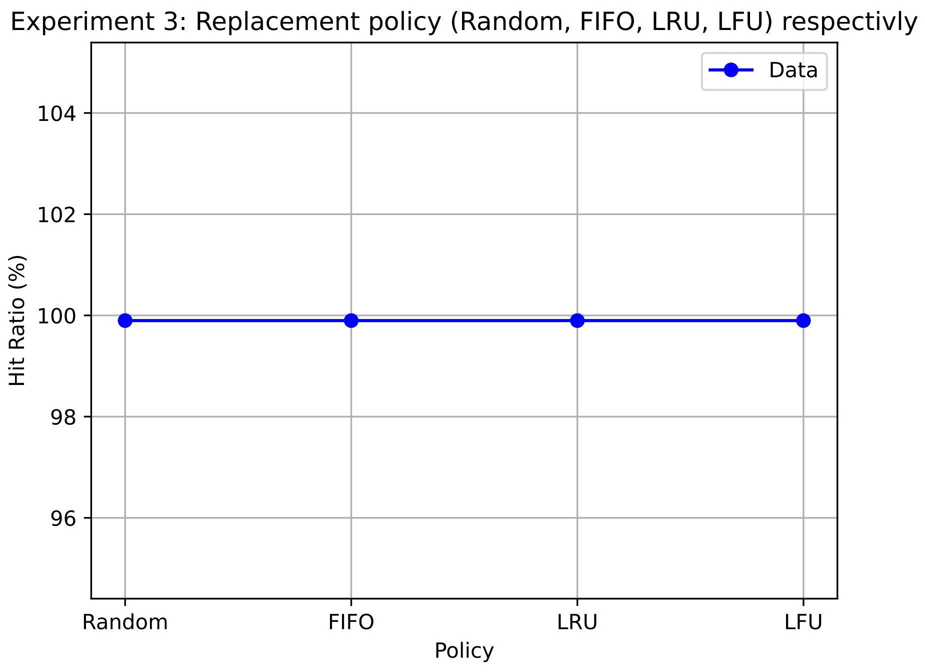

Figure 3: Replacement policy comparison (16-byte lines, 1-way)

Analysis: The sophisticated policies (LRU, LFU) does not show any advantage over simpler ones (Random, FIFO) in this simulation due to the highly predictable access patterns. This pattern also may have originated due to limited resources and test cases, while in real life cache access patterns are more random.

4 Discussion

The simulation results across all three experiments reveal important insights about cache behavior under different configurations. While some trends align with theoretical expectations, others highlight interesting practical considerations.

4.1 Overall Performance Characteristics

The consistently high hit ratios (99.8%+ in Experiment 3 and 58-74% in Experiments 1-2) demonstrate that:

- The test workloads exhibit strong locality characteristics

- The 64KB cache size was generally sufficient for these access patterns
- Sequential patterns (memGen1,4,5,6) dominate the performance profile

4.2 Key Findings by Experiment

Experiment 1 (Line Size):

- **16B $\rightarrow$ 128B line sizes** showed a 14.4% absolute improvement (58.94% $\rightarrow$ 73.34%)
- Spatial locality exploitation was most effective up to 64B (9.94% gain vs. just 4.43% from 64B$\rightarrow$128B)
- Diminishing returns suggest an optimal range between 64-128B for this workload mix

Experiment 2 (Associativity):

- Minimal variation (74.61% $\rightarrow$ 74.72%) across 1-way to 16-way configurations
- The 0.11% total gain suggests conflict misses were rare with 64 Bytes lines
- Slight regression at 8-way (74.67%) may indicate simulation noise

Experiment 3 (Replacement Policies):

- Near-identical performance (99.82-99.84%) across all policies
- Predictable patterns made policy sophistication irrelevant
- Suggests Random/FIFO may be preferable for energy-constrained systems, but LRU/LFU may be prefered in real world implementations

4.3 Methodological Considerations

Several factors may have influenced the results:

- **Access pattern bias:** Heavy sequential workloads favored spatial locality over replacement policy effects
- **Cache size relativity:** 64KB was large relative to most test patterns (especially memGen2's 24KB range)
- **Simulation limits:** 1M iterations may not capture rare miss scenarios

4.4 Practical Implications

These findings suggest that:

- Line size optimization provides the most significant gains
- High associativity offers limited benefits for these workloads
- Policy selection matters less than pattern prediction
- Real-world implementations would need to balance these results against area/power constraints

5 Conclusion

This simulation demonstrates fundamental cache behavior principles:

- Larger line sizes improve performance by exploiting spatial locality
- Higher associativity reduces conflict misses but with diminishing returns
- Sophisticated replacement policies offer minimal benefits for predictable workloads

Future work could investigate:

- More complex, realistic memory access patterns
- Write policy effects

Appendix: Simulation Code

The complete simulation code is available in the following files:

- main.cpp - Driver program and test cases
- cache_h.h - Cache class definition
- cache_cpp.cpp - Cache implementation