Impact of L1 Cache Size, Block Size, and Associativity on Matrix Multiplication Performance using SimpleScalar For LRU, FIFO and LFU Policy

A Report Submitted By

Avneel Singh
25CS06002

Under the Supervision of

Dr. Devashree Tripathy
Assistant Professor

Department of Computer Science and Engineering
School of Electrical and Computer Sciences (SECS)
IIT Bhubaneswar

Abstract

This mini-project studies how L1 cache parameters (cache size, block size, and associativity) influence the cache behavior and overall execution performance of a memory-intensive workload - dense matrix multiplication. We use the SimpleScalar sim-cache simulator to run controlled experiments, collect cache statistics (miss rates, hit rates), and analyze performance trade-offs. The final deliverable includes a clear problem statement, step-by-step procedure, scripts to run experiments, data collection and analysis instructions, sample plots, and conclusions.

Introduction

Modern processors can execute instructions much faster than data can be fetched from main memory. This mismatch between processor speed and memory access latency, known as the **memory wall**, can significantly degrade overall system performance. To overcome this, computer architects design a **memory hierarchy** that includes multiple levels of storage — from the fastest (registers, caches) to the slowest (main memory, disk).

Within this hierarchy, **cache memory** serves as a critical bridge between the CPU and main memory. It temporarily stores frequently accessed data and instructions so that subsequent accesses can be served much faster. By taking advantage of **temporal locality** (reuse of recently accessed data) and **spatial locality** (access to data located close to recently used addresses), cache memory helps reduce the average memory access time and keeps the processor pipelines busy.

The **Advanced Computer Architecture (ACA)** course focuses on understanding how such architectural optimizations impact performance. This mini-project explores cache design using the **SimpleScalar simulator** by analyzing how different L1 cache configurations affect the execution of a computationally intensive workload — matrix multiplication.

Background

A **cache** is a small, fast memory unit placed between the CPU and main memory. Its primary purpose is to reduce the effective memory access latency by storing a subset of data that is likely to be reused soon. Modern processors typically include multiple cache levels:

- **L1 Cache:** The smallest and fastest, located closest to the CPU core.
- **L2 Cache:** Larger but slightly slower, often private to each core.
- **L3 Cache:** Shared among cores, providing a final high-speed buffer before main memory.

The **L1 cache design** is particularly crucial because every memory reference from the CPU first checks this level. The hit or miss behavior of the L1 cache directly influences the instruction throughput (IPC) and processor stall time.

L1 cache performance depends on several design parameters:

- **Cache size:** Determines how much data can be stored. Larger caches reduce capacity misses but increase access latency.
- **Block size:** The unit of data transfer between cache and main memory. Larger blocks exploit spatial locality but may cause more misses if unused data fills the cache.
- **Associativity:** The number of cache lines per set, which helps reduce conflict misses but increases hardware complexity.
- **Replacement policy:** Determines which cache line is evicted when new data must be loaded (e.g., LRU, FIFO, or LFU).

Matrix multiplication is a suitable workload for studying these factors because it involves repetitive and structured memory accesses. Its performance is heavily influenced by cache behavior, as multiple elements from large matrices must be loaded repeatedly during computation. Analyzing how cache parameters affect the **miss rate** offers valuable insight into real-world architectural trade-offs in cache design.

Problem Statement

Statement: Matrix multiplication is a core computational kernel used extensively in scientific computing, image processing, and machine learning applications. Its performance depends heavily on memory hierarchy efficiency, particularly cache behavior, due to frequent and repetitive data accesses.

The **goal** of this mini-project is to evaluate how **different L1 cache** design parameters affect the performance of a matrix multiplication workload on a processor simulator.

Using the **SimpleScalar** simulation environment, the project involves:

1. Executing a dense integer-based matrix multiplication benchmark.
2. Systematically varying L1 cache parameters:
   - **Cache size**
   - **Block size**
   - **Set associativity**
3. Applying multiple cache replacement policies:
   - **LRU (Least Recently Used)**
   - **FIFO (First-In First-Out)**
   - **LFU (Least Frequently Used)**
4. Measuring performance metrics : Cache hit/miss ratio for both **Instruction Cache** and **Data Cache**

Objective: To analyze how cache configuration choices influence memory locality exploitation and overall cache efficiency, and to identify a cache configuration that delivers the best balance between performance improvement and hardware cost.

Motivation

Modern processors spend a significant portion of time waiting for data from main memory. The L1 cache, being the closest and fastest memory to the CPU, is the first line of defense against memory latency. Its design heavily affects performance, especially for memory-intensive operations such as matrix multiplication.

Matrix multiplication has a structured memory access pattern:

- **A[i][k]** is accessed row-wise to good spatial locality.
- **B[k][j]** is accessed column-wise to poor spatial locality.
- **C[i][j]** is updated multiple times to good temporal locality.

These access patterns interact differently with L1 cache parameters:

- **Cache size:** Larger caches can store more rows/columns of matrices, reducing capacity misses.
- **Block size:** Determines how many contiguous elements are fetched; larger blocks improve spatial locality but may cause unused data to occupy cache.
- **Associativity:** Reduces conflict misses when multiple memory locations map to the same cache set.

By systematically varying these parameters using **SimpleScalar**, the project allows us to:

- Quantify how each parameter affects cache hit/miss behavior.
- Observe the relationship between cache performance and overall execution efficiency.
- Make informed recommendations for L1 cache design for memory-intensive workloads.

Methodology

Requirements:

- Program: Matrix Multiplication in C
- Cross Compiler: sslittle-na-sstrix gcc (SimpleScalar PISA cross compiler)
- Parameters varied: L1 cache size (8KB–64KB), block size (16B–64B), associativity (1–8)
- Metrics collected: Miss rate

The methodology for this mini-project involves **four main steps**: coding the workload, compiling for SimpleScalar, configuring and running the simulator, and exploring L1 cache parameters. Each step is explained below.

1. Code

The workload used is **dense matrix multiplication** with fixed-size square matrices of size 32 x 32. This code is memory-intensive and allows us to study cache behavior using SimpleScalar.

Code Details:

- Matrices A, B, and C are declared as 2D arrays of floats.
- All elements of A and B are initialized with simple arithmetic formulas; C is initialized to zero.
- Standard triple-nested loops perform the matrix multiplication.
- A single element of C is printed to prevent compiler optimization.

2. Compilation

The program must be **cross-compiled** for the SimpleScalar target architecture using your provided cross-compiler.

Cross-compiler path:
/home/student/Desktop/simplescalar_project/sslittle-na-sstrix/bin/gcc

Compilation Steps:

1. Navigate to the directory containing your source code **matmul.c**.
2. Compile the code using the SimpleScalar cross-compiler with optimization (-O2 recommended).

Compilation Command:
/home/student/Desktop/simplescalar_project/sslittle-na-sstrix/bin/gcc -O2 -o matmul.pisa matmul.c

Explanation:

- -O2 enables compiler optimizations to produce realistic instruction execution.
- Output binary is named **matmul.pisa**, ready to run on the SimpleScalar simulator.

Output:

- matmul.pisa is the executable to be used with **sim-cache**.
- Optimizations ensure that the CPU executes meaningful memory accesses without unnecessary overhead.

3. Simulator Configuration (Updated for matmul.pisa)

Use **SimpleScalar sim-cache** to study L1 cache performance.

Sample Command:

../simplesim-3.0/sim-cache -cache:dl1 dl1:8:16:1:l -cache:il1 il1:8:16:1:l -cache:dl2 none -cache:il2 none -redir:sim ./lru/output_dl1_8k_b16_a1.txt matmul.pisa

Explanation:

- dl1:16:32:2:l to L1 data cache: 16 KB, 32-byte blocks, 2-way set associative, write-back.
- il1:16:32:2:l to L1 instruction cache: same configuration.
- ./matmul.pisa to binary compiled with your cross-compiler.

Automation:

- A shell script can iterate over multiple cache sizes, block sizes, and associativities, using matmul.pisa as the executable for all simulations.
- Each configuration is stored in a separate output file for later parsing and analysis.

Observations

This section presents the outcome of cache simulations performed using the SimpleScalar sim-cache tool. The benchmark used for testing was a standard matrix multiplication program with a 32×32 matrix size. The cache performance was analyzed by varying key L1 cache parameters — cache size, block (line) size, and set associativity - independently for both the Instruction Cache (I-Cache) and the Data Cache (D-Cache).

The experiments were conducted under three replacement policies: **LRU** (Least Recently Used), **FIFO** (First-In First-Out), and **LFU** (Least Frequently Used). The primary performance metric considered in this study is the **miss rate**, since lower miss rates directly translate to fewer memory accesses and therefore better execution performance.

The subsequent tables and graphs summarize the measured miss rates, followed by a detailed analysis of performance trends for each parameter and each replacement policy.

➢ Performance Analysis For LRU Replacement Policy

Configurations			LRU			
Cache Size	Block Size	Associativity	Instruction Cache		Data Cache	
			Hit Rate	Miss Rate	Hit Rate	Miss Rate
8KB	16B	1	0.9689	0.0311	0.4668	0.5332
		2	0.9891	0.0109	0.6724	0.3276
		4	0.9895	0.0105	0.7410	0.2590
		8	0.9915	0.0085	0.7418	0.2582
	32B	1	0.9936	0.0064	0.5309	0.4691
		2	0.9939	0.0061	0.6754	0.3246
		4	0.9948	0.0052	0.7499	0.2501
		8	0.9960	0.0040	0.7504	0.2496

	64B	1	0.9962	0.0038	0.5169	0.4831
		2	0.9967	0.0033	0.6878	0.3122
		4	0.9976	0.0024	0.7513	0.2487
		8	0.9981	0.0019	0.9051	0.0949
16KB	16B	1	0.9891	0.0109	0.5701	0.4299
		2	0.9896	0.0104	0.6989	0.3011
		4	0.9915	0.0085	0.7416	0.2584
		8	0.9930	0.0070	0.7426	0.2574
	32B	1	0.9940	0.0060	0.5758	0.4242
		2	0.9946	0.0054	0.7031	0.2969
		4	0.9960	0.0040	0.7504	0.2496
		8	0.9970	0.0030	0.9239	0.0761
	64B	1	0.9967	0.0033	0.5394	0.4606
		2	0.9975	0.0025	0.7092	0.2908
		4	0.9981	0.0019	0.9458	0.0542
		8	0.9986	0.0014	0.9955	0.0045
32KB	16B	1	0.9898	0.0102	0.6216	0.3784
		2	0.9911	0.0089	0.7119	0.2881
		4	0.9930	0.0070	0.7422	0.2578
		8	0.9950	0.0050	0.9041	0.0959
	32B	1	0.9947	0.0053	0.5986	0.4014
		2	0.9959	0.0041	0.7170	0.2830
		4	0.9969	0.0031	0.9514	0.0486
		8	0.9977	0.0023	0.9914	0.0086
	64B	1	0.9973	0.0027	0.5509	0.4491
		2	0.9980	0.0020	0.9324	0.0676
		4	0.9986	0.0014	0.9955	0.0045
		8	0.9987	0.0013	0.9968	0.0032
64KB	16B	1	0.9912	0.0088	0.6478	0.3522
		2	0.9929	0.0071	0.7187	0.2813
		4	0.9948	0.0052	0.9395	0.0605
		8	0.9958	0.0042	0.9832	0.0168
	32B	1	0.9957	0.0043	0.6100	0.3900
		2	0.9967	0.0033	0.9410	0.0590
		4	0.9976	0.0024	0.9914	0.0086

		8	0.9977	0.0023	0.9939	0.0061
	64B	1	0.9976	0.0024	0.7793	0.2207
		2	0.9983	0.0017	0.9574	0.0426
		4	0.9987	0.0013	0.9968	0.0032
		8	0.9988	0.0012	0.9969	0.0031

➢ Performance Analysis For FIFO Replacement Policy

Configurations			FIFO			
Cache Size	Block Size	Associativity	Instruction Cache		Data Cache	
			Hit Rate	Miss Rate	Hit Rate	Miss Rate
8KB	16B	1	0.9689	0.0311	0.4668	0.5332
		2	0.9891	0.0109	0.5714	0.4286
		4	0.9894	0.0106	0.6733	0.3267
		8	0.9914	0.0086	0.7129	0.2871
	32B	1	0.9936	0.0064	0.5309	0.4691
		2	0.9939	0.0061	0.6319	0.3681
		4	0.9947	0.0053	0.7101	0.2899
		8	0.9960	0.0040	0.7352	0.2648
	64B	1	0.9962	0.0038	0.5169	0.4831
		2	0.9967	0.0033	0.6651	0.3349
		4	0.9976	0.0024	0.7300	0.2700
		8	0.9980	0.0020	0.8961	0.1039
16KB	16B	1	0.9891	0.0109	0.5701	0.4299
		2	0.9896	0.0104	0.6486	0.3514
		4	0.9914	0.0086	0.7117	0.2883
		8	0.9929	0.0071	0.7279	0.2721
	32B	1	0.9940	0.0060	0.5758	0.4242
		2	0.9947	0.0053	0.6796	0.3204
		4	0.9960	0.0040	0.7342	0.2658
		8	0.9970	0.0030	0.9150	0.0850

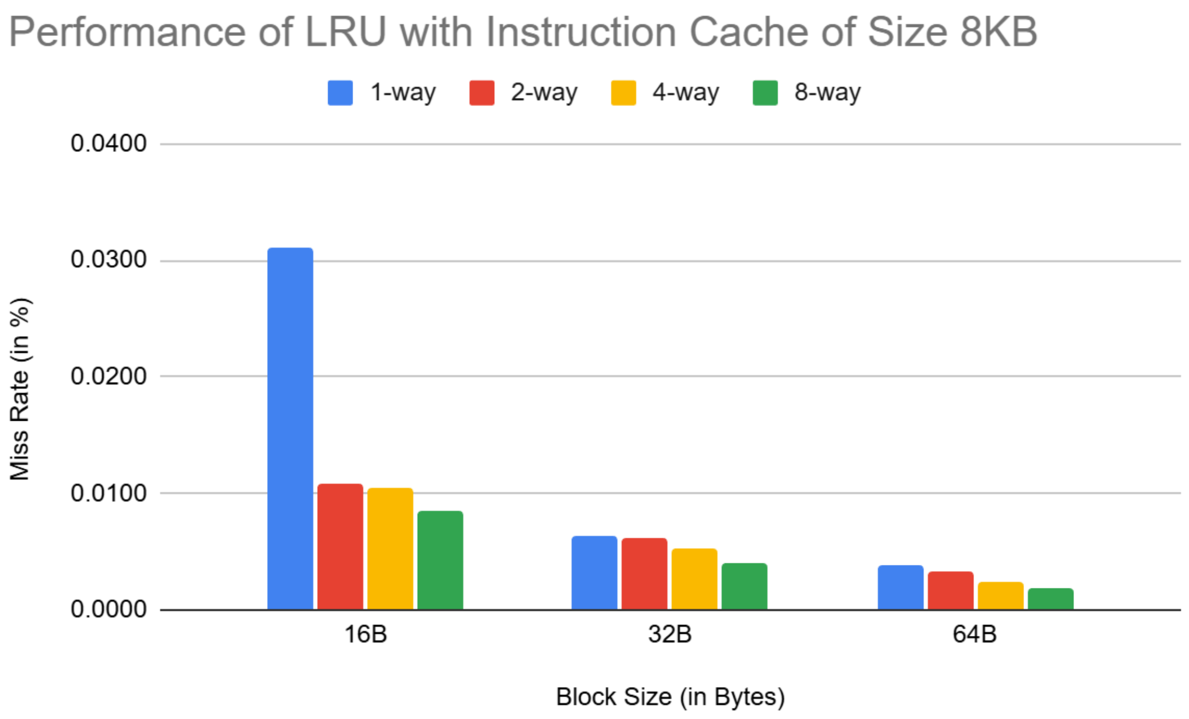
	64B	1	0.9967	0.0033	0.5394	0.4606
		2	0.9974	0.0026	0.6972	0.3028
		4	0.9981	0.0019	0.9329	0.0671
		8	0.9986	0.0014	0.9949	0.0051
32KB	16B	1	0.9898	0.0102	0.6216	0.3784
		2	0.9911	0.0089	0.6841	0.3159
		4	0.9929	0.0071	0.7264	0.2736
		8	0.9949	0.0051	0.9055	0.0945
	32B	1	0.9947	0.0053	0.5986	0.4014
		2	0.9959	0.0041	0.7031	0.2969
		4	0.9968	0.0032	0.9427	0.0573
		8	0.9976	0.0024	0.9903	0.0097
	64B	1	0.9973	0.0027	0.5509	0.4491
		2	0.9979	0.0021	0.9237	0.0763
		4	0.9986	0.0014	0.9950	0.0050
		8	0.9987	0.0013	0.9968	0.0032
64KB	16B	1	0.9912	0.0088	0.6478	0.3522
		2	0.9929	0.0071	0.7032	0.2968
		4	0.9948	0.0052	0.9323	0.0677
		8	0.9957	0.0043	0.9810	0.0190
	32B	1	0.9957	0.0043	0.6100	0.3900
		2	0.9967	0.0033	0.9324	0.0676
		4	0.9976	0.0024	0.9905	0.0095
		8	0.9977	0.0023	0.9939	0.0061
	64B	1	0.9976	0.0024	0.7793	0.2207
		2	0.9983	0.0017	0.9531	0.0469
		4	0.9986	0.0014	0.9968	0.0032
		8	0.9988	0.0012	0.9969	0.0031

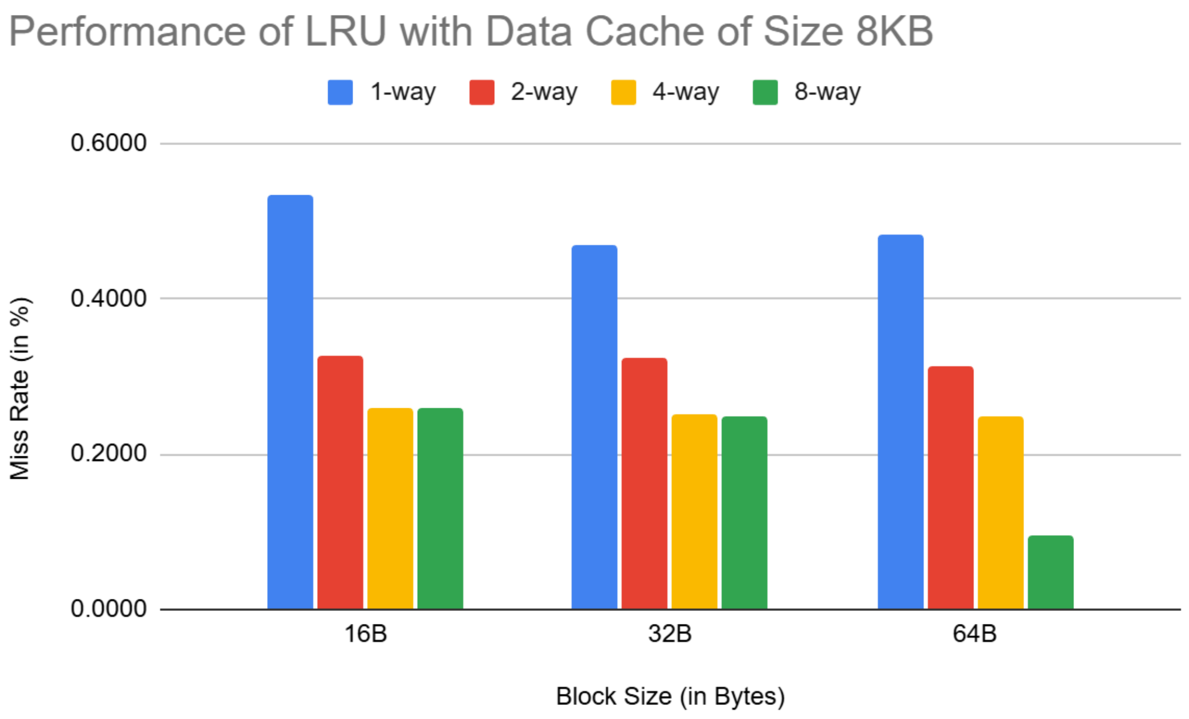
➢ Performance Analysis For LFU Replacement Policy

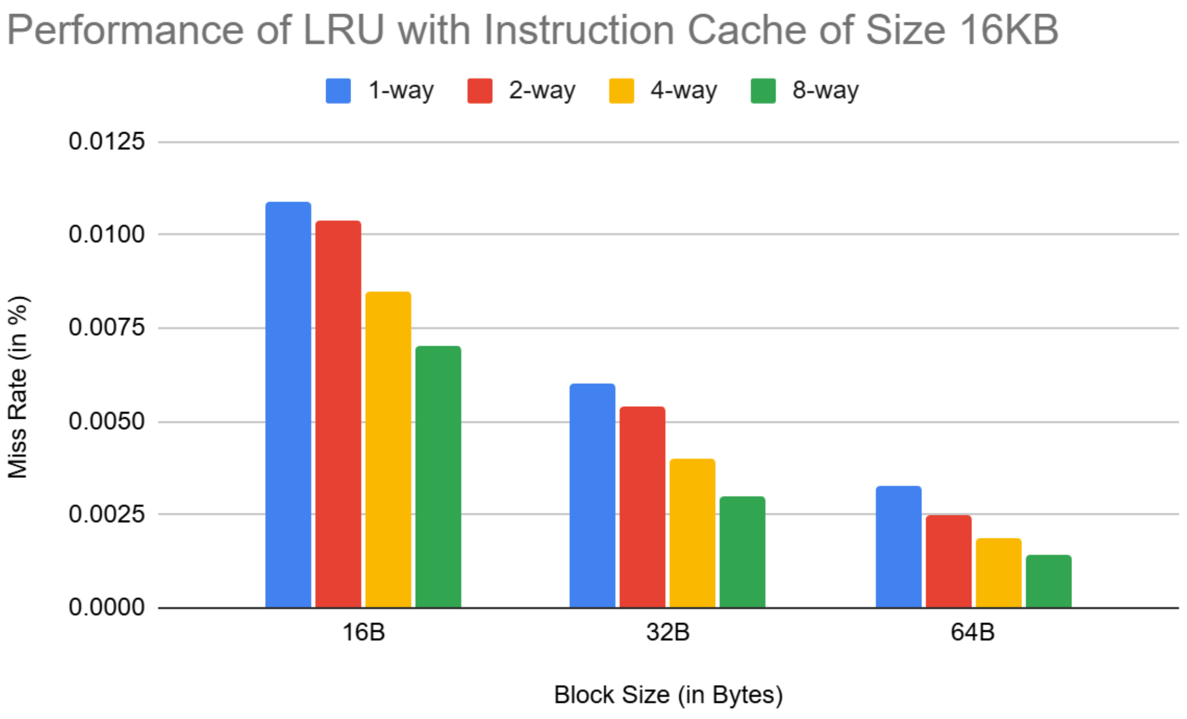
Configurations			LFU			
Cache Size	Block Size	Associativity	Instruction Cache		Data Cache	
			Hit Rate	Miss Rate	Hit Rate	Miss Rate
8KB	16B	1	0.9689	0.0311	0.4669	0.5331
		2	0.9698	0.0302	0.4678	0.5322
		4	0.9747	0.0253	0.4742	0.5258
		8	0.9750	0.0250	0.4887	0.5113
	32B	1	0.9936	0.0064	0.5309	0.4691
		2	0.9936	0.0064	0.5378	0.4622
		4	0.9936	0.0064	0.5492	0.4508
		8	0.9940	0.0060	0.5802	0.4198
	64B	1	0.9962	0.0038	0.5169	0.4831
		2	0.9963	0.0037	0.5358	0.4642
		4	0.9964	0.0036	0.5731	0.4269
		8	0.9968	0.0032	0.6690	0.3310
16KB	16B	1	0.9891	0.0109	0.5701	0.4299
		2	0.9891	0.0109	0.5734	0.4266
		4	0.9892	0.0108	0.5841	0.4159
		8	0.9900	0.0100	0.6088	0.3912
	32B	1	0.9940	0.0060	0.5758	0.4242
		2	0.9940	0.0060	0.5887	0.4113
		4	0.9943	0.0057	0.6117	0.3883
		8	0.9949	0.0051	0.6884	0.3116
	64B	1	0.9967	0.0033	0.5394	0.4606
		2	0.9967	0.0033	0.5873	0.4127
		4	0.9970	0.0030	0.6564	0.3436
		8	0.9976	0.0024	0.8611	0.1389
32KB	16B	1	0.9898	0.0102	0.6216	0.3784
		2	0.9899	0.0101	0.6292	0.3708
		4	0.9904	0.0096	0.6475	0.3525
		8	0.9915	0.0085	0.7104	0.2896

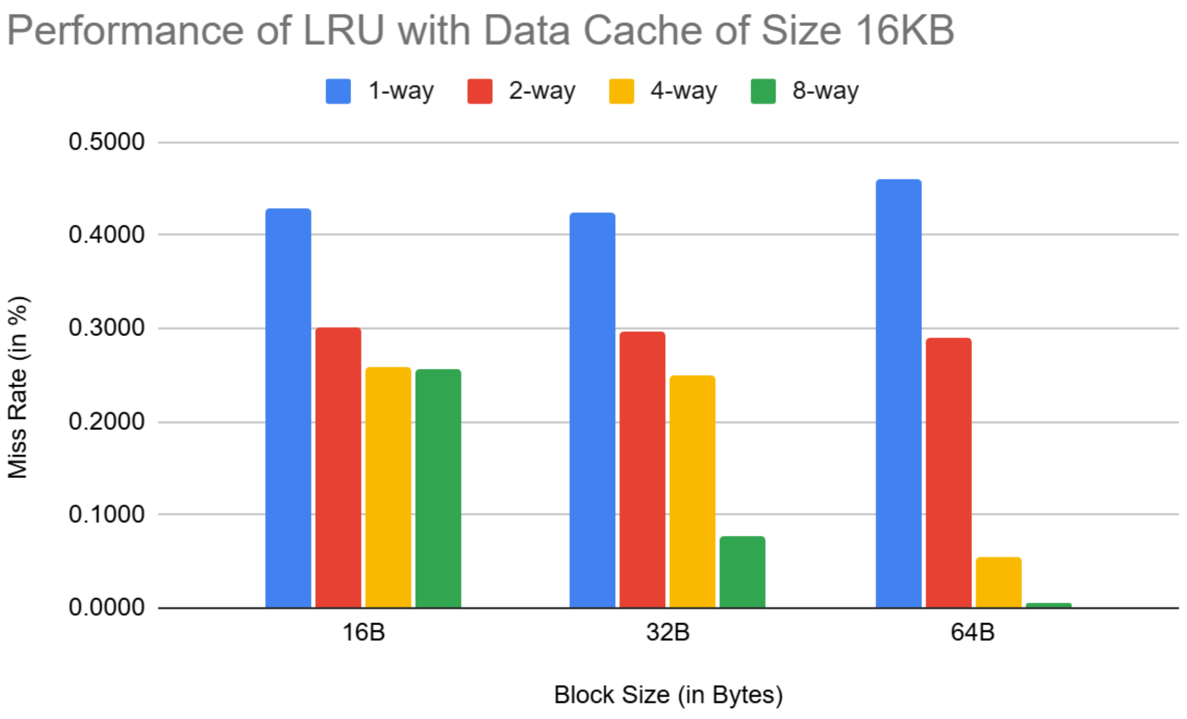
	32B	1	0.9947	0.0053	0.5986	0.4014
		2	0.9948	0.0052	0.6273	0.3727
		4	0.9953	0.0047	0.6750	0.3250
		8	0.9962	0.0038	0.8851	0.1149
	64B	1	0.9973	0.0027	0.5509	0.4491
		2	0.9975	0.0025	0.7092	0.2908
		4	0.9979	0.0021	0.8841	0.1159
		8	0.9985	0.0015	0.9965	0.0035
64KB	16B	1	0.9912	0.0088	0.6478	0.3522
		2	0.9915	0.0085	0.6666	0.3334
		4	0.9926	0.0074	0.7044	0.2956
		8	0.9934	0.0066	0.9035	0.0965
	32B	1	0.9957	0.0043	0.6100	0.3900
		2	0.9961	0.0039	0.7015	0.2985
		4	0.9967	0.0033	0.9015	0.0985
		8	0.9974	0.0026	0.9937	0.0063
	64B	1	0.9976	0.0024	0.7793	0.2207
		2	0.9979	0.0021	0.8473	0.1527
		4	0.9985	0.0015	0.9966	0.0034
		8	0.9988	0.0012	0.9969	0.0031

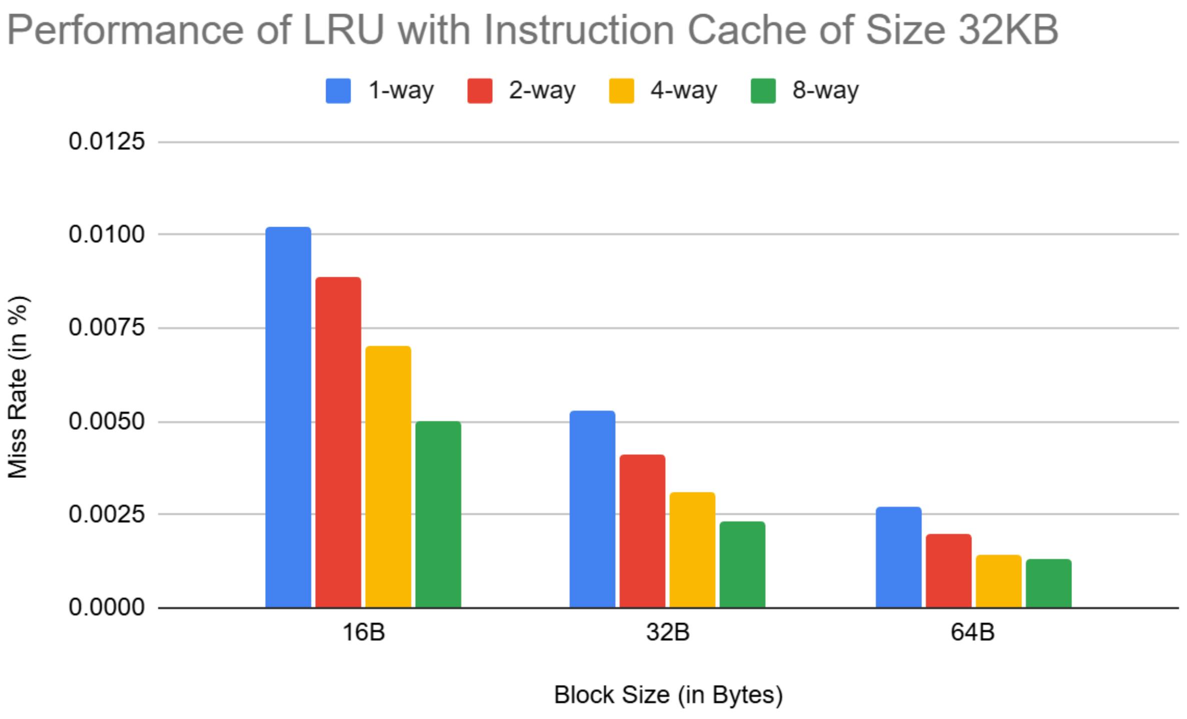
➢ The Graphs for LRU Replacement Policy

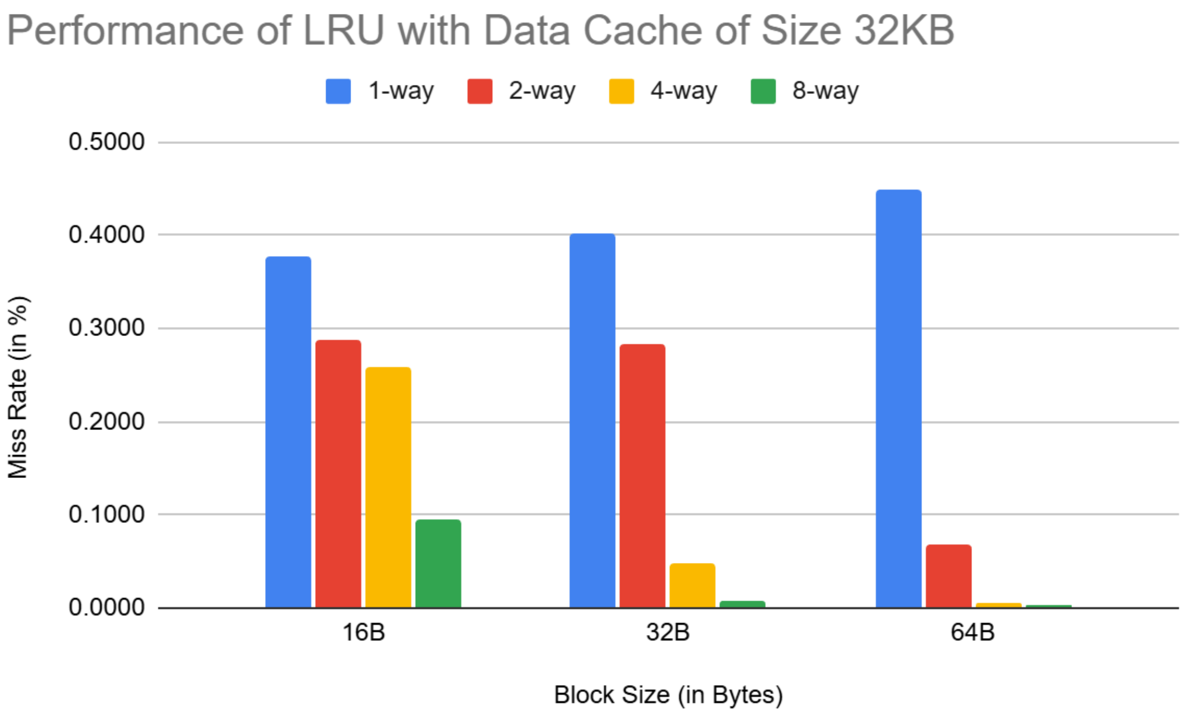


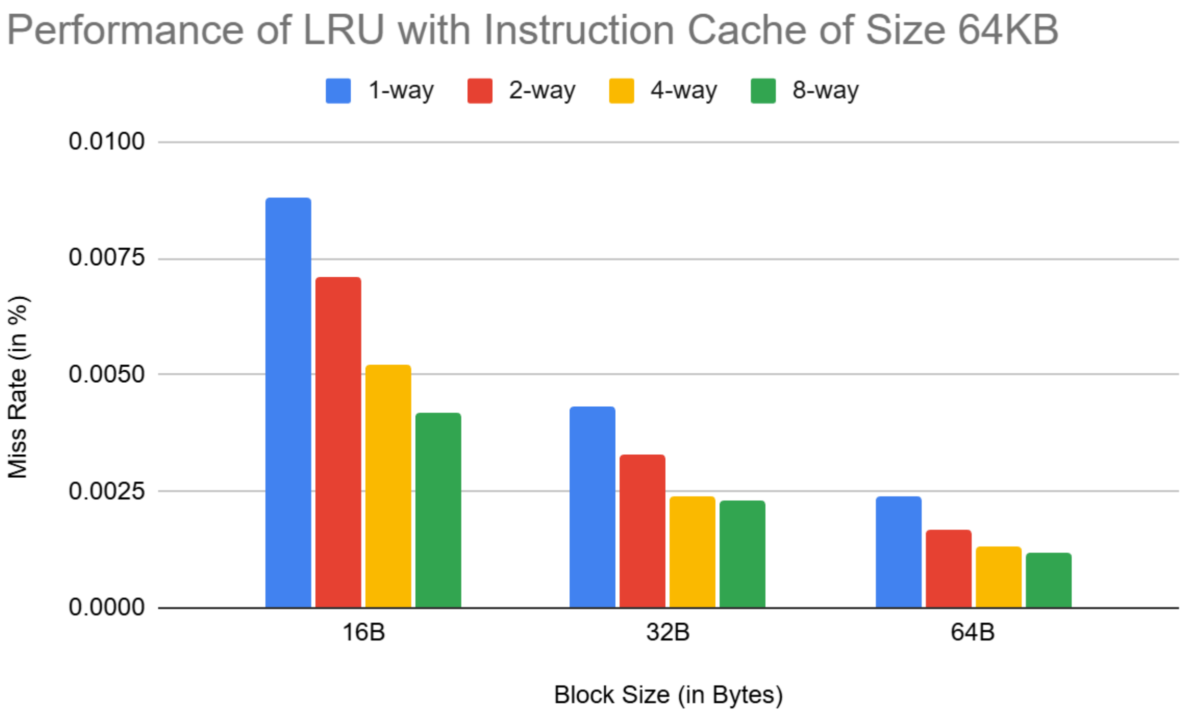


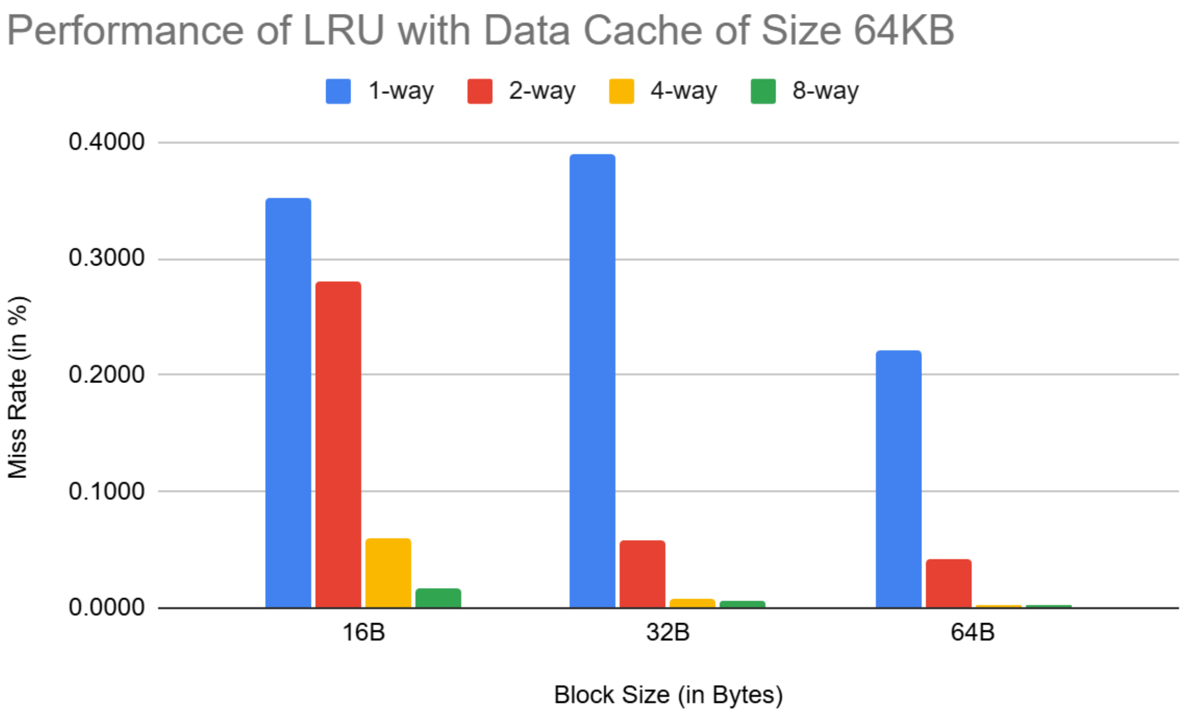

Performance of LRU with Instruction Cache of Size 16KB
1-way
2-way
4-way
8-way
0.0125
0.0100
0.0075
0.0050
0.0025
0.0000
Miss Rate (in %)
16B
32B
64B
Block Size (in Bytes)


Performance of LRU with Data Cache of Size 16KB
1-way
2-way
4-way
8-way
0.5000
0.4000
0.3000
0.2000
0.1000
0.0000
Miss Rate (in %)
16B
32B
64B
Block Size (in Bytes)

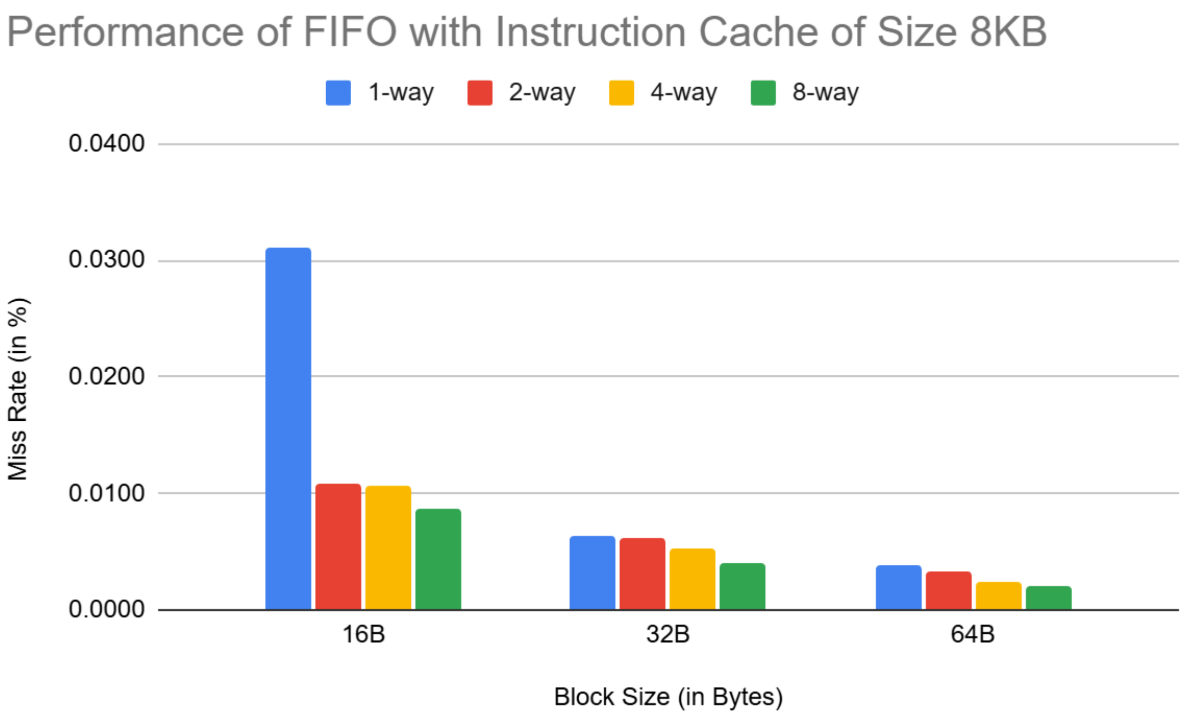

Performance of LRU with Instruction Cache of Size 32KB
1-way
2-way
4-way
8-way
Miss Rate (in %)
0.0125
0.0100
0.0075
0.0050
0.0025
0.0000
16B
32B
64B
Block Size (in Bytes)

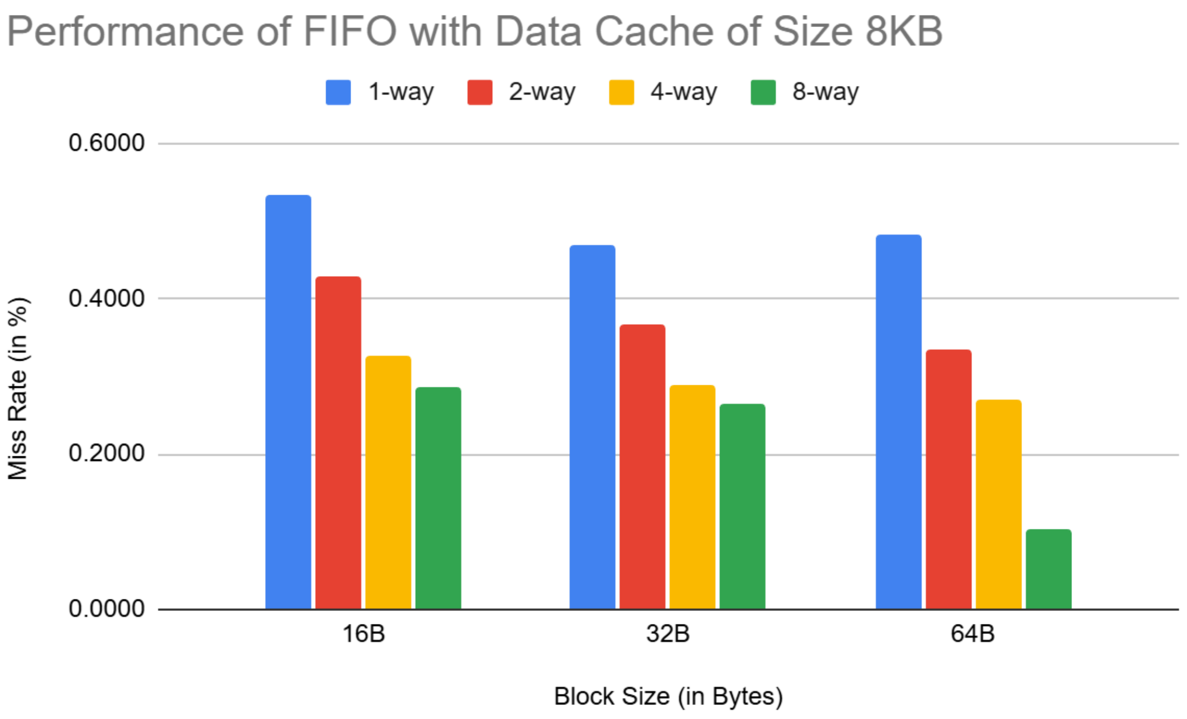

Performance of LRU with Data Cache of Size 32KB
1-way
2-way
4-way
8-way
Miss Rate (in %)
0.5000
0.4000
0.3000
0.2000
0.1000
0.0000
16B
32B
64B
Block Size (in Bytes)

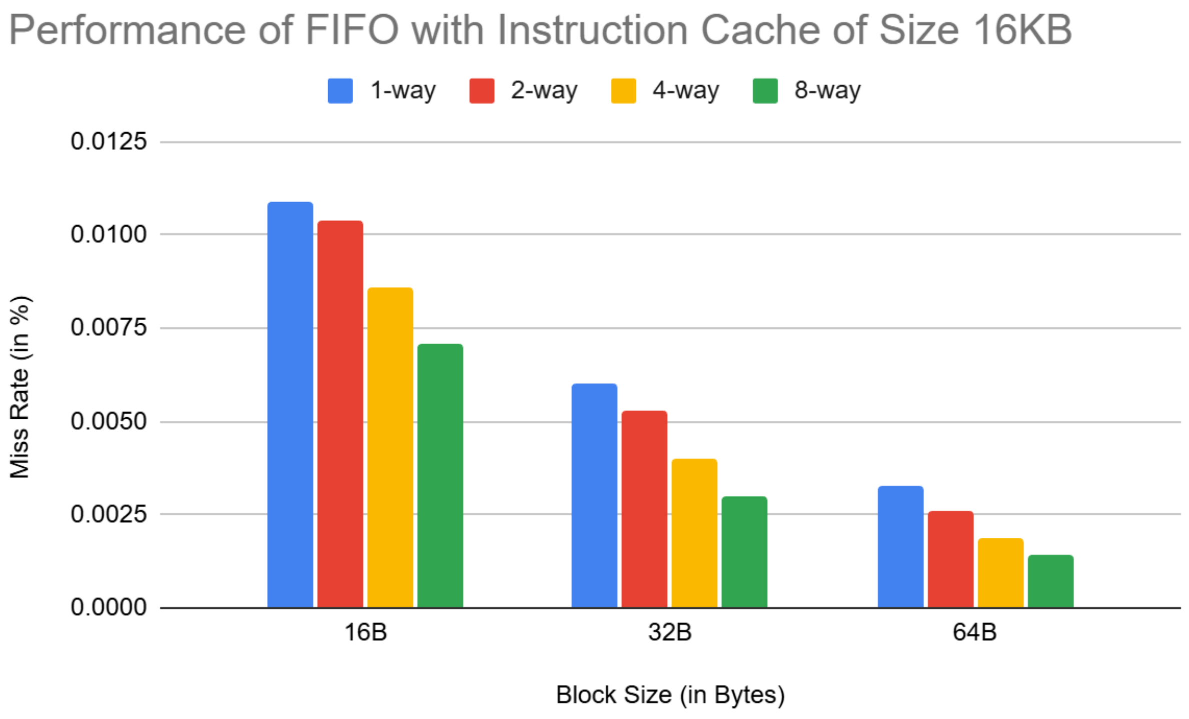

Performance of LRU with Instruction Cache of Size 64KB
1-way
2-way
4-way
8-way
0.0100
0.0075
0.0050
0.0025
0.0000
Miss Rate (in %)
16B
32B
64B
Block Size (in Bytes)

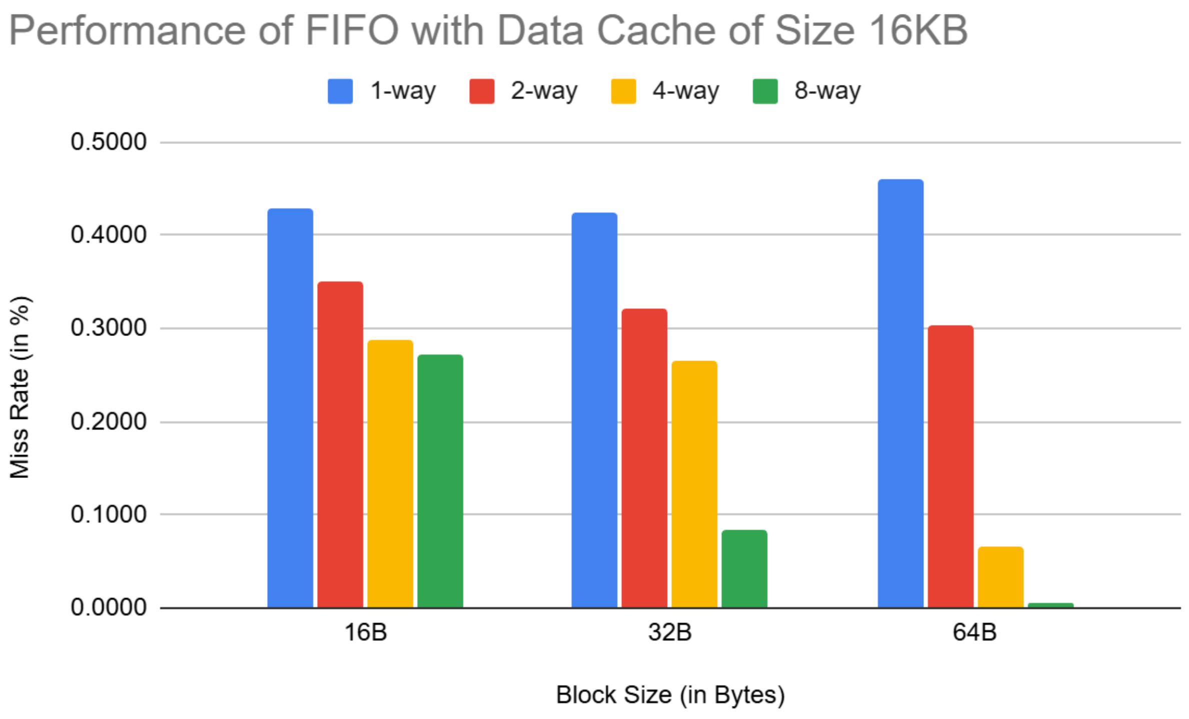

Performance of LRU with Data Cache of Size 64KB
1-way
2-way
4-way
8-way
0.4000
0.3000
0.2000
0.1000
0.0000
Miss Rate (in %)
16B
32B
64B
Block Size (in Bytes)

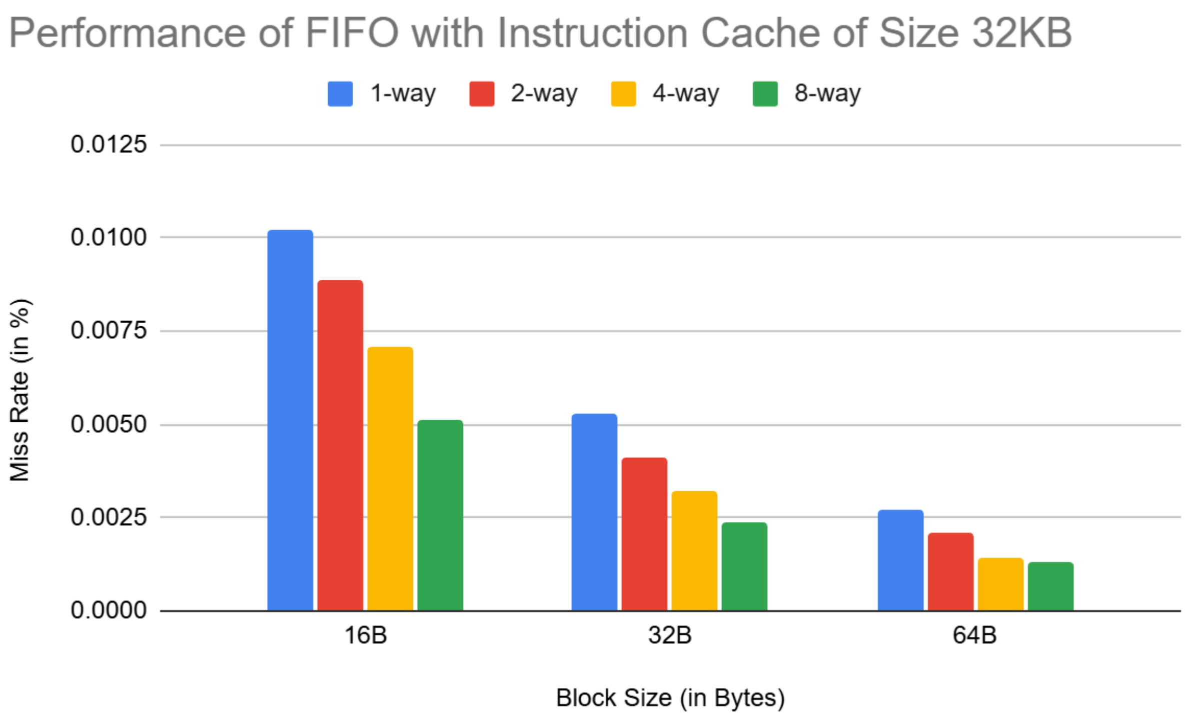
➢ The Graphs for FIFO Replacement Policy

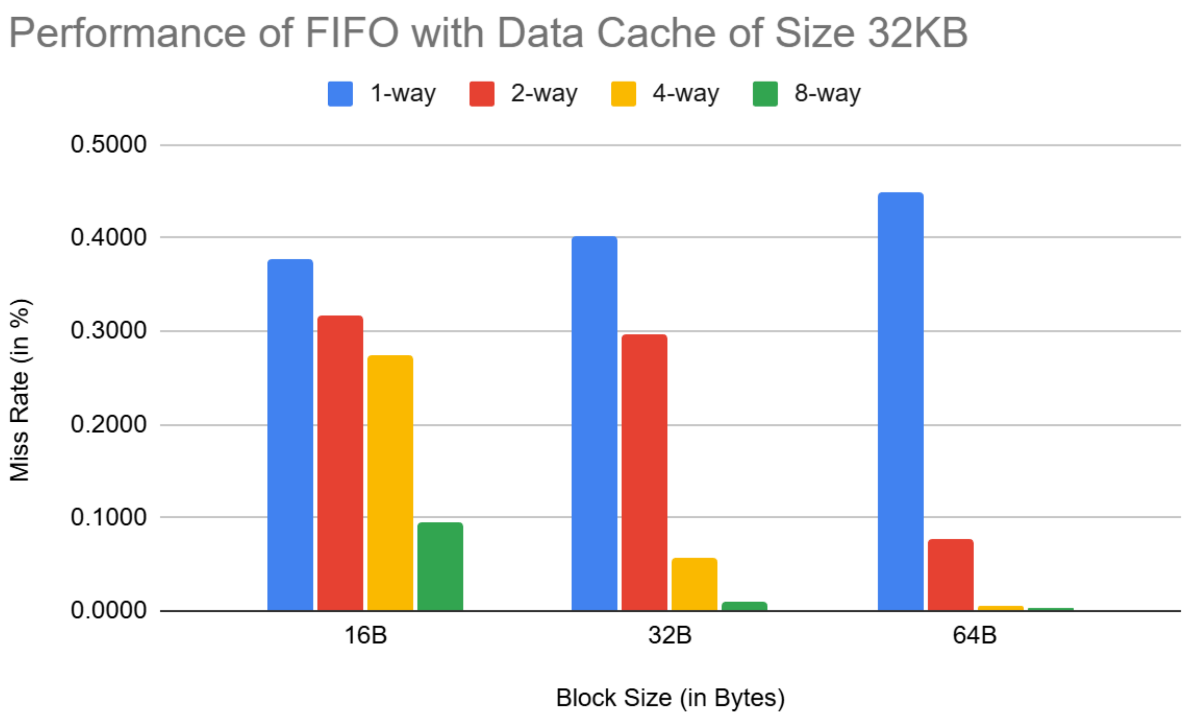


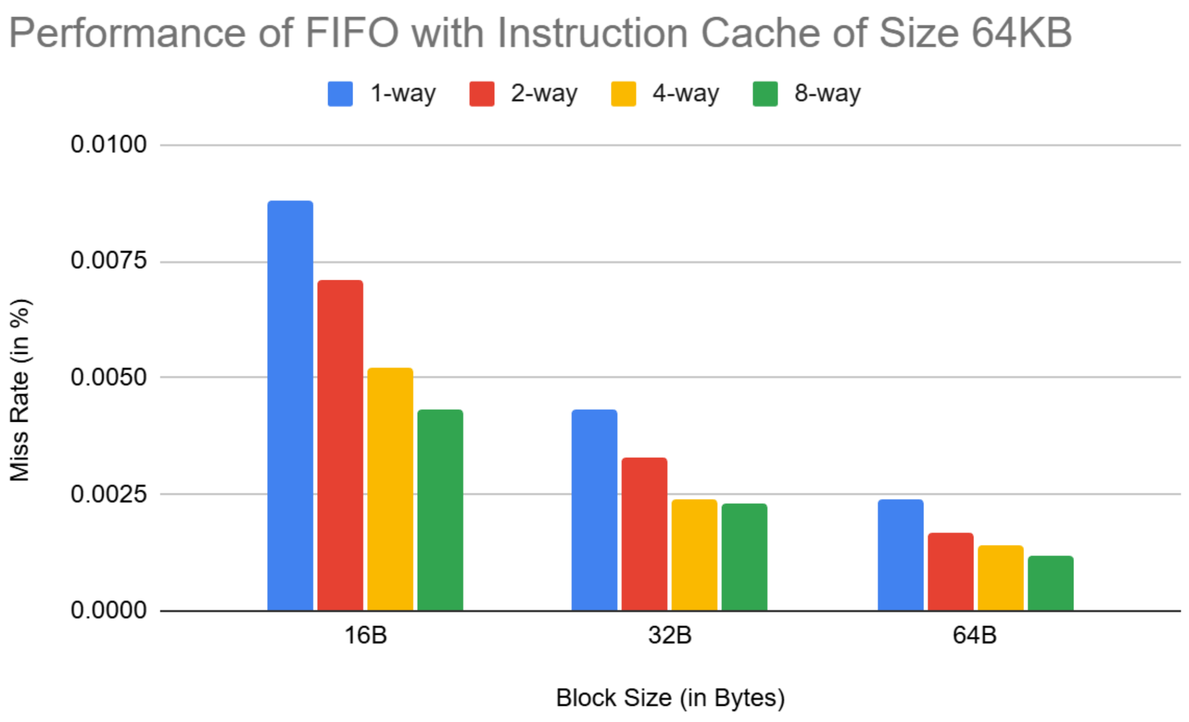


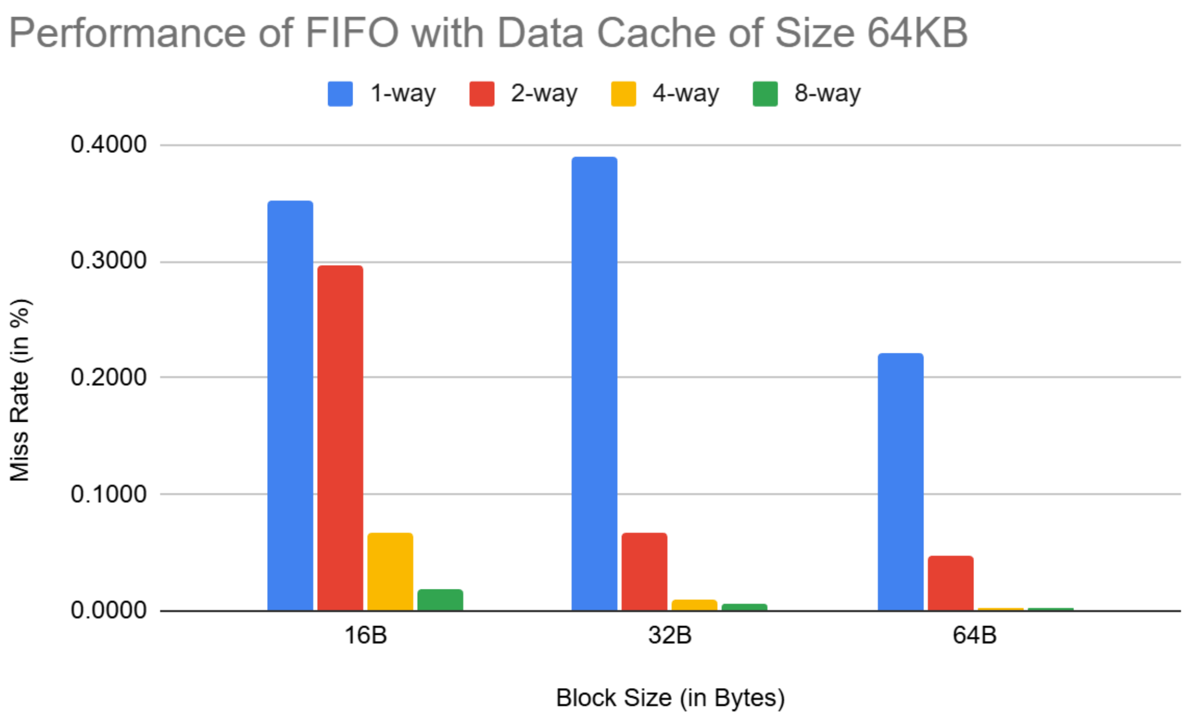

Performance of FIFO with Instruction Cache of Size 16KB
1-way
2-way
4-way
8-way
0.0125
0.0100
0.0075
0.0050
0.0025
0.0000
Miss Rate (in %)
16B
32B
64B
Block Size (in Bytes)


Performance of FIFO with Data Cache of Size 16KB
1-way
2-way
4-way
8-way
0.5000
0.4000
0.3000
0.2000
0.1000
0.0000
Miss Rate (in %)
16B
32B
64B
Block Size (in Bytes)

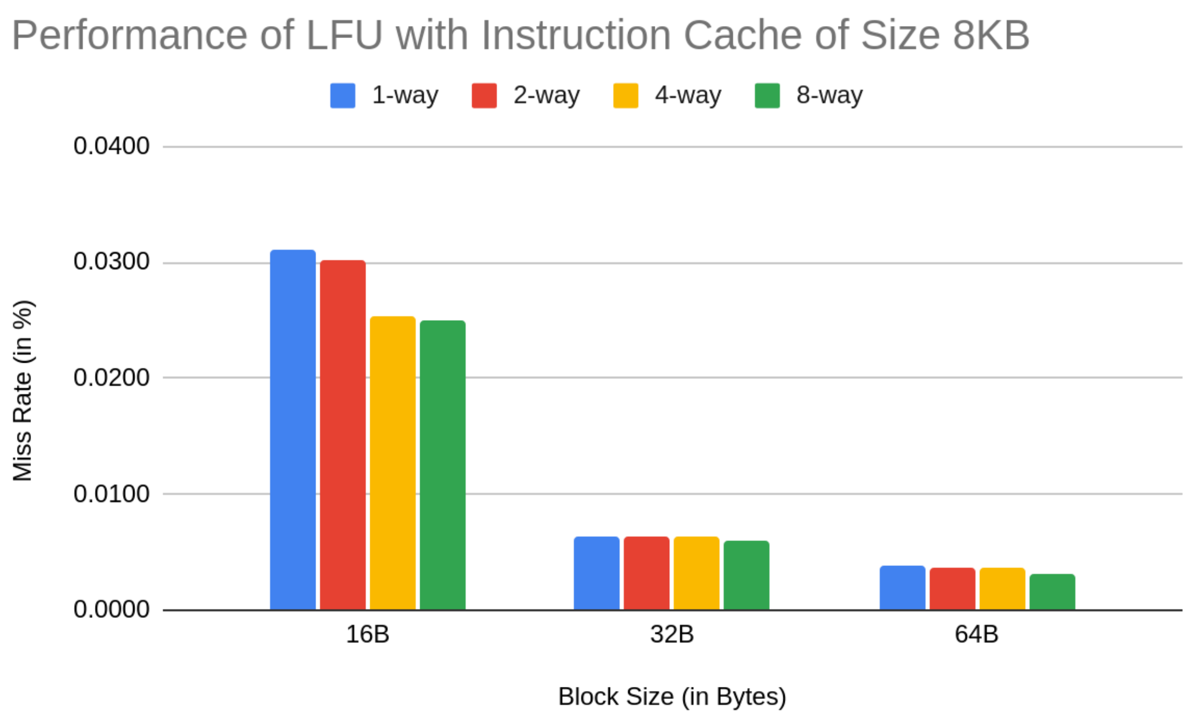

Performance of FIFO with Instruction Cache of Size 32KB
1-way
2-way
4-way
8-way
Miss Rate (in %)
0.0125
0.0100
0.0075
0.0050
0.0025
0.0000
16B
32B
64B
Block Size (in Bytes)

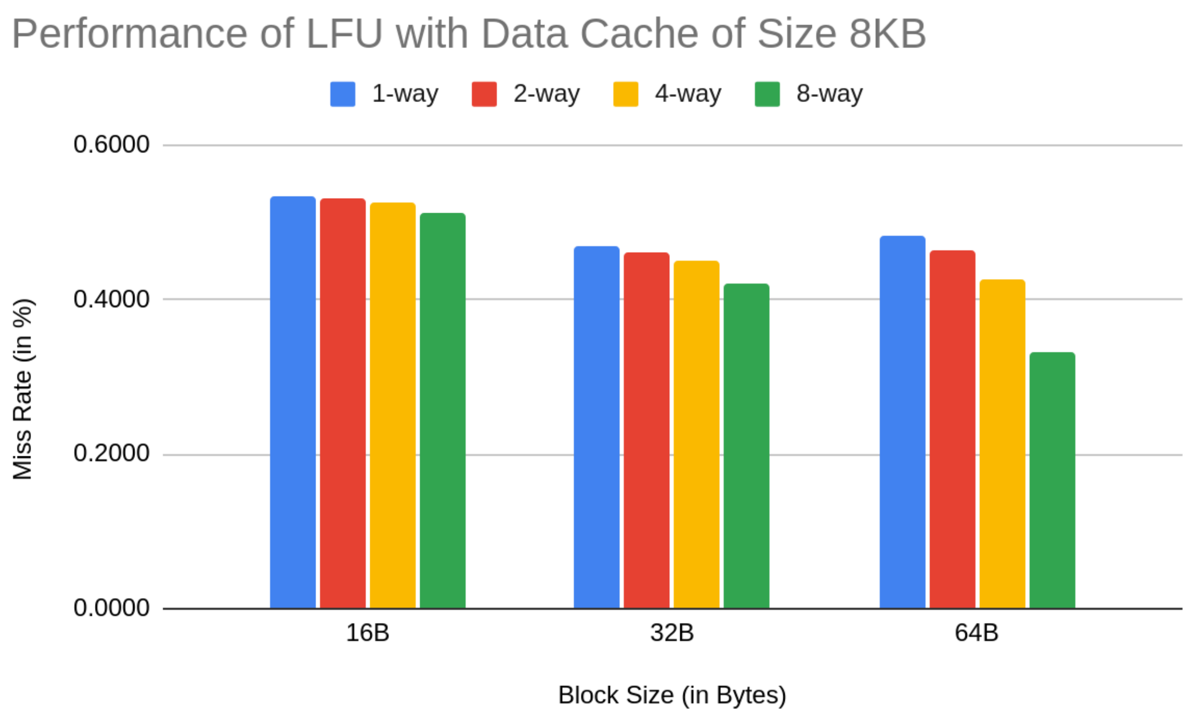

Performance of FIFO with Data Cache of Size 32KB
1-way
2-way
4-way
8-way
Miss Rate (in %)
0.5000
0.4000
0.3000
0.2000
0.1000
0.0000
16B
32B
64B
Block Size (in Bytes)

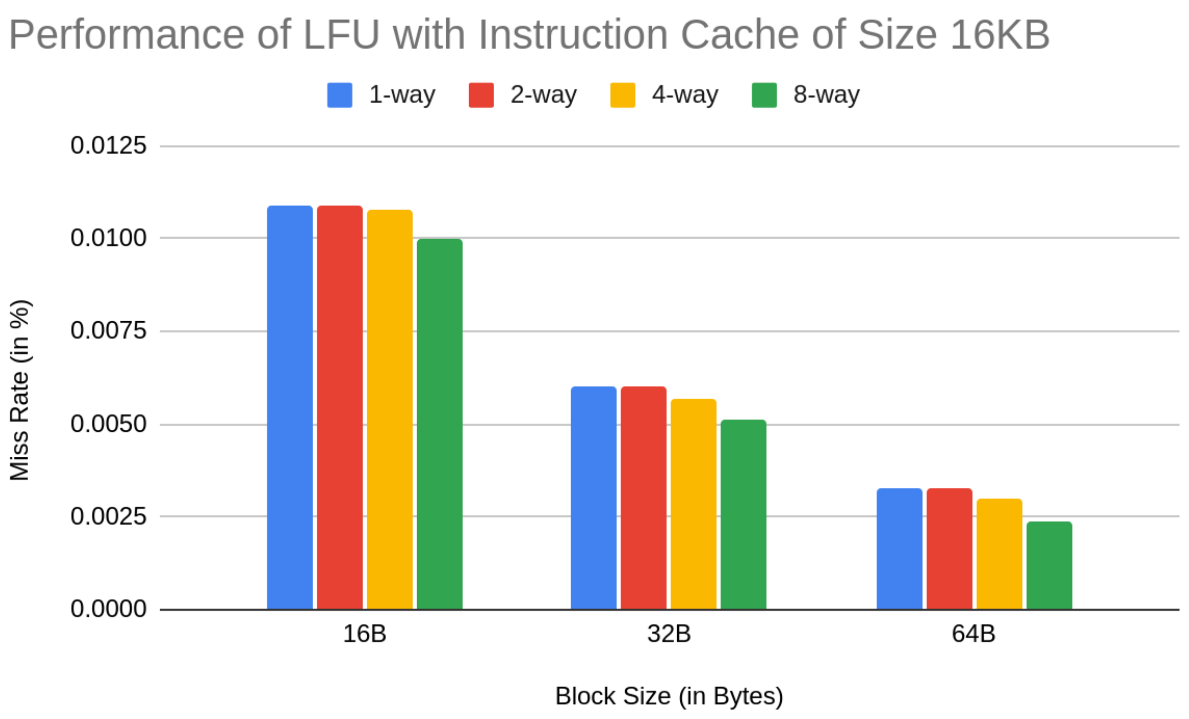

Performance of FIFO with Instruction Cache of Size 64KB
1-way
2-way
4-way
8-way
Miss Rate (in %)
0.0100
0.0075
0.0050
0.0025
0.0000
16B
32B
64B
Block Size (in Bytes)

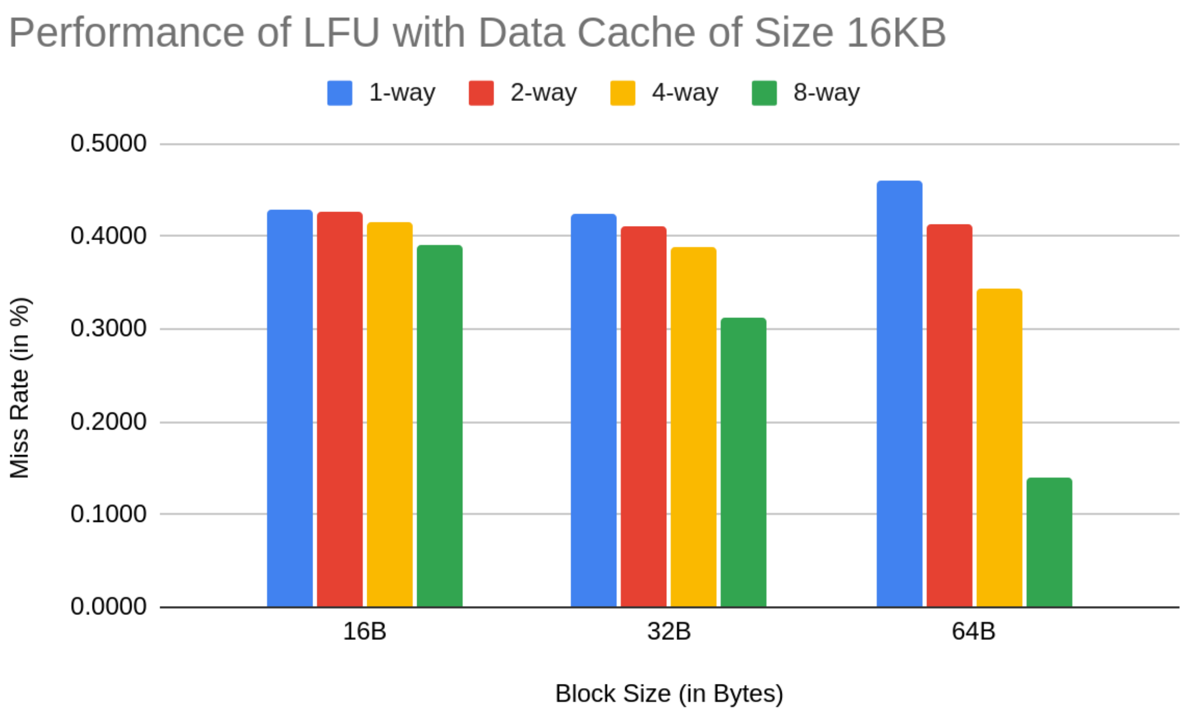

Performance of FIFO with Data Cache of Size 64KB
1-way
2-way
4-way
8-way
Miss Rate (in %)
0.4000
0.3000
0.2000
0.1000
0.0000
16B
32B
64B
Block Size (in Bytes)

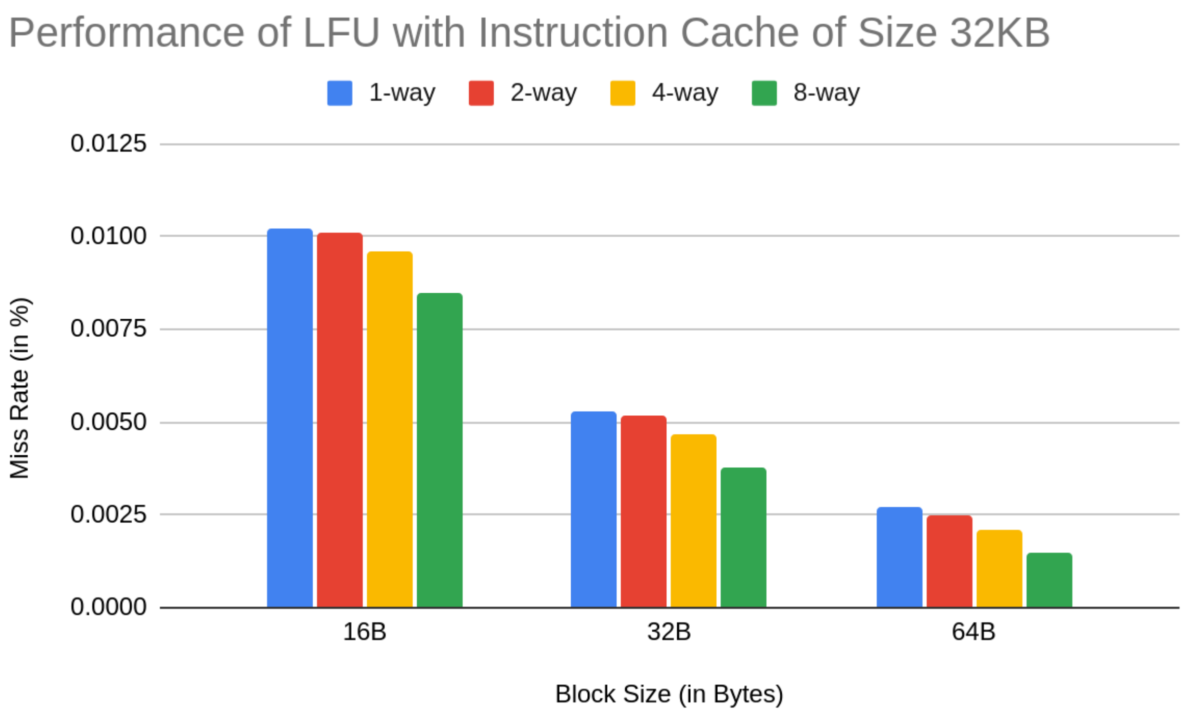
➢ The Graphs for LFU Replacement Policy



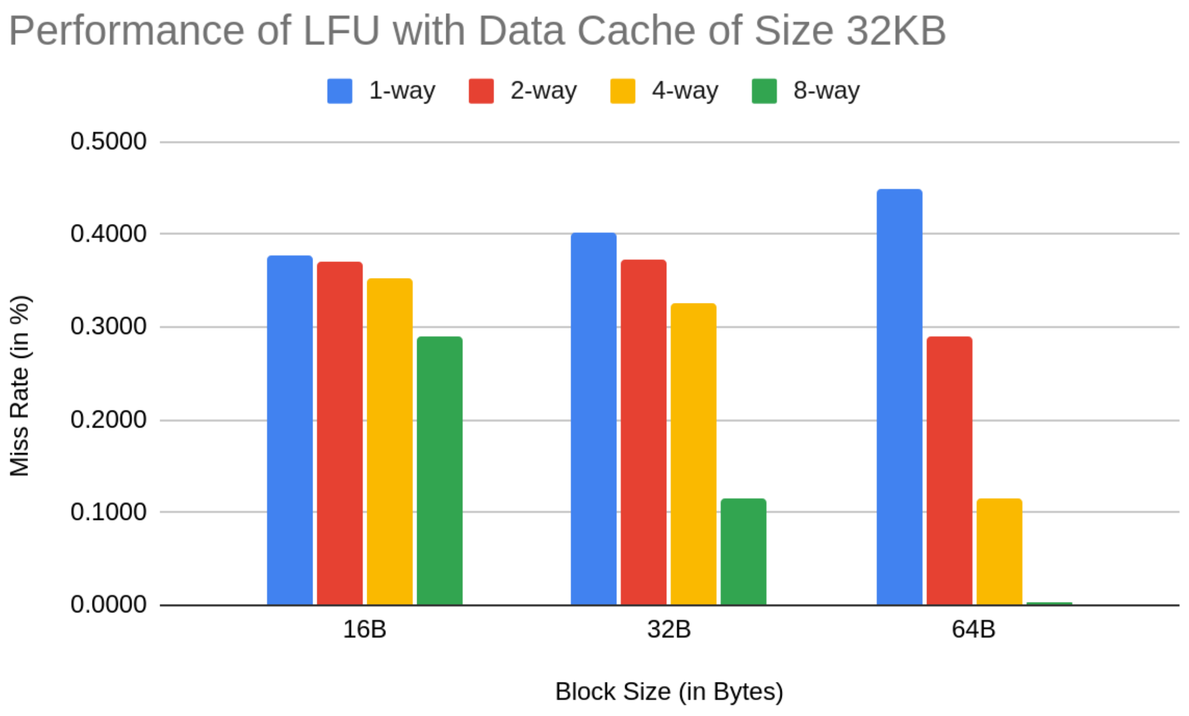



Performance of LFU with Instruction Cache of Size 16KB
1-way
2-way
4-way
8-way
0.0125
0.0100
0.0075
0.0050
0.0025
0.0000
Miss Rate (in %)
16B
32B
64B
Block Size (in Bytes)

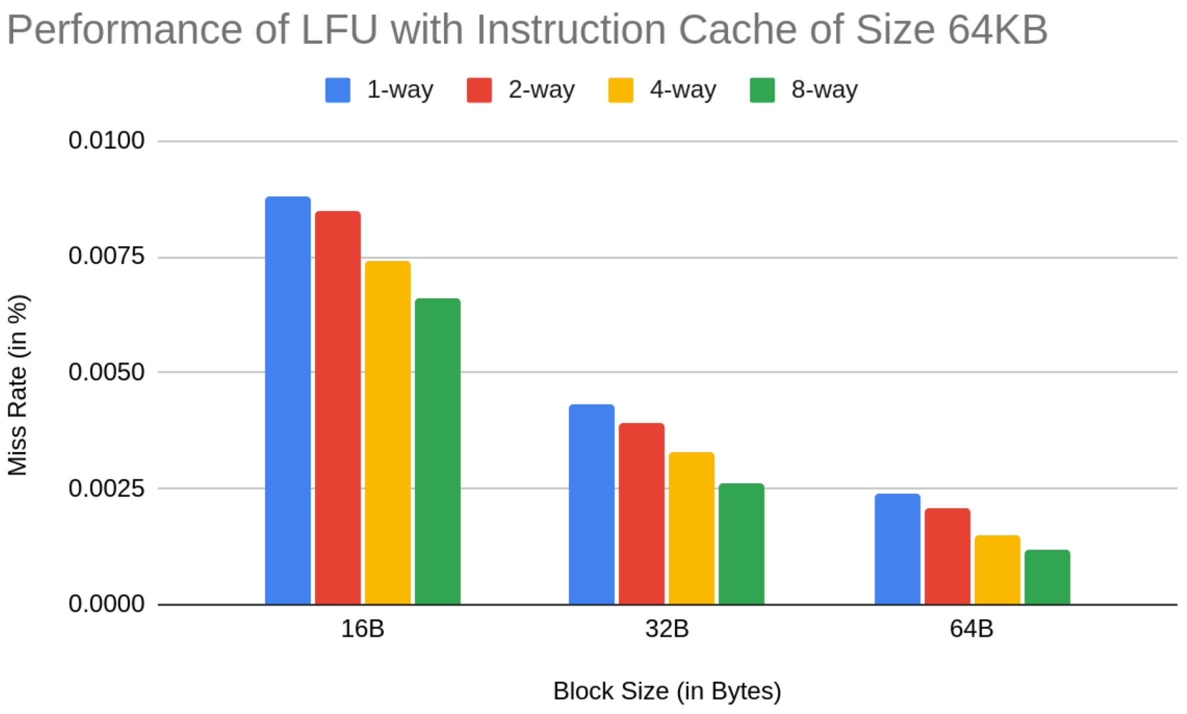

Performance of LFU with Data Cache of Size 16KB
1-way
2-way
4-way
8-way
0.5000
0.4000
0.3000
0.2000
0.1000
0.0000
Miss Rate (in %)
16B
32B
64B
Block Size (in Bytes)

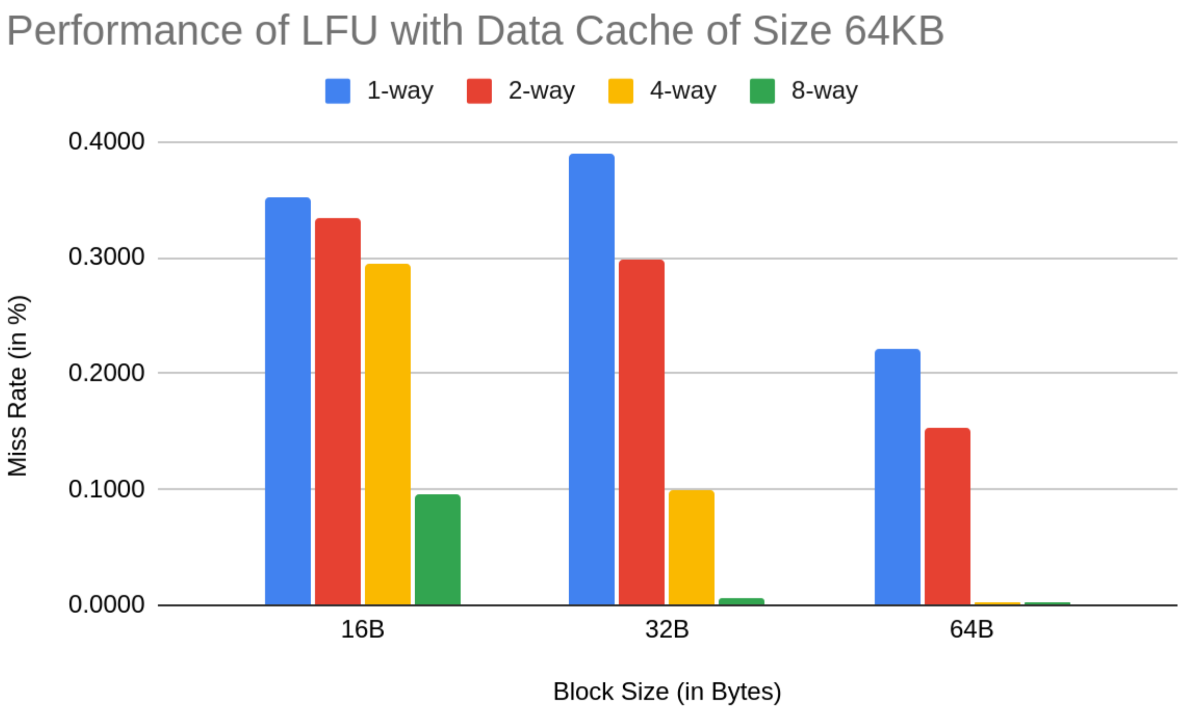


Performance of LFU with Instruction Cache of Size 32KB
1-way
2-way
4-way
8-way
Miss Rate (in %)
0.0125
0.0100
0.0075
0.0050
0.0025
0.0000
16B
32B
64B
Block Size (in Bytes)



Performance of LFU with Data Cache of Size 32KB
1-way
2-way
4-way
8-way
Miss Rate (in %)
0.5000
0.4000
0.3000
0.2000
0.1000
0.0000
16B
32B
64B
Block Size (in Bytes)


Performance of LFU with Instruction Cache of Size 64KB
1-way
2-way
4-way
8-way
Miss Rate (in %)
0.0100
0.0075
0.0050
0.0025
0.0000
16B
32B
64B
Block Size (in Bytes)


Performance of LFU with Data Cache of Size 64KB
1-way
2-way
4-way
8-way
Miss Rate (in %)
0.4000
0.3000
0.2000
0.1000
0.0000
16B
32B
64B
Block Size (in Bytes)

Results

This section presents the experimental results obtained from analyzing cache performance using the SimpleScalar **sim-cache** simulator. The goal of the study was to understand how different cache configuration parameters affect the execution efficiency of a matrix multiplication workload. The benchmark program multiplies two 32×32 matrices, producing a third matrix, which represents a common compute-intensive kernel in scientific and engineering applications.

To evaluate cache behavior, several combinations of L1 cache parameters were tested, including:

- **Cache Sizes:** 8 KB, 16 KB, 32 KB, and 64 KB
- **Block Sizes:** 16 B, 32 B, and 64 B
- **Set Associativities:** 1-way, 2-way, 4-way, and 8-way
- **Replacement Policies:** LRU, FIFO and LFU

For each configuration, the **instruction cache (I-Cache)** and **data cache (D-Cache)** miss rates were recorded. These miss rates are key indicators of memory performance, as higher miss rates lead to frequent main-memory accesses and increased execution time.

Separate analyses are provided for I-Cache and D-Cache because their locality characteristics differ significantly in matrix multiplication.

➢ LRU

- **Instruction Cache**

  a) I-Cache miss rates are very low overall.

  b) Miss rate decreases with larger cache size, larger block size, and higher associativity - but the improvements are small because the instruction stream (loop body) shows strong temporal & spatial locality.

  c) **Impact of Cache Size:** Going from 8 KB to 64 KB steadily reduces miss rate.

d) **Impact of Block Size:** Increasing block size from 16B to 32B to 64B consistently decreases miss rate across sizes and associativities.

e) **Impact of Set-Associativity:** Increasing associativity reduces misses, but with diminishing returns beyond 2–4 ways.

- **Data Cache**

a) D-Cache miss rates are **much larger and more sensitive** to configuration than I-Cache.

b) Effects are non-linear and show strong **interactions** among cache size, line size, and associativity.

c) **Impact of Cache Size:** Increasing cache size generally reduces miss rate dramatically - especially moving from 8 KB to 32 KB to 64 KB.

d) **Impact of Block Size:** Larger block sizes help when accesses are sequential. *But* for column access, larger blocks can fetch many unused bytes, increasing misses if the cache is small or low-associativity.

e) **Impact of Set-Associativity:** Higher associativity strongly reduces conflict misses.

➢ FIFO

- **Instruction Cache**

a) Instruction miss rates are small overall, but FIFO performs a bit worse than LRU in cases where recency matters.

b) **Impact of Cache Size:** Miss rate decreases as cache size increases (8KB to 64KB).

c) **Impact of Block Size:** Increasing block size (16 to 32 to 64 B) consistently reduces I-cache miss rate across sizes/ways.

d) **Impact of Set-Associativity:** Higher associativity reduces misses; improvements are present but small beyond 2–4 ways.

- **Data Cache**

a) Data cache is where FIFO shows its limitations most clearly.

b) Matmul's mixed access patterns (row-major A, column-strided B, repeated C updates) create strong demands: spatial locality for A and temporal reuse for C, but poor spatial locality for B.

c) FIFO's evictions can discard still-useful blocks and increase conflict/capacity misses.

d) **Impact of Cache Size:** Increasing cache size markedly reduces miss rate in most cases — especially moving from 8KB to 32KB to 64KB.

e) **Impact of Block Size:** Block size effect is mixed and depends on cache size & associativity. With **sufficient capacity/associativity**, larger blocks (64B) yield **lower** miss rates, but in **small/low-associativity** caches, large blocks can *increase* pollution and harm performance.

f) **Impact of Set-Associativity:** Increasing associativity consistently lowers miss rate.

➢ LFU

- **Instruction Cache**

a) **I-Cache miss rates remain very low** for all configurations because the loop instructions are reused frequently, ie, LFU keeps the most frequently accessed instructions in cache.

b) **Impact of Cache Size:** Increasing size from 8 KB to 64 KB slightly decreases miss rate, but improvements are small since I-cache already performs well.

c) **Impact of Block Size:** Increasing block size from 16B to 32B to 64B improves hit rate by bringing more useful instructions per fetch.

d) **Impact of Set-Associativity:** Higher associativity (4-way and 8-way) helps reduce conflict misses, but improvements are minor due to already high locality.

- **Data Cache**

a) **D-Cache miss rates are much higher than I-Cache**, especially with small size and low associativity, due to the scattered access pattern of matrix B.

b) LFU improves performance more effectively than FIFO because frequently reused blocks (mainly from matrix C & A rows) are retained.

c) **Impact of Cache Size:** Increasing size significantly reduces capacity misses - especially moving from 8 KB to 32 KB to 64 KB.

d) **Impact of Block Size:** Larger block sizes help only if cache size is also large. If cache is small (8 KB), big blocks cause **cache pollution** and useless data fetches.

e) **Impact of Set-Associativity:** Higher associativity strongly reduces conflict misses. Best performance is seen with 8-way, 64 KB, 64B block size, which gives near-ideal miss rate.

Conclusion

This mini-project evaluated the impact of cache hierarchy parameters in the SimpleScalar architecture using a matrix multiplication benchmark. By varying cache size, block size, associativity, and replacement policies (LRU, FIFO and LFU), clear insights were obtained on both instruction and data cache behavior.

Instruction cache performance remained excellent across all configurations due to strong locality in program loops, resulting in very low miss rates regardless of size or associativity. All three replacement policies (LRU, FIFO, and LFU) performed almost similar for instruction accesses with minimal differences.

Data cache performance showed significant variation. Matrix multiplication exhibits:

- Good spatial locality for matrix **A** access.
- Poor spatial locality for matrix **B** column access.
- Strong temporal locality for matrix **C** updates.

Because of this mixed behavior, the D-cache experienced much higher miss rates than the I-cache. Increasing cache size and associativity substantially reduced capacity and conflict misses. Larger block sizes improved performance mainly when the cache was sufficiently large; otherwise, they could introduce cache pollution.

Among replacement policies:

- **LRU** consistently gave the best results.
- **LFU** performed close to LRU, particularly in larger caches where reuse patterns are stable.
- **FIFO** suffered from unnecessary evictions, leading to higher miss rates.

Overall, the most balanced performance was observed with:

- **D-Cache:** 32 KB, 4-way, 64-byte block size (**LRU preferred**, LFU close second).
- **I-Cache:** 16–32 KB, 2–4-way, 32–64-byte block size.

In summary, this study highlights that cache configuration has a major influence on execution efficiency, particularly for data-intensive workloads, reinforcing key architectural principles in ACA by showing how locality, associativity, capacity, and replacement policy collectively influence memory hierarchy efficiency.

Appendix

1. Source Code Repository

The complete matrix multiplication benchmark (matmul.c) and supporting files used in this project are available at:

https://github.com/25cs06002/ACA-Lab-Project

This repository includes:

- C source code for Matrix Multiplication Benchmark
- Compilation instructions using SimpleScalar cross-compiler
- Shell scripts / command files to automate cache simulation runs
- Output logs of all tested cache configurations

2. Cross Compiler Used

All executables in this project were built using the SimpleScalar PISA cross-compiler:

/home/student/Desktop/simplescalar/sslittle-na-sstrix/bin/gcc

Example compilation command:

/home/student/Desktop/simplescalar/sslittle-na-sstrix/bin/gcc -O2 matmul.c -o matmul.pisa

3. Simulation Tools Used

All simulations were executed using:

../simplesim-3.0/sim-cache

4. Directory Structure

```
simplescalar_project/
|
├── f2c-1994.09.27
├──gcc-2.7.2.3
├──glibc-1.09
├──results
│    ├── lru
│    │    ├── output_dl1_16k_b32_a2.txt
│    │    ├── output_dl1_32k_b64_a4.txt
│    │    ├── …
│    ├── fifo
│    │    ├── output_dl1_16k_b32_a2.txt
│    │    ├── output_dl1_32k_b64_a4.txt
│    │    ├── …
│    ├── lfu
│    │    ├── output_dl1_16k_b32_a2.txt
│    │    ├── output_dl1_32k_b64_a4.txt
│    │    ├── …
│    └── matmul.c
|
├──simplesim-3.0
├──simpleutils-990811
├──ssbig-na-sstrix
└──sslittle-na-sstrix
```