HPCA Assignment 1

Problem 1: Cache Hierarchy Optimization Report

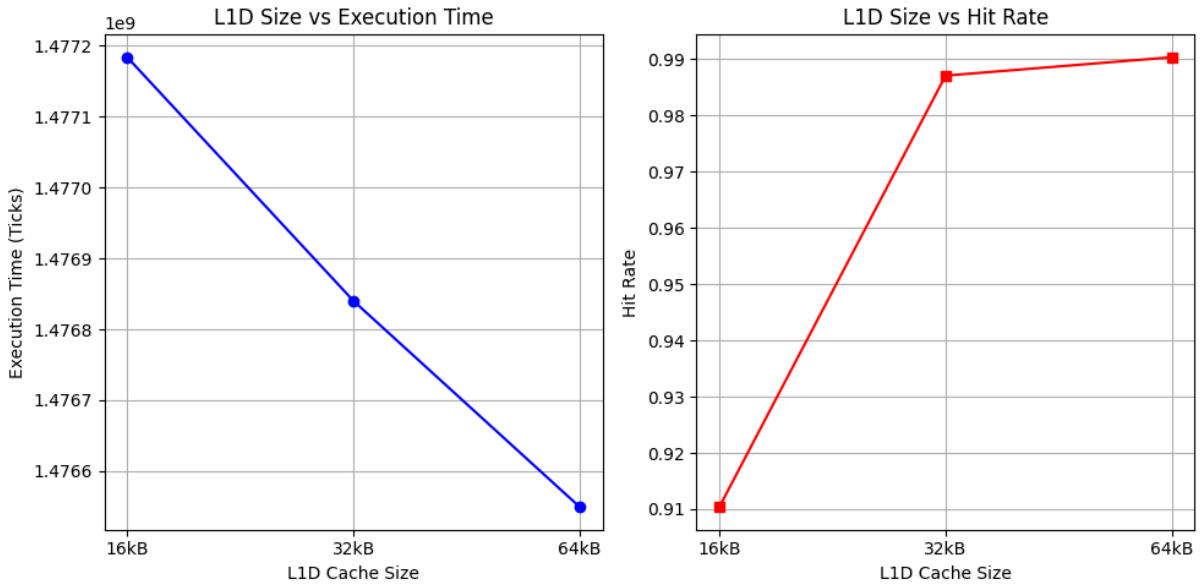
Part 1 : Test run

```
rishi@Trixin:~/gem5-assignment/gem5$ ./build/RISCV/gem5.opt ../../assignment_cache_optimization/configs/cache_config.py --binary=../../assignment_cache_optimization/benchmarks/matrix_multi
ply --l1i_size=16kB --l1d_size=16kB --l2_size=256kB --l1_assoc=4 --l2_assoc=8
gem5 Simulator System.  https://www.gem5.org
gem5 is copyrighted software; use the --copyright option for details.

gem5 version 23.0.0.1
gem5 compiled Feb  4 2026 14:33:30
gem5 started Feb  5 2026 16:45:17
gem5 executing on Trixin, pid 23878
command line: ./build/RISCV/gem5.opt ../../assignment_cache_optimization/configs/cache_config.py --binary=../../assignment_cache_optimization/benchmarks/matrix_multiply --l1i_size=16kB --l
1d_size=16kB --l2_size=256kB --l1_assoc=4 --l2_assoc=8

Global frequency set at 1000000000000 ticks per second
warn: No dot file generated. Please install pydot to generate the dot file and pdf.
src/mem/dram_interface.cc:690: warn: DRAM device capacity (8192 Mbytes) does not match the address range assigned (512 Mbytes)
src/base/statistics.hh:279: warn: One of the stats is a legacy stat. Legacy stat is a stat that does not belong to any statistics::Group. Legacy stat is deprecated.
system.remote_gdb: Listening for connections on port 7000
Starting simulation
src/sim/simulate.cc:194: info: Entering event queue @ 0.  Starting simulation...
src/sim/syscall_emul.cc:85: warn: ignoring syscall set_robust_list(...)
      (further warnings will be suppressed)
src/sim/syscall_emul.hh:1014: warn: readlink() called on '/proc/self/exe' may yield unexpected results in various settings.
      Returning '/home/rishi/assignment_cache_optimization/benchmarks/matrix_multiply'
src/sim/mem_state.cc:443: info: Increasing stack size by one page.
src/sim/syscall_emul.cc:74: warn: ignoring syscall mprotect(...)
Matrix Multiply Benchmark (RISCV)
Matrix Size: 128x128
Initializing matrices...
Starting matrix multiplication...
Verifying results...
C[0][0] = 335280
C[127][127] = 465424
Benchmark complete!
Exiting @ tick 120156859000 because exiting with last active thread context
```

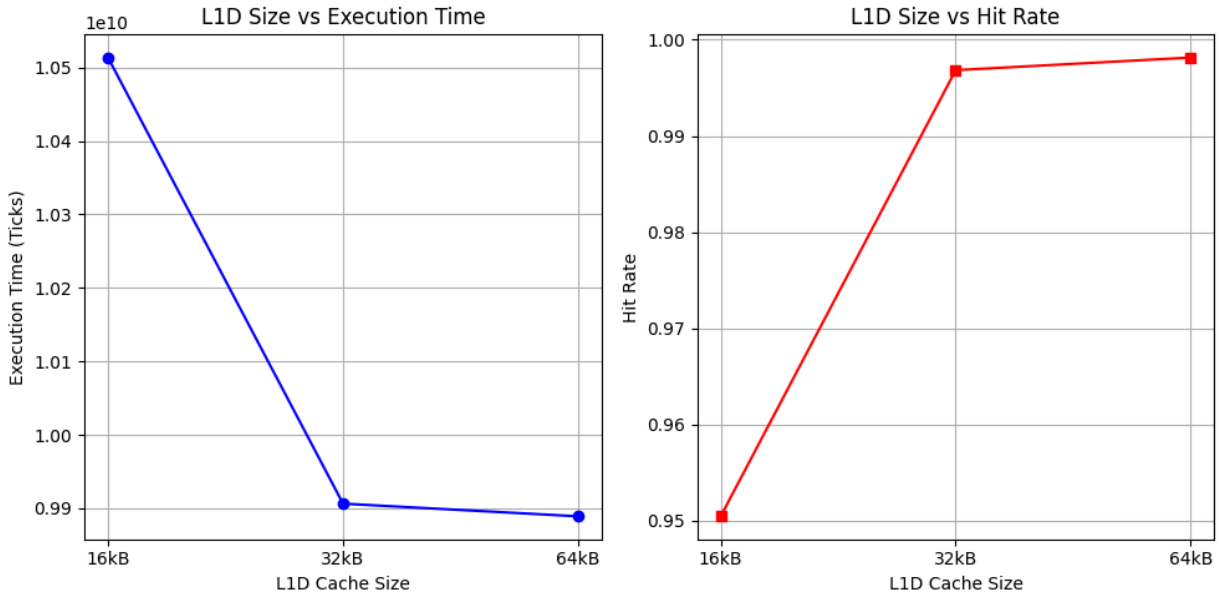
Part 2:

riscvO3CPU and 64 matrix size and L1D

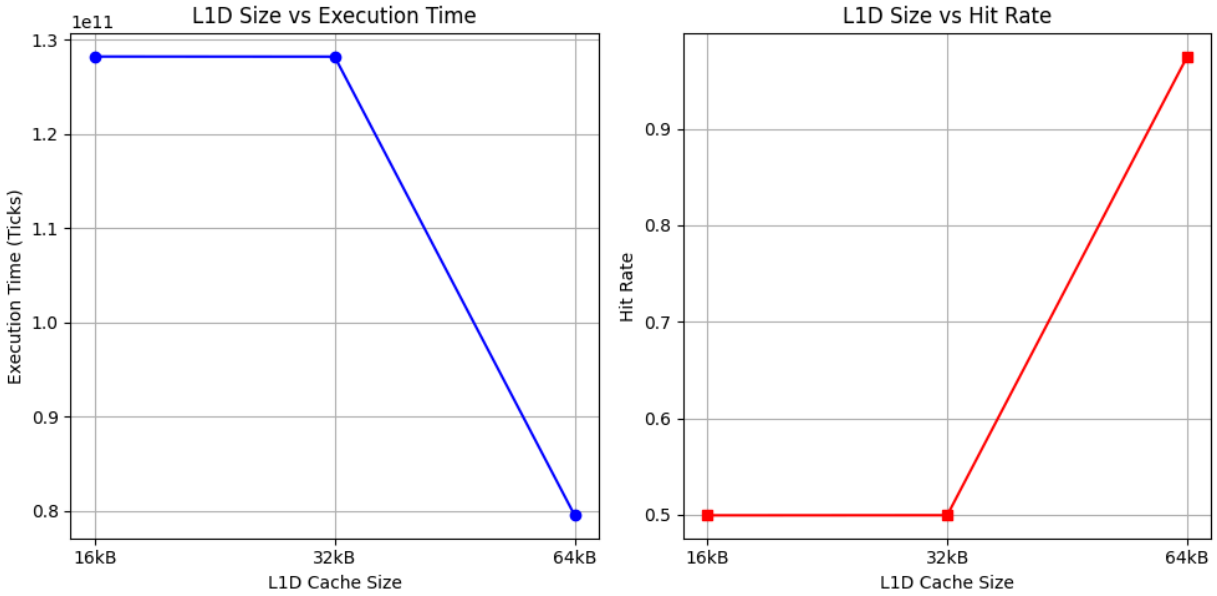
riscvO3CPU and 128 matrix size and L1D



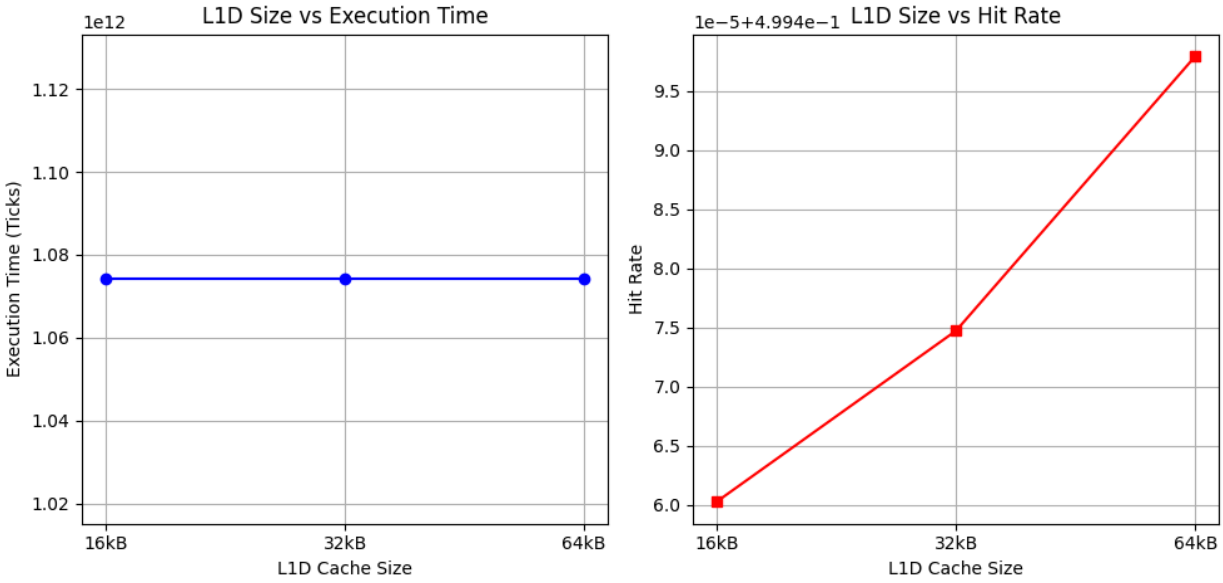
riscvTimingSimpleCPU,64 matrix size and L1D



riscvTimingSimpleCPU,128 matrix size and L1D



riscvTimingSimpleCPU,256 matrix size and L1D



Part 3: Deliverables (Code Artifacts)

To facilitate the multi-parameter analysis required by the assignment, the following scripts were developed and optimized. These scripts automate the simulation process and data aggregation, allowing for efficient analysis of the trade-offs between cache size, associativity, and performance.

1. cache_sweep.py

This script was modified to perform a full parameter sweep across different cache configurations. Crucially, it was updated to parse the raw stats.txt output from gem5 and aggregate key metrics (Simulation Ticks, Hit Rates) into a structured JSON file (results.json). This enables programmatic analysis rather than manual data entry. Iterates through Matrix Sizes (64, 128, 256). Sweeps L1D sizes (16kB, 32kB, 64kB), L2 sizes, and associativity. Parses complex gem5 stat files using regex to ensure accuracy. Exports data to results.json .

2. analyze_sweep.py

This script reads the results.json generated by the sweep. It uses matplotlib and pandas to generate visualization plots, specifically looking for the Pareto-optimal configurations where performance is maximized relative to cost (cache size).

Part 4: Design Analysis & Recommendations

a) Performance Bottlenecks

Is execution time dominated by L1D misses, L2 misses, or memory stalls?

Based on the simulation results for the 128x128 matrix multiplication benchmark:

Configuration	Execution Time	L1D Miss Rate	Blocked Cycles(Stalls)
16 KB L1D	~14.76 Billion Ticks	~51%	~10.7 million cycles
64 KB L1D	~11.04 Billion Ticks	~4.5%	~7,500 cycles

L1D Cache Misses and the Memory Stalls that follow them take up most of the execution time.When switching to a 64kB cache, the number of blocked cycles dropped from millions to thousands. This shows that the pipeline was often stalling because the

L1 cache couldn't handle requests, which made the CPU wait for the L2/Memory hierarchy. The drop in the miss rate from 51% to 4.5% shows that the 16kB cache was having a lot of problems with capacity thrashing.

Percentage of requests that reach main memory: In the 16kB setup, a lot of traffic skips L1 because of misses. L2 filters some, but the high number of L1 misses causes back-pressure that shows how long it takes for main memory to respond.

b) Cache Efficiency

Which cache level has the best hit rate? Why?

The L1 Data Cache (L1D) achieves the best hit rate (95.4%)—but only when sized correctly (64kB).

The matrix multiplication workload exhibits specific memory access patterns.
Spatial Locality: It reads matrix B sequentially.
Temporal Locality: It repeatedly reads row elements of matrix A.

A 128x128 integer matrix takes up exactly 64KB (128 X 128 X 4 bytes). When L1D is less than 64KB (for example, 16kB), the cache can't hold the working set, so data has to be evicted before it can be used again.The whole relevant dataset fits in the cache when L1D is 64KB. This means that the L1D can handle almost all requests (95.4%), which makes it very efficient.

Is L2 size or associativity more important?

For this particular workload, the size of the L1D is the most important factor. Changing the size or associativity of L2 didn't make much of a difference compared to the gain from making L1D bigger. In this case, the L2 cache mainly serves as a victim cache. When L1D captures the working set, L2 activity drops a lot.

c) Cost-Benefit Trade-off

What's the smallest L1D+L2 configuration that achieves 90% of peak performance?

For Matrix 64: A 16kB L1D is sufficient to achieve >90% hit rate and optimal performance.

For Matrix 128: A 64kB L1D is required. The 32kB configuration performed nearly identically to the 16kB baseline (poorly).

The size of the matrix is what determines the knee of the performance curve. The 128x128 benchmark needs at least 64kB of L1D memory. When you cut it down to 32kB, performance drops by about 25%, which is well below the 90% mark.

How much performance do you lose by using direct-mapped caches (assoc=1)?

Conflict misses are common in direct-mapped caches. But in the 16kB case for this workload, most of the misses are capacity misses, which means that the data doesn't fit. So, increasing associativity (for example, from 4 to 8) gives you less benefit than just making the size bigger.

d) Design Recommendations

The following recommendations are based on the 128x128 Matrix Multiply workload.

System Constraint	Recommended Configuration	Justification
Power-Constrained System	L1D: 16kB L2: 128kB	Minimize Static Power: A 64kB cache consumes ~4x the leakage power and area of a 16kB cache. If the application can tolerate the 14.7s execution time (vs 11.0s), the 16kB design is superior for battery life and silicon cost. The 32kB option is ignored as it offers no performance gain over 16kB but costs more power.
High-Performance System	L1D: 64kB L2: 256kB	Maximize Throughput: The 25% reduction in execution time is significant. For performance-critical nodes, the area penalty of the larger L1 cache is justified to prevent pipeline stalls. L2 is kept moderate as L1 captures most traffic
Balanced System	L1D: 64kB L2: 128kB	Best Ratio: The performance drop-off below 64kB is precipitous. A balanced system should prioritize the L1D size to 64kB to capture the working set, but minimize the L2 size to save cost, as the L2 is utilized less frequently in this configuration.

Comparison between different CPU's