Assignment 2

Cache Performance Analysis using GEM-5

Submitted By : Manish Kumar(2023MCS2497)

Submitted On : **6th October, 2024**

DEPARTMENT OF COMPUTER SCIENCE & TECHNOLOGY
Indian Institute of Technology, Delhi
Hauz Khas, New Delhi, 110016

Academic Year: 2024-25

Problem Statement:

- **Implementation of Matrix Multiplication**: Perform matrix multiplication on relatively small input sizes.

- **Simulation Using gem5**: Run the implemented program on the gem5 simulator, varying the following hardware parameters:
  - **CPU Models**: Test the program across different CPU models available in gem5.
  - **L1/L2 cache parameters**: Evaluate performance by varying different parameters for L1 Instruction & Data, L2 cache as mentioned in

    > gem5/configs/learning_gem5/part1/caches.py

  - **CPU Clock Frequency:** (fixed) 2GHz.
  - **Memory Configurations**: (fixed) DDR4_2400_8x8.

- **Performance Analysis**: Describe and analyze the performance statistics (e.g., difference in execution times & Cache Miss rate) resulting from the different CPU models & varying cache sizes & latencies .

Simulation Parameters:

- **CPU Models**:. DerivO3CPU, TimingSimpleCPU

- **3 Matrix Multiplication algo.:**
  - **simple MM implementation (ijk)**
  - **reordered loops arrangement (ikj)**

    (to verify cache access pattern)

  - **blocked MM with block size of 15 over matrices of 75x75**

- **Different set of Passes of various characteristic evaluation**:

  Pass 1-2 for the effect of MM algo., *(on all 3 binaries)*

  When ran over TimingSimpleCPU vs DerivO3CPU

  Pass 3-5 for effect of cache size *(on mm_blocked, DerivO3CPU)*

  Changing sizes of L1/L2 caches, or change in CPU with similar cache size

  Pass 6-8 for effect of cache Associativity *(on mm_blocked, DerivO3CPU)*

  Changing associativity of L1/L2 caches

  Pass 9-10 for effect of cache latency *(on mm_blocked, DerivO3CPU)*

  Changing latency during of L1/L2 caches

Evaluation:

- ***Simple MM** (ijk)* :

```
void multiplyMatrices(int A[N][N], int B[N][N], int res[N][N]) {
    for(int i = 0; i < N; i++) {
        for(int j = 0; j < N; j++) {
            res[i][j] = 0;              // Initialize value to 0

            for(int k = 0; k < N; k++)          // iterate over elements in selected R & C
                res[i][j] += A[i][k] * B[k][j];         // compute pdt sum
        }
    }
}
```

- ***Rearranged Loops MM** (ikj)* :

```
void multiplyMatrices(int A[N][N], int B[N][N], int res[N][N]) {
    for(int i = 0; i < N; i++) {
        for(int j = 0; j < N; j++)
            res[i][j] = 0;              // Initialize value to 0
    }

    for(int i = 0; i < N; i++) {
        for(int k = 0; k < N; k++) {        // iterate over elements in selected R & C
            for(int j = 0; j < N; j++)                  // reordered loop
                res[i][j] += A[i][k] * B[k][j];         // accumulate partial sums
        }
    }
}
```

- ***Blocked MM** (with block_size 15)*:

```
void multiplyMatrices(int A[N][N], int B[N][N], int res[N][N]) {
    for(int i = 0; i < N; i++) {
        for(int j = 0; j < N; j++)
            res[i][j] = 0;              // Initialize value to 0
    }

    // Blocked Matrix multiplication
    for(int ii = 0; ii < N; ii += BLOCK_SIZE) {
        for(int jj = 0; jj < N; jj += BLOCK_SIZE) {
            for(int kk = 0; kk < N; kk += BLOCK_SIZE) {
                for(int i = ii; i < ii + BLOCK_SIZE && i < N; i++) {
                    for(int j = jj; j < jj + BLOCK_SIZE && j < N; j++) {
                        for(int k = kk; k < kk + BLOCK_SIZE && k < N; k++)
                            res[i][j] += A[i][k] * B[k][j];
                    }
                }
            }
        }
    }
}
```

- ***Simulation results*** **:**

Following parameters were acquired on running the simulation on both DerivO3CPU & TimingSimpleCPU using various branch prediction methods:

Pass Count	Configuration	simSeconds	cpi	L1icache_MissRate	L1dcache_MissRate	L2_MissRate
Pass 1	TimingSimpleCPU_16kB_32kB_2_2_256kB_8_20_mm_simple.1	0.0304	3.698375	0.000025	0.00047	0.425148
Pass 1	TimingSimpleCPU_16kB_32kB_2_2_256kB_8_20_mm_rearranged.1	0.029163	3.544177	0.000027	0.000528	0.382371
Pass 1	TimingSimpleCPU_16kB_32kB_2_2_256kB_8_20_mm_blocked.1	0.033234	3.587809	0.000024	0.00045	0.39496
Pass 2	DerivO3CPU_16kB_32kB_2_2_256kB_8_20_mm_simple.1	0.004365	0.531065	0.000619	0.000494	0.563453
Pass 2	DerivO3CPU_16kB_32kB_2_2_256kB_8_20_mm_rearranged.1	0.004404	0.535194	0.000637	0.000937	0.439172
Pass 2	DerivO3CPU_16kB_32kB_2_2_256kB_8_20_mm_blocked.1	0.007608	0.82137	0.000562	0.000543	0.426286
Pass 3	TimingSimpleCPU_64kB_128kB_2_2_256kB_8_20_mm_blocked.1	0.0332	3.584161	0.000023	0.000144	0.999464
Pass 4	DerivO3CPU_64kB_128kB_2_2_256kB_8_20_mm_blocked.1	0.007574	0.81771	0.000519	0.000189	1
Pass 5	TimingSimpleCPU_64kB_128kB_2_2_1MB_8_20_mm_blocked.1	0.0332	3.584161	0.000023	0.000144	0.999464
Pass 6	DerivO3CPU_64kB_128kB_4_2_256kB_8_20_mm_blocked.1	0.007574	0.81771	0.000519	0.000189	1
Pass 7	DerivO3CPU_64kB_128kB_8_2_256kB_8_20_mm_blocked.1	0.007574	0.81771	0.000519	0.000189	1
Pass 8	DerivO3CPU_64kB_128kB_8_2_256kB_16_20_mm_blocked.1	0.007574	0.81771	0.000519	0.000189	1
Pass 9	DerivO3CPU_64kB_128kB_8_2_256kB_16_10_mm_blocked.1	0.007566	0.816769	0.000517	0.000189	1
Pass 10	DerivO3CPU_64kB_128kB_8_4_256kB_16_20_mm_blocked.1	0.008977	0.969131	0.000492	0.000188	1

The configurations examined included micro-architectures both TimingSimpleCPU and DerivO3CPU, with varying sizes for L1(i & d) and L2 caches, associativity levels, and tag latencies.

Simulation Time

The configurations utilizing the DerivO3CPU architecture demonstrated significantly lower **simulation times** compared to the TimingSimpleCPU setups. For example, the configuration DerivO3CPU_16kB_32kB_2_2_256kB_8_20_mm_simple.1 achieved the fastest simulation time of 0.004365 seconds, highlighting its efficiency.

In contrast, the TimingSimpleCPU_16kB_32kB_2_2_256kB_8_20_mm_simple.1 configuration had a longer simulation time of 0.0304 seconds, indicating that the DerivO3CPU is more suitable for this type of workload.

Cycles per Instruction (CPI):

The **CPI values** reflect the efficiency of each configuration in terms of the number of cycles needed per instruction executed. The lowest CPI recorded was 0.531065 for DerivO3CPU_16kB_32kB_2_2_256kB_8_20_mm_simple.1, indicating high performance.

Conversely, the TimingSimpleCPU configurations exhibited higher CPI values, with the highest recorded CPI of 3.698375 for TimingSimpleCPU_16kB_32kB_2_2_256kB_8_20_mm_simple.1. This substantial difference underscores the efficiency of the DerivO3CPU in executing instructions.

Cache Miss Rates:

The **L1 instruction cache overall miss rate**s were low across all configurations, with the TimingSimpleCPU achieving a miss rate of 0.000025 in its best configuration, TimingSimpleCPU_16kB_32kB_2_2_256kB_8_20_mm_simple.1. This indicates effective caching for instruction retrieval.

For the **L1 data cache**, the miss rates were relatively low as well, with a minimum miss rate of 0.000189 in several configurations, showing good performance.

The **L2 cache miss rates** varied more significantly. The best-performing L2 cache miss rate was recorded at 0.39496 for the configuration TimingSimpleCPU_16kB_32kB_2_2_256kB_8_20_mm_blocked.1, while multiple configurations showed a concerning miss rate of 1.0, indicating no hits, which suggests potential inefficiencies in how data is cached at this level.

- ***TimingSimpleCPU*** **(simplified, in-order non-pipelined CPU model) :**

Simulation Time : Varied from 0.029 to 0.033 for various MM algo.

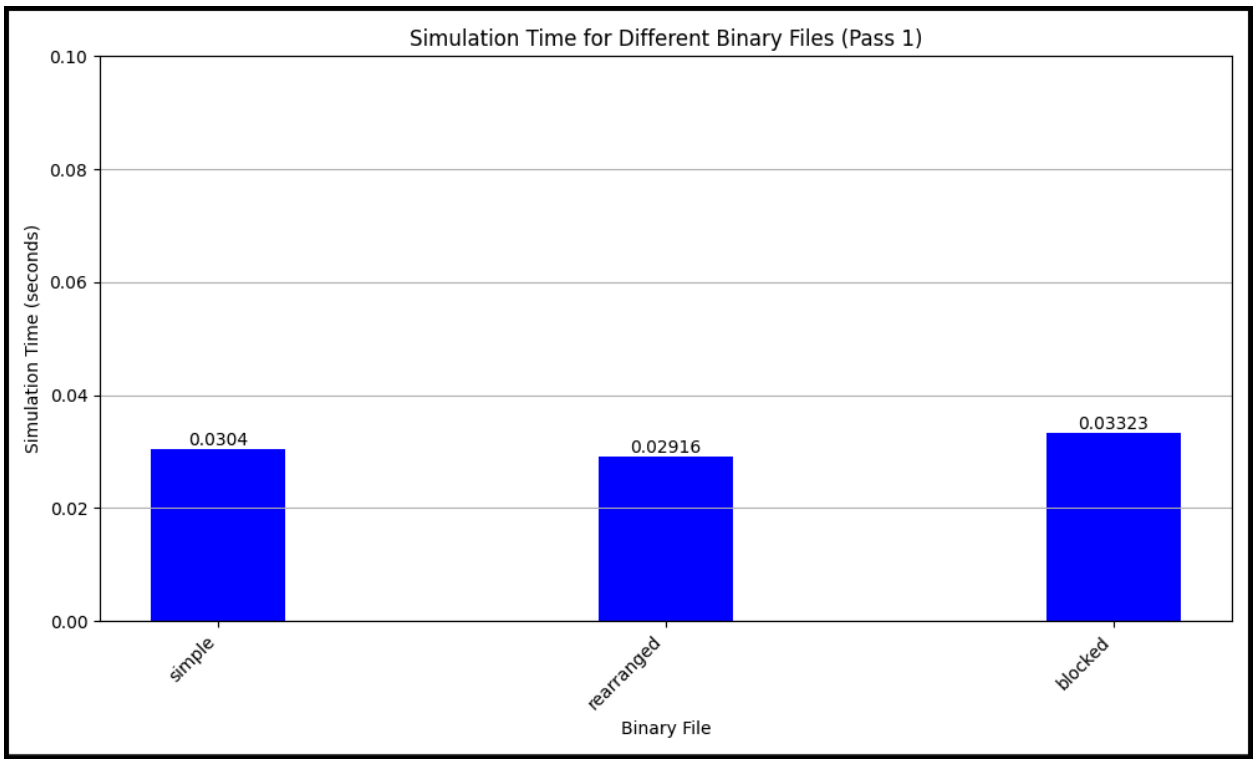
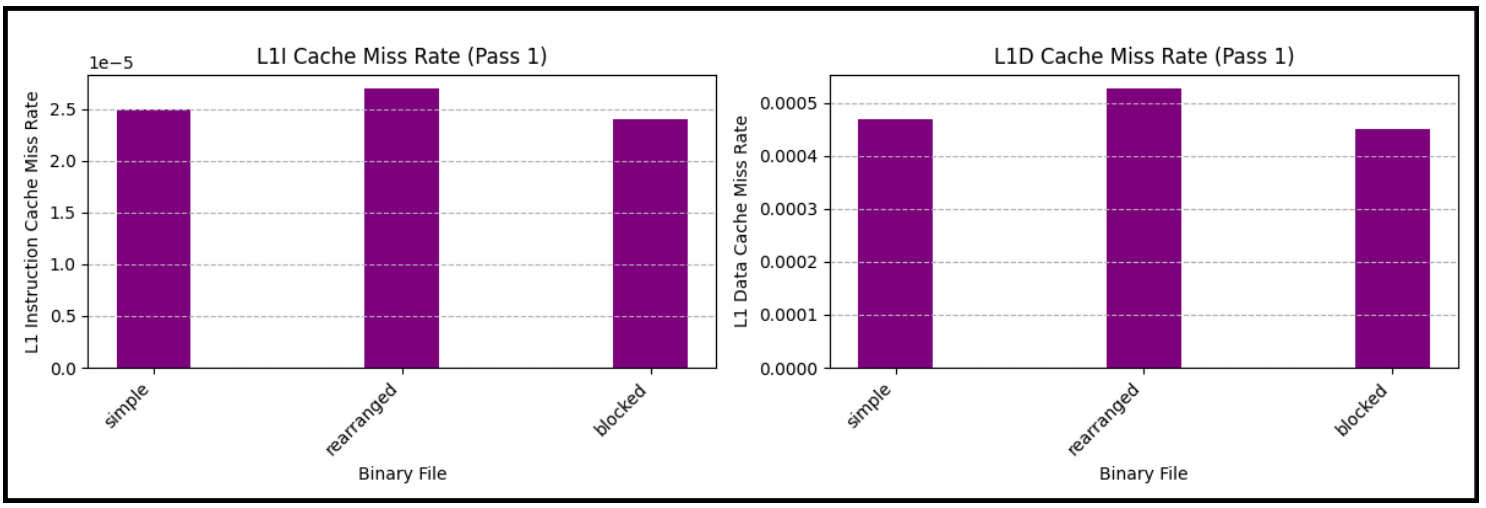
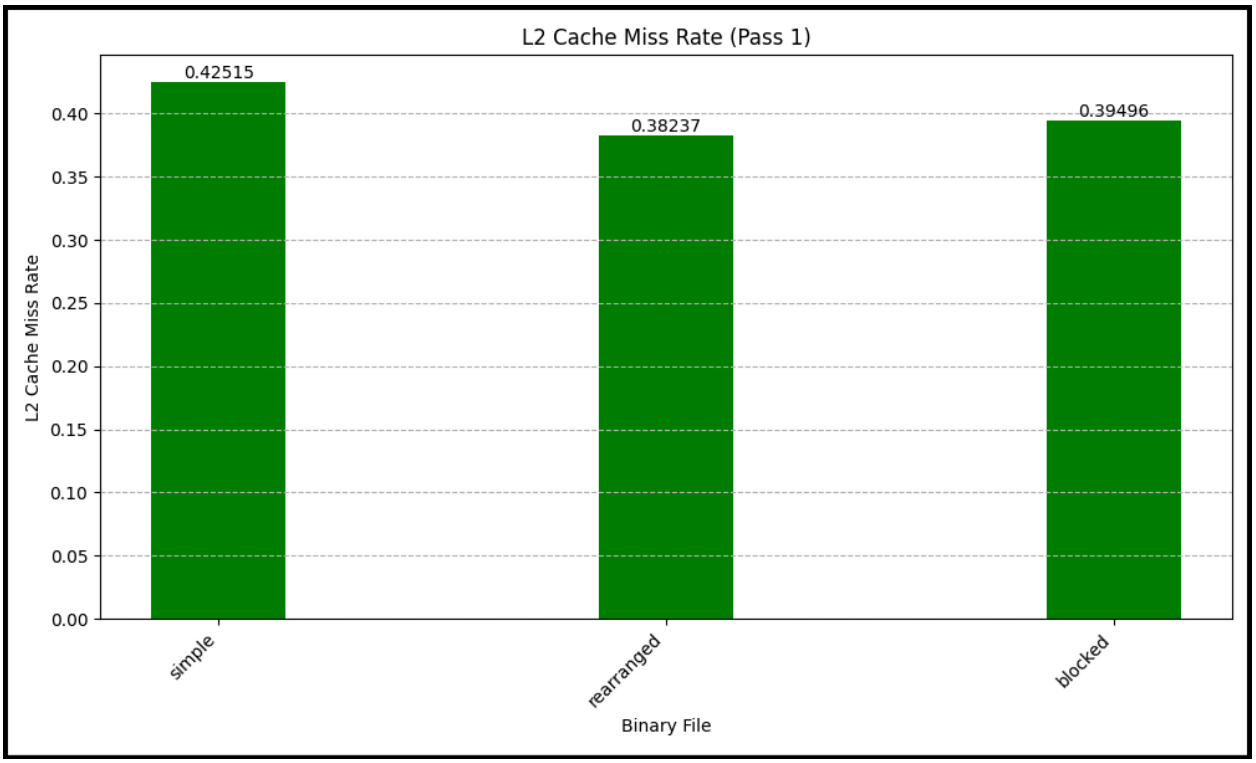


Fig. PASS 1 : Simulation of various MM algo. On TimingSimpleCPU

(16Kb L1i cache, 32Kb L1d cache, 256Kb L2 cache)





Miss Rates :

As expected, we see a maximum number of misses when we use the loop reordered MM algo.

Very low miss rates (0.000024 to 0.000027) across all configurations, indicating efficient instruction fetching.

Slightly higher miss rates (0.00045 to 0.000528), with mm_rearranged showing the highest at 0.000528, suggesting less efficient data access.

Higher miss rates (0.382371 to 0.425148), especially for mm_simple (0.425148), indicate that data locality may be an issue and the working set exceeds L1 cache capacity.

- ***DerivO3CPU*** **(detailed, out-of-order pipelined CPU model) :**

Simulation Time : Varied from 0.004 to 0.007 for various MM algo. (relatively much less than TimingSimpleCPU).

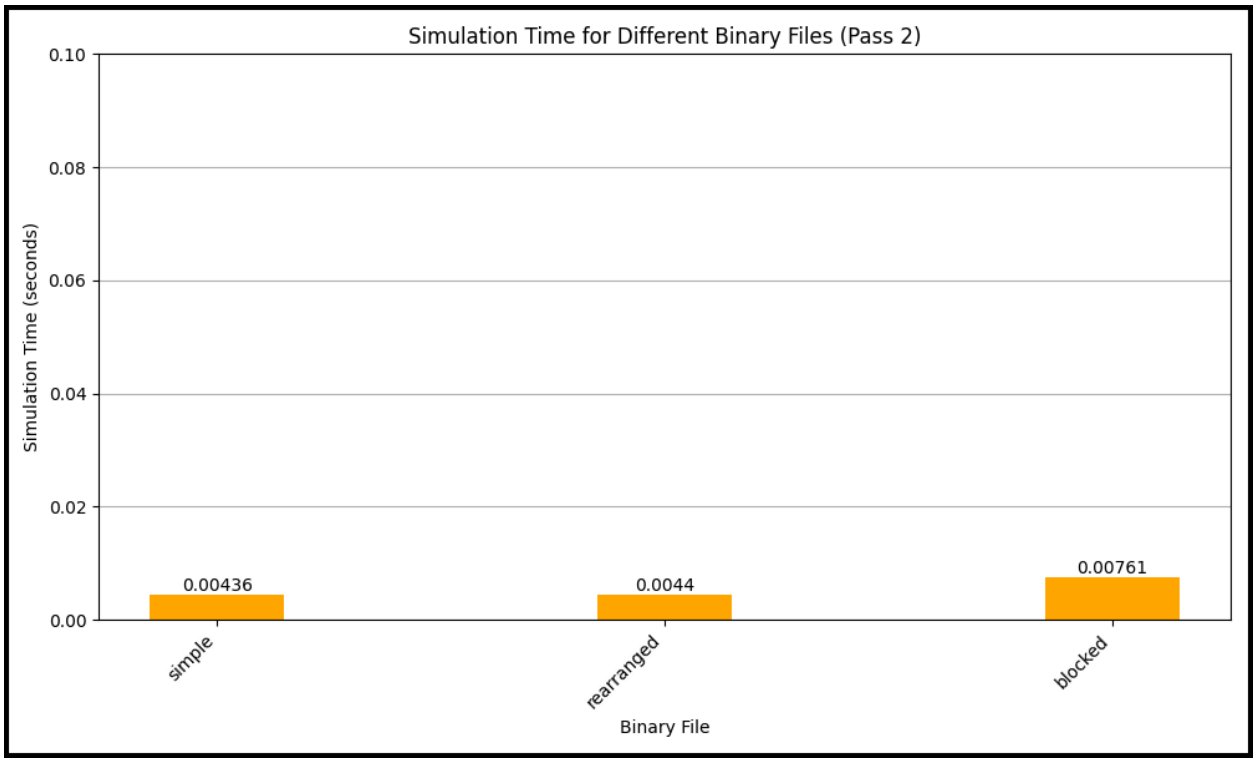
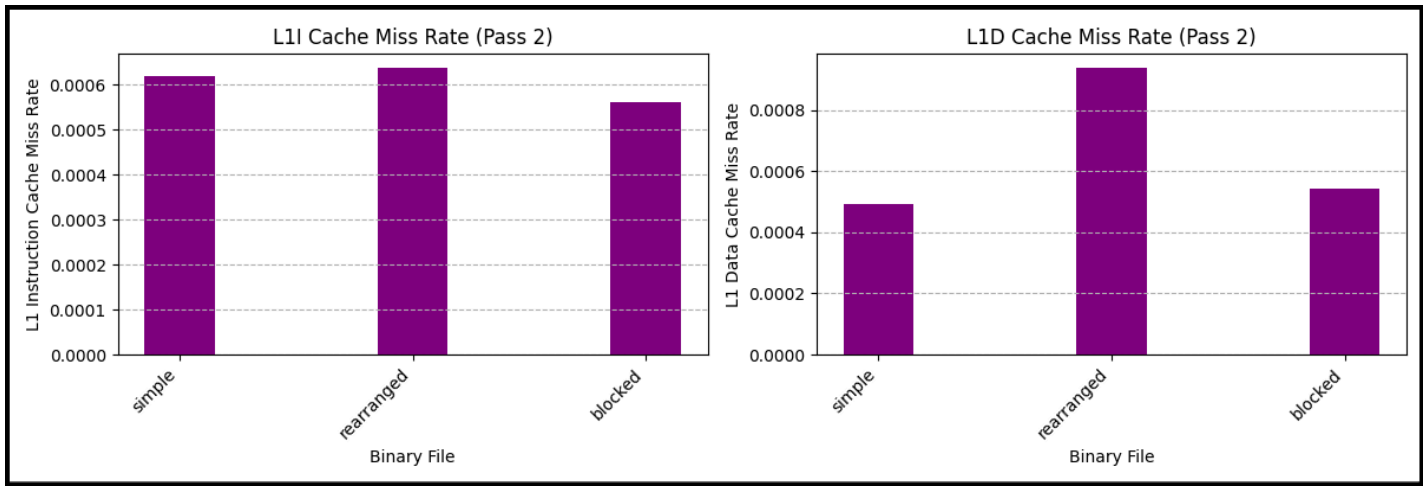
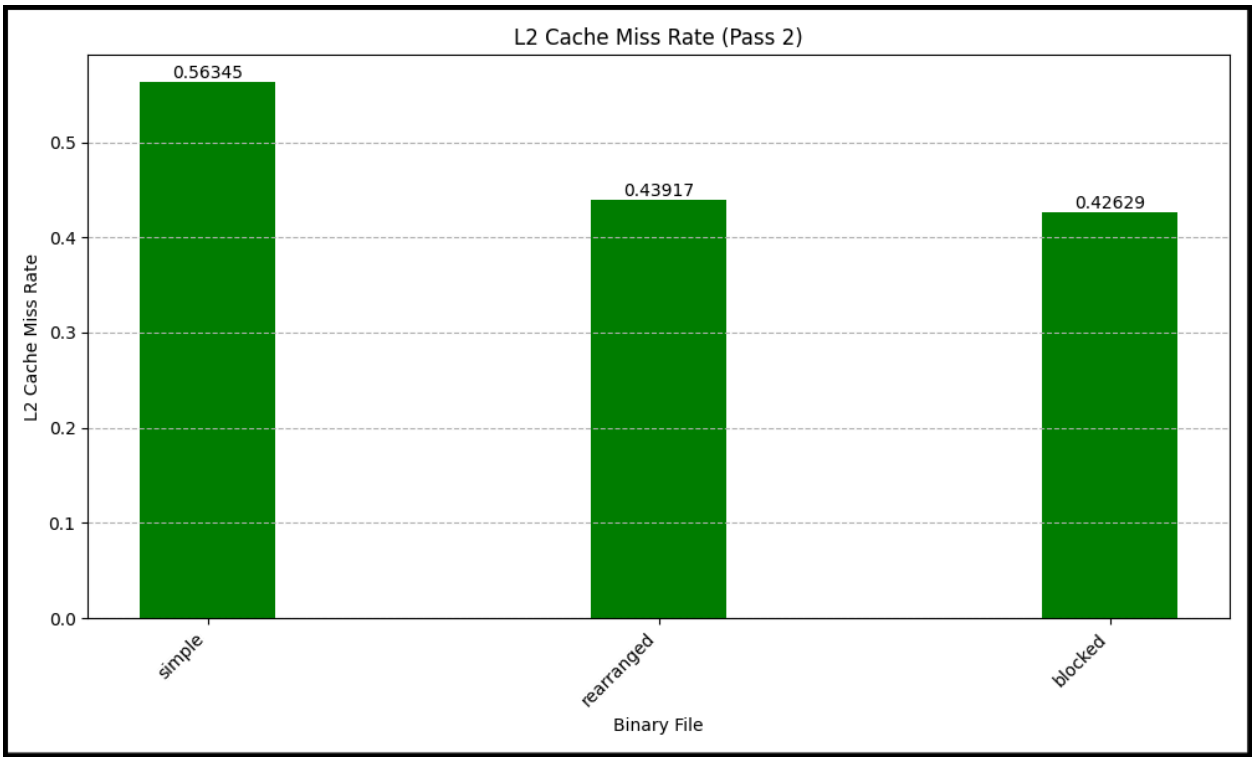


Fig. PASS 2 : Simulation of various MM algo. On DerivO3CPU

(16Kb L1i cache, 32Kb L1d cache, 256Kb L2 cache)





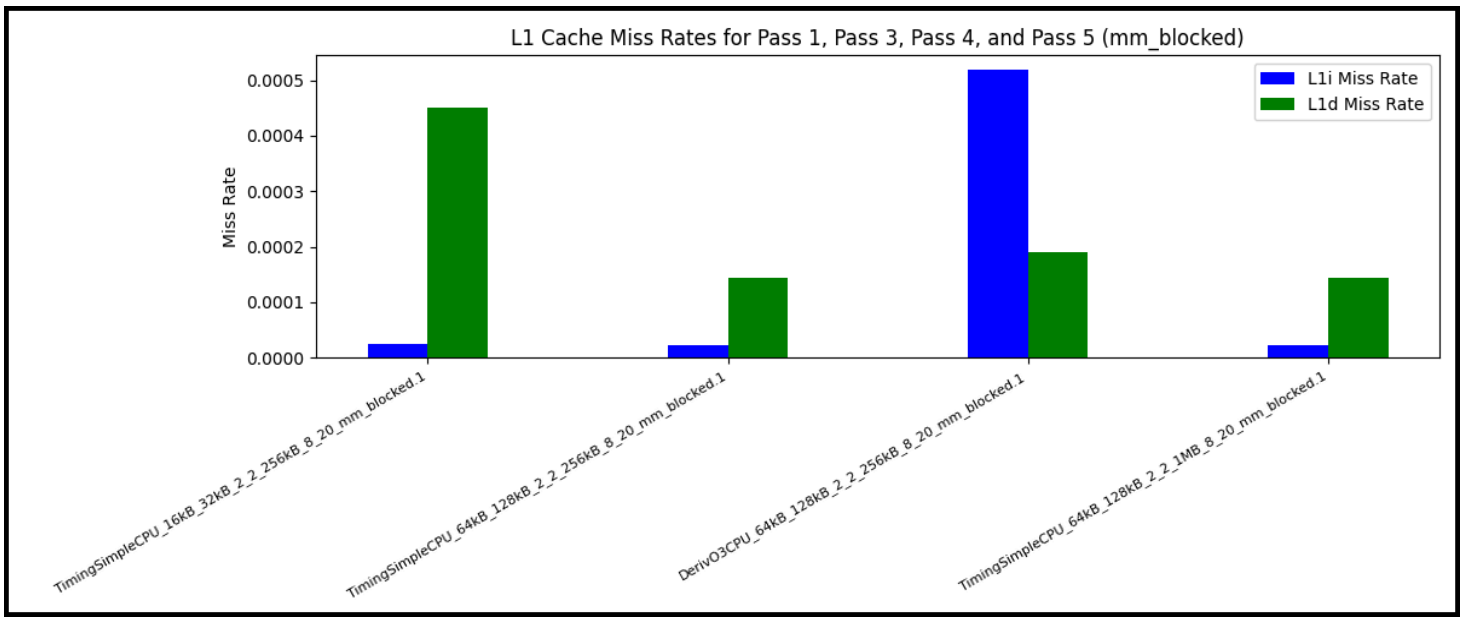
Miss Rates :

Slightly higher miss rates (0.000562 to 0.000637) compared to Pass 1, with mm_rearranged showing the highest at 0.000637, indicating a minor increase in instruction cache inefficiency.

Noticeably higher miss rates (0.000494 to 0.000937), particularly for mm_rearranged (0.000937), suggesting that data access patterns may not be optimized effectively in this configuration.

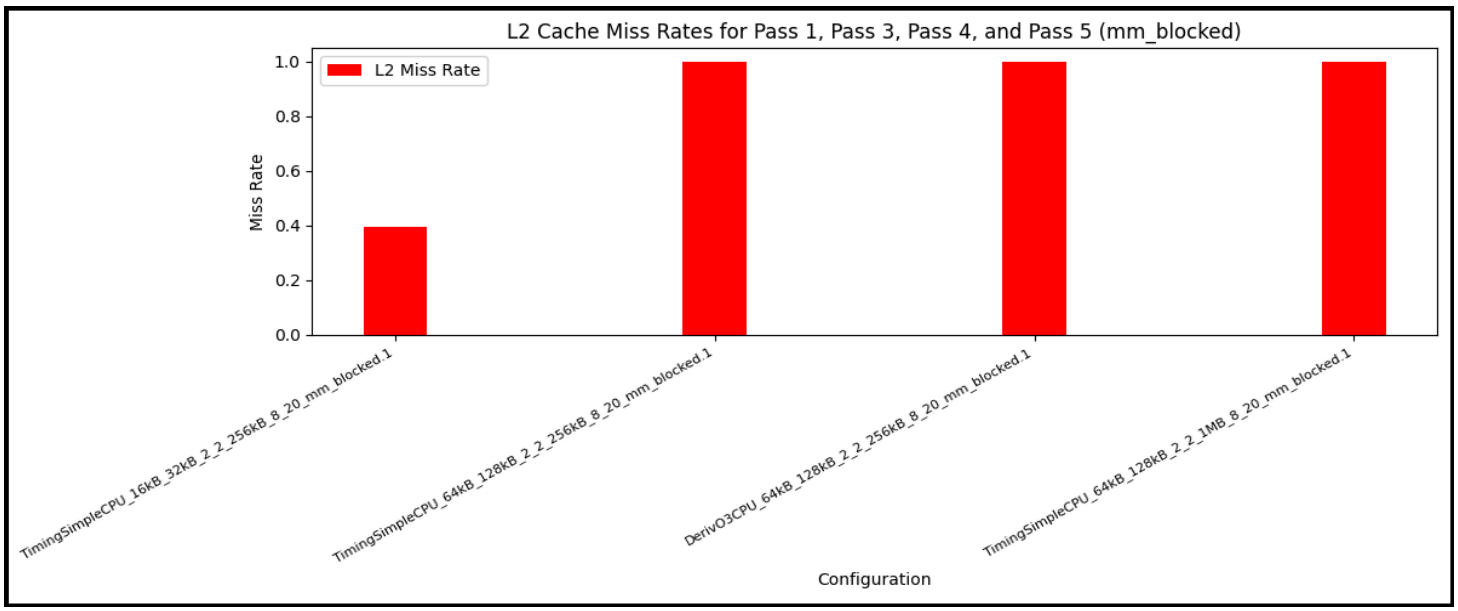
The L2 miss rates are relatively high (0.42 to 0.56), with mm_simple experiencing the highest at 0.56. This indicates that the working set is likely exceeding L2 cache capacity, resulting in significant data fetching delays.

- ***Effect of Cache Size*** **:{Pass 3-5}** ***(blocked MM Algo.)***

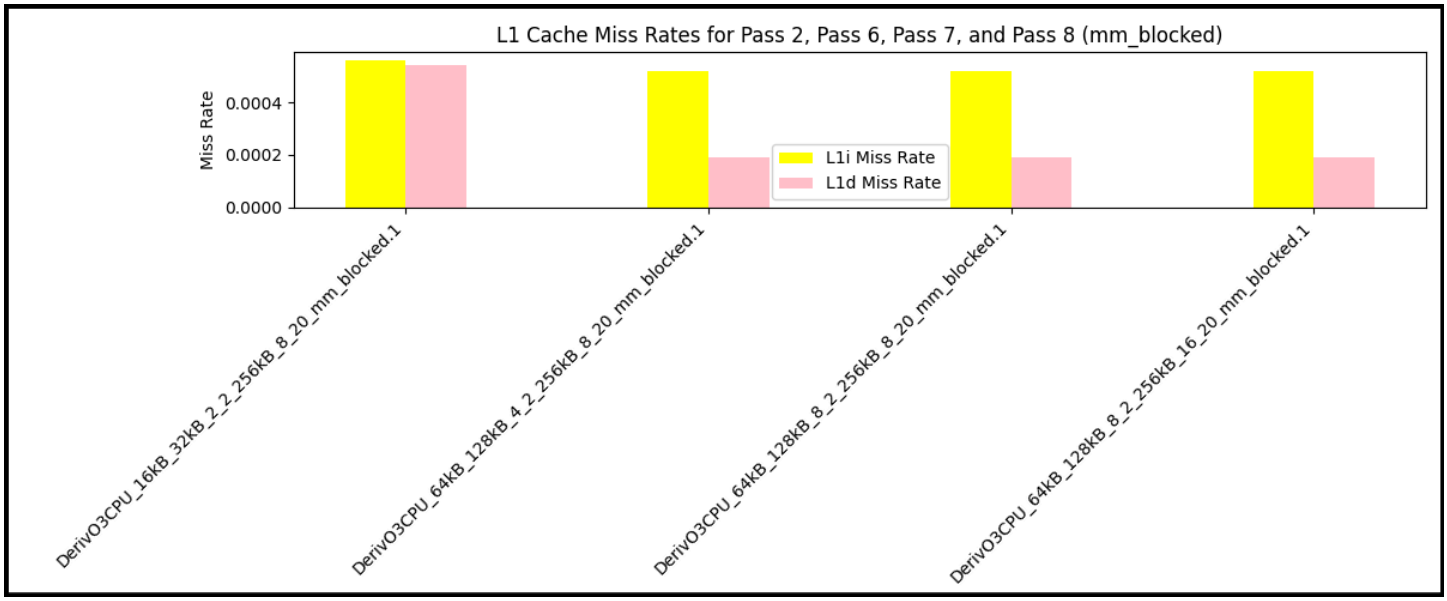


Initially we had 16Kb L1 i-cache, 32Kb L1 d-cache, 256Kb L2 cache. We then Increased L1 i-cache to 64Kb, & reduced L1 d-cache to 128Kb ***{PASS 3}****. We also checked if switching the micro-architecture helps or not, by using DeriveO3CPU instead.* ***{PASS 4}*** *Similarly, we switched to 1 Mb L2 cache & analyzed the code performance.* ***{PASS 5}***

Here, the increase in Cache size didn't necessarily improve performance, rather we had more L2 cache miss rate for larger cache size.

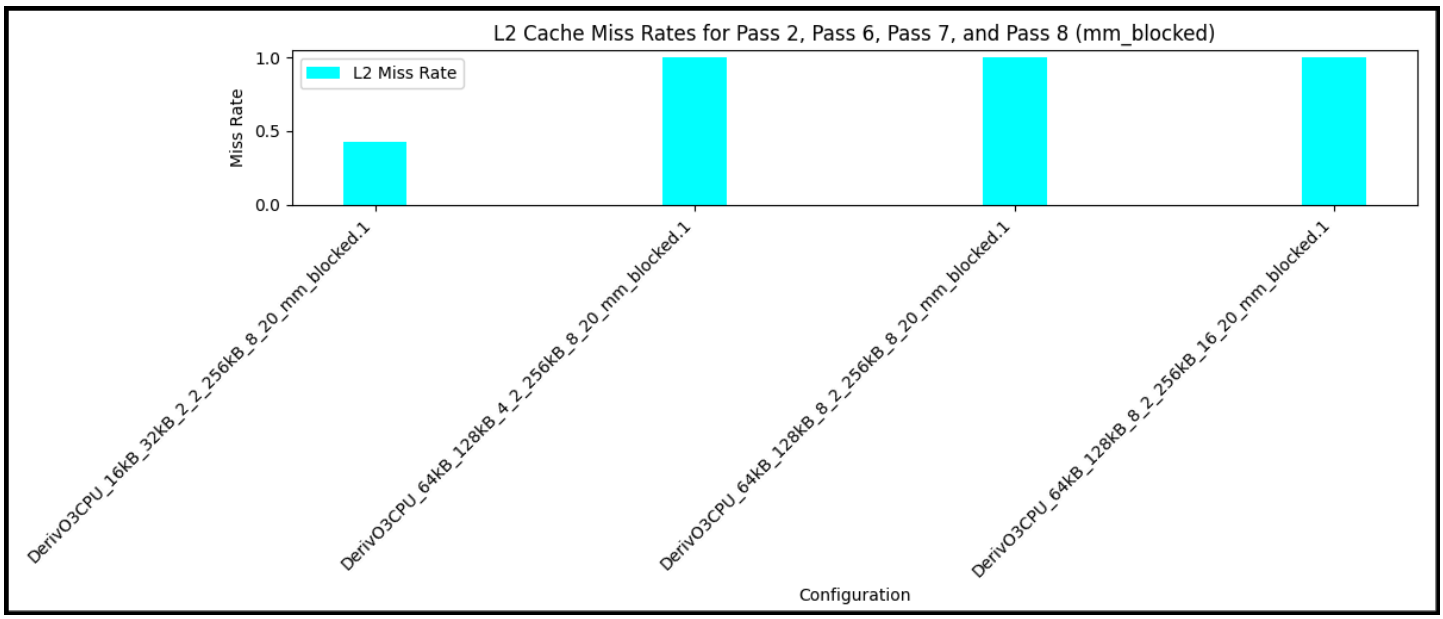


- ***Effect of Cache Associativity*:** **{Pass 6-8}** ***(blocked MM Algo.)***



Initially we had L1 as a 2-way associative cache, L2 as a 8-way associative cache. We then Increased L1 to 4-way associative cache ***{PASS 6}****, 8-way associative cache* ***{PASS 7}*** *Lastly, we Increased L2 to 16-way associative cache* ***{PASS 8}.***

Here, the increase in Cache associativity had a mere performance improvement.



- ***Result:***

    - Configurations with smaller L1 cache sizes, such as 16kB, *tended to yield lower simulation times and CPIs*, demonstrating a balanced trade-off between cache size and performance. The configuration DerivO3CPU_16kB_32kB_2_2_256kB_8_20_mm_simple.1 exemplified this, achieving a simulation time of 0.004365 seconds with a CPI of 0.531065.

    - In contrast, **larger L1 and L2 cache sizes** sometimes resulted in higher miss rates, particularly for the DerivO3CPU configurations, suggesting that increasing cache sizes without optimizing their structure can lead to diminishing returns in performance.