CS 301
High-Performance Computing

Lab 02 - Matrix Multiplication Strategies & Performance Analysis

Naman Patel (202301423)
Jainesh Patel (202301446)

Contents

1 Hardware Details

1.1 Hardware Details for LAB207 PCs

- Architecture: x86_64
- CPU: 12-core 12th Gen Intel(R) Core(TM) i5-12500
- CPU max MHz: 4600.0000
- L1d cache: 288K
- L1i cache: 192K
- L2 cache: 7.5M
- L3 cache: 18M

1.2 Hardware Details for HPC Cluster (Node gics2)

- Architecture: x86_64
- CPU op-mode(s): 32-bit, 64-bit
- Byte Order: Little Endian
- CPU(s): 16
- On-line CPU(s) list: 0-15
- Thread(s) per core: 1
- Core(s) per socket: 8
- Socket(s): 2
- NUMA node(s): 2
- Vendor ID: GenuineIntel
- CPU family: 6
- Model: 63
- Model name: Intel(R) Xeon(R) CPU E5-2640 v3 @ 2.60GHz
- Stepping: 2
- CPU MHz: 1229.414
- BogoMIPS: 5204.38
- Virtualization: VT-x
- L1d cache: 32K

- L1i cache: 32K
- L2 cache: 256K
- L3 cache: 20480K
- NUMA node0 CPU(s): 0-7
- NUMA node1 CPU(s): 8-15

2 Lab PC Analysis

2.1 Loop Permutation: i-j-k (Lab PC)

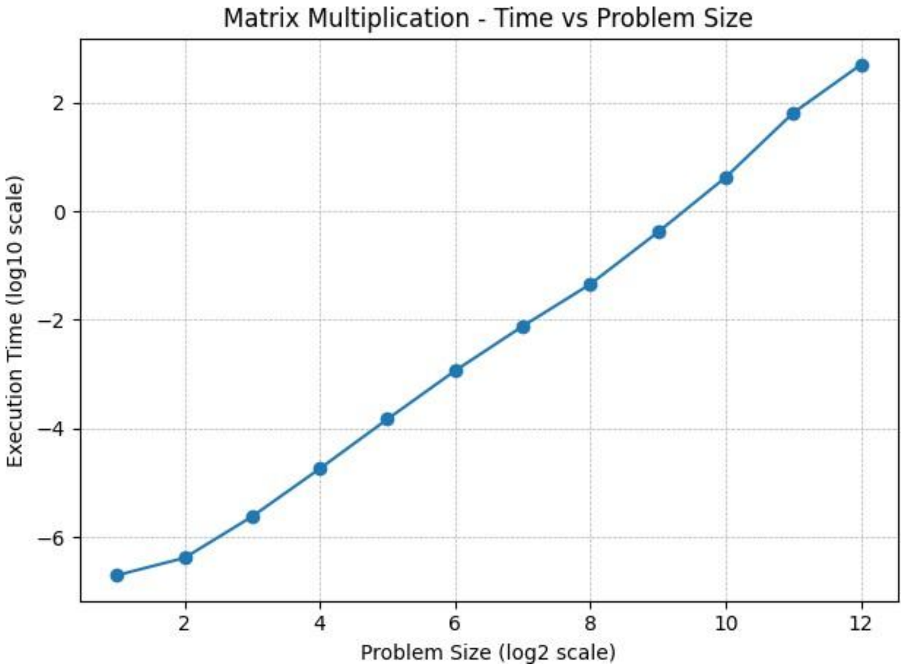

Figure 1: Performance of i-j-k loop order on Lab PC.

Observations: The execution time scales polynomially with the problem size. For this permutation, the inner loop iterates over k, causing the second matrix to be accessed in a non-contiguous (column-wise) manner. This results in frequent cache misses as the problem size N grows, preventing the CPU from reaching peak theoretical throughput.

2.2 Loop Permutation: i-k-j (Lab PC)

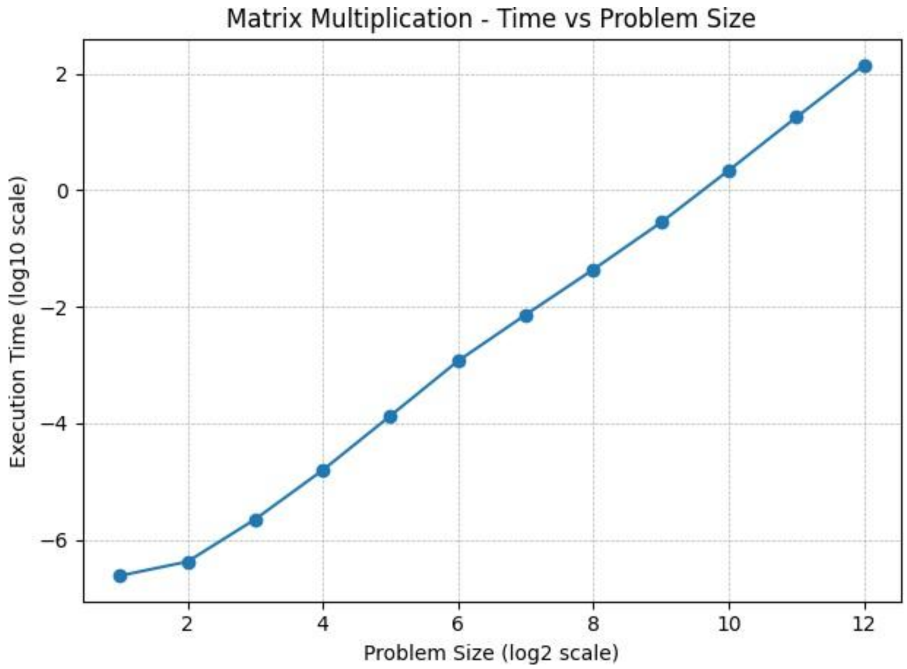

Figure 2: Performance of i-k-j loop order on Lab PC.

Observations: This permutation demonstrates superior performance compared to $i - j - k$. The inner loop iterates over j, allowing both the result matrix and the second operand matrix to be accessed with unit stride (row-major order). This excellent spatial locality maximizes L1 cache hits and vectorization opportunities.

2.3 Loop Permutation: j-i-k (Lab PC)

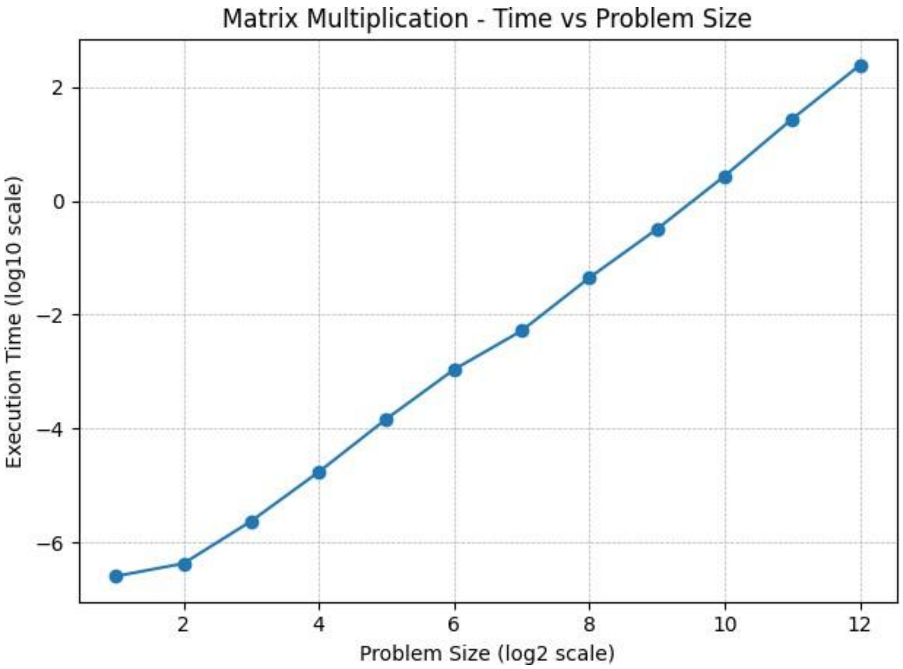

Figure 3: Performance of j-i-k loop order on Lab PC.

Observations: The performance is relatively poor due to the memory access pattern. The inner loop iterates over k, which causes stride-N access for the second matrix. This leads to poor spatial locality and a high number of cache misses, similar to the $i-j-k$ case but often slightly worse due to loop overheads.

2.4 Loop Permutation: j-k-i (Lab PC)

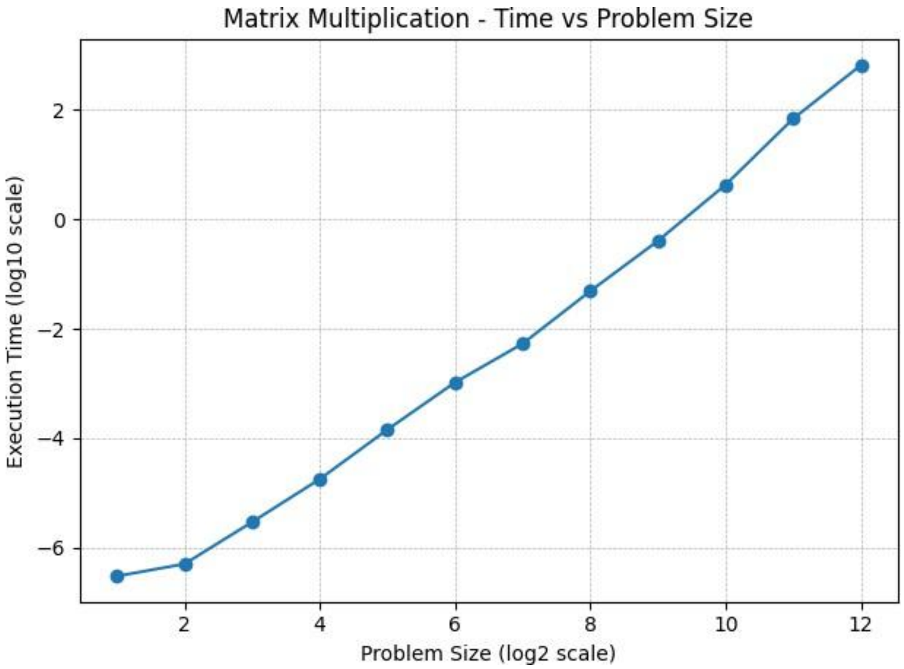

Figure 4: Performance of j-k-i loop order on Lab PC.

Observations: This is one of the slowest permutations. The inner loop iterates over i, which means we are traversing columns for the matrices in the inner loop. This results in a large stride (equal to dimension N) for memory accesses, causing excessive Translation Lookaside Buffer (TLB) misses and cache thrashing.

2.5 Loop Permutation: k-i-j (Lab PC)

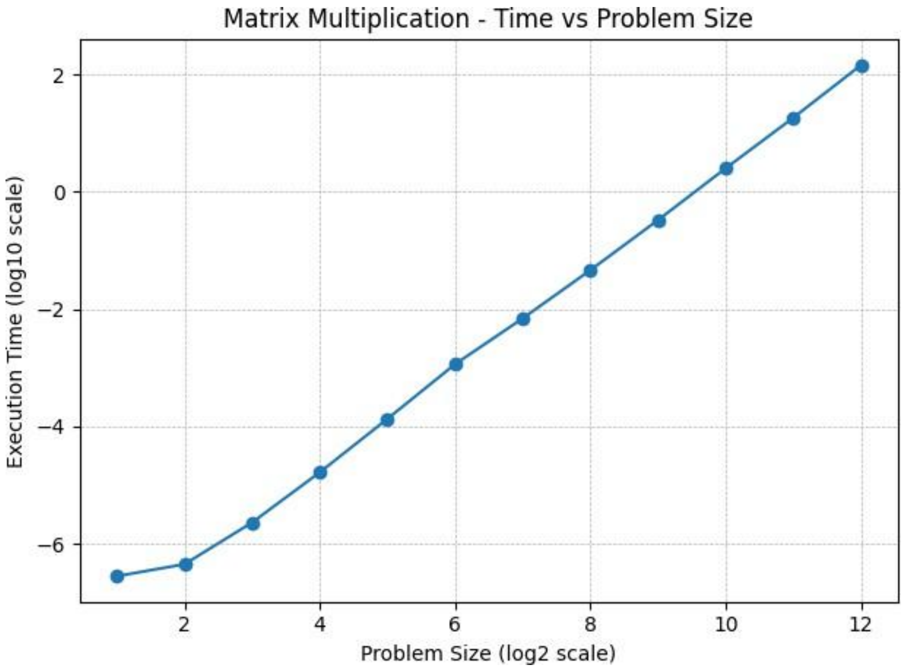

Figure 5: Performance of k-i-j loop order on Lab PC.

Observations: Similar to $i - k - j$, this permutation performs very well. The inner loop variable is j, which ensures unit-stride memory access for the matrices. The prefetching hardware can effectively predict the next memory addresses, keeping the execution pipeline fed and minimizing latency.

2.6 Loop Permutation: k-j-i (Lab PC)

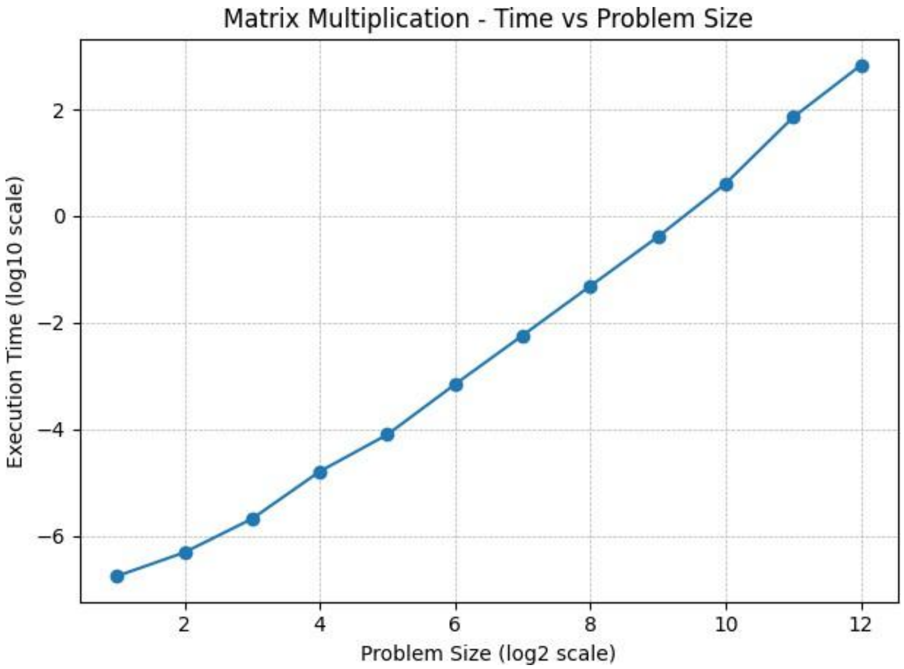

Figure 6: Performance of k-j-i loop order on Lab PC.

Observations: This permutation suffers from significant performance degradation. Like the $j - k - i$ ordering, the inner loop iterates over i, leading to non-contiguous memory access. The CPU spends a majority of cycles waiting for data to be fetched from the main memory rather than performing computations.

2.7 Matrix Multiplication using Transpose (Lab PC)

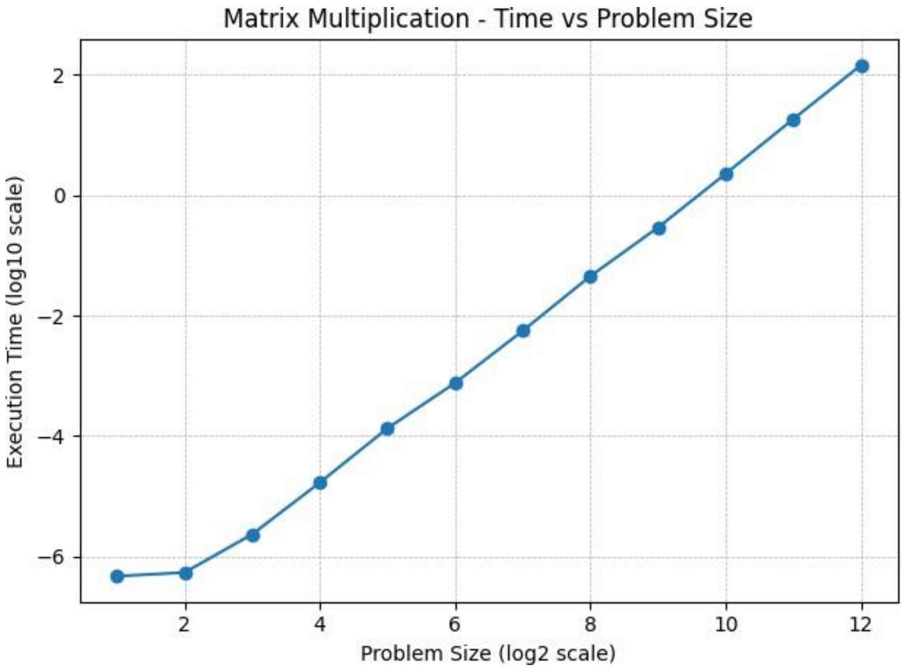

Figure 7: Performance of Matrix Multiplication with Transpose on Lab PC.

Observations: By explicitly transposing the second matrix before multiplication, we convert the access pattern from column-wise to row-wise. This eliminates the stride-N access penalty found in naive implementations. The performance is comparable to the best loop permutations (*ikj/kij*) as it effectively utilizes spatial locality.

2.8 Block Matrix Multiplication (Lab PC)

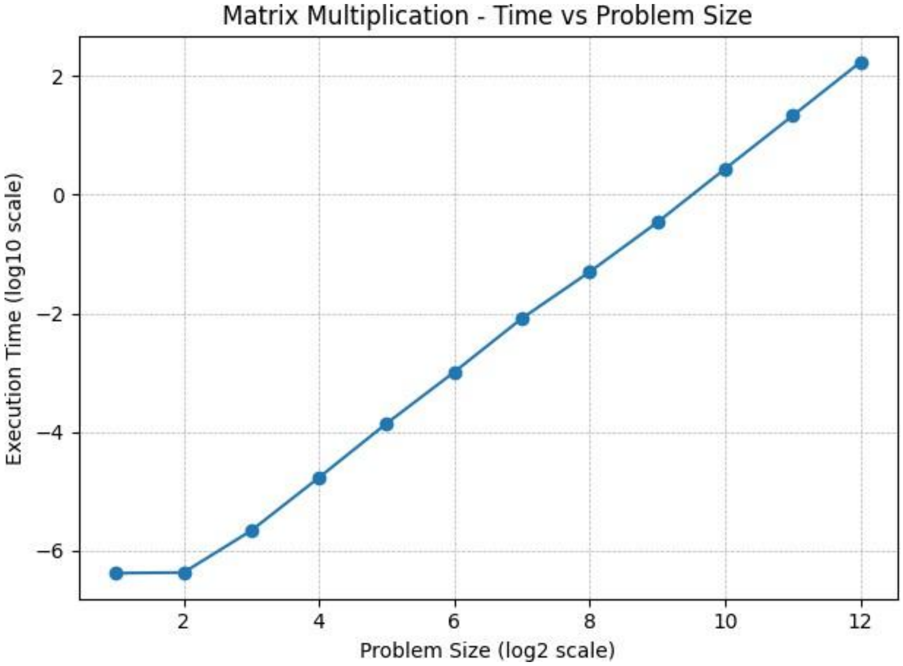

Figure 8: Performance of Block Matrix Multiplication on Lab PC.

Observations: The blocked implementation helps maintain performance as problem size increases. By processing sub-matrices (blocks) that fit entirely within the L1/L2 cache, we maximize temporal locality and reuse data elements before they are evicted. This approach is robust against cache capacity misses for large N.

3 HPC Cluster Analysis

3.1 Loop Permutation: i-j-k (Cluster)

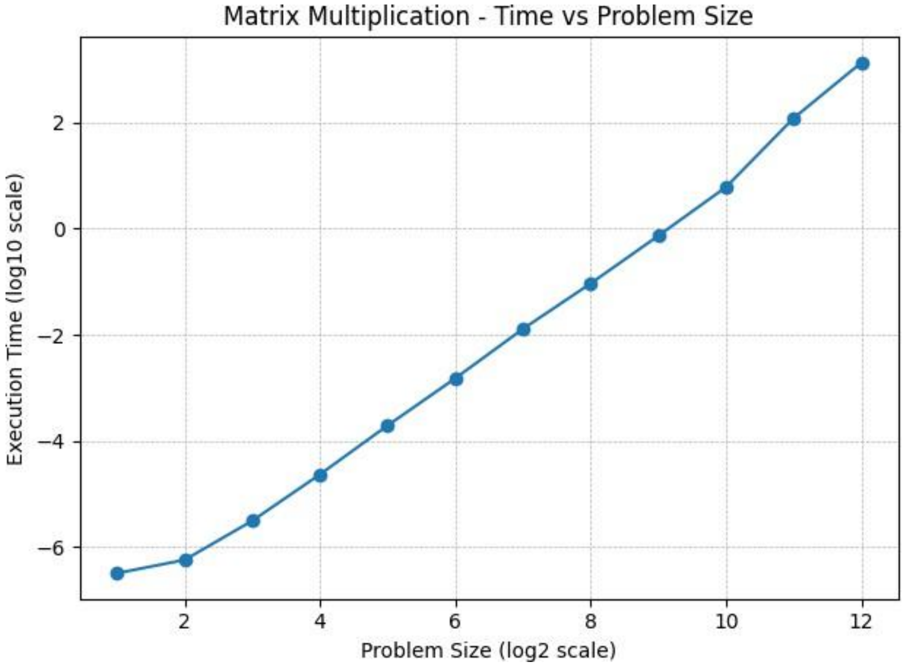

Figure 9: Performance of i-j-k loop order on HPC Cluster.

Observations: On the cluster, the $i-j-k$ order shows a standard performance curve. However, due to the larger L3 cache (20MB) compared to the Lab PC, the drop in performance due to cache misses occurs at a slightly larger problem size. The bottleneck remains the non-unit stride access of the second matrix.

3.2 Loop Permutation: i-k-j (Cluster)

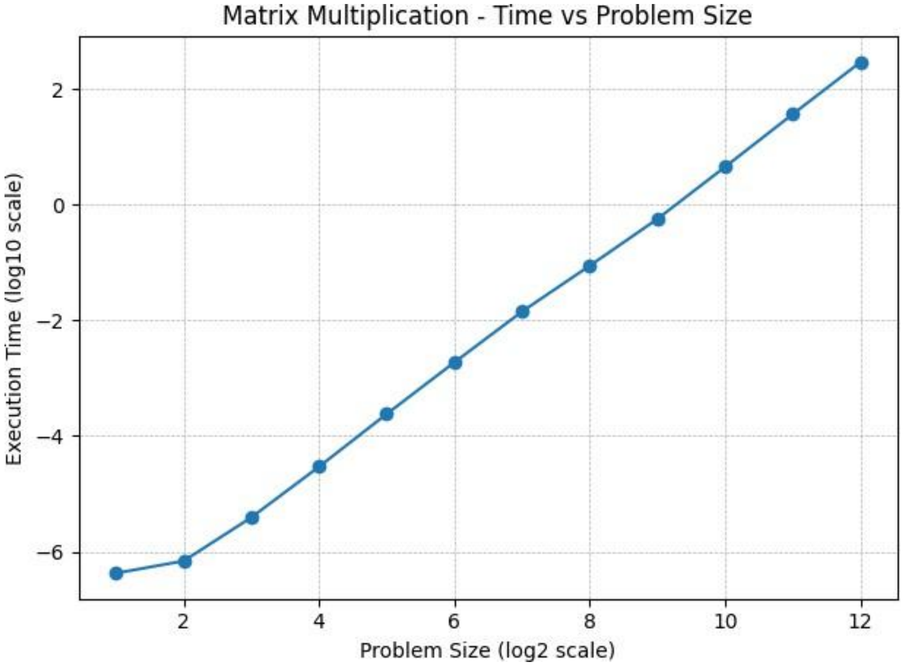

Figure 10: Performance of i-k-j loop order on HPC Cluster.

Observations: This permutation yields optimal performance on the Cluster. The Xeon architecture benefits significantly from the unit-stride access pattern in the inner loop, allowing for efficient vectorization (AVX instructions) and minimizing stalls in the execution pipeline.

3.3 Loop Permutation: j-i-k (Cluster)

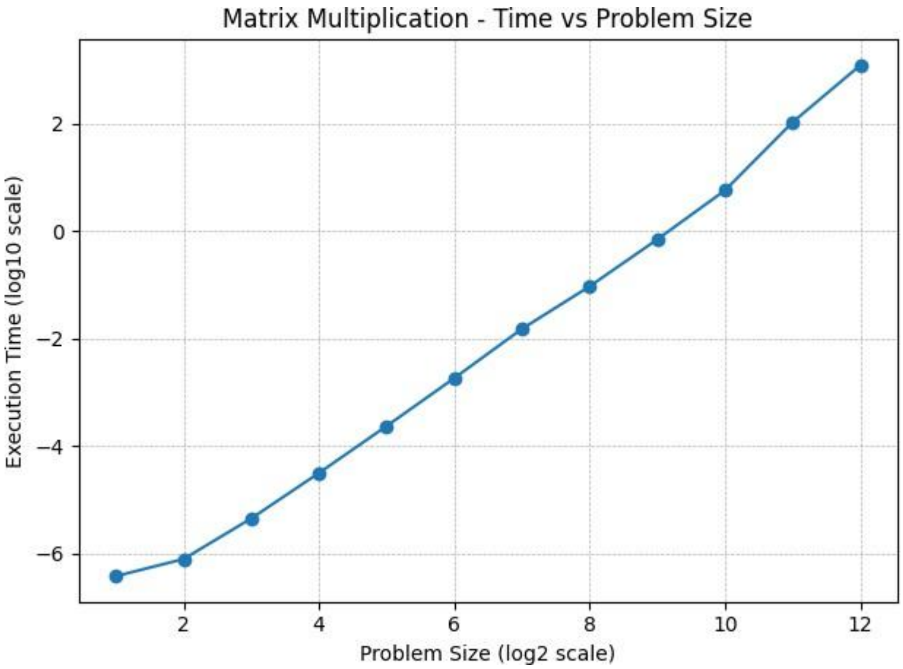

Figure 11: Performance of j-i-k loop order on HPC Cluster.

Observations: Performance is suboptimal. The graph indicates a steeper increase in execution time as N grows, confirming that the memory bandwidth is saturated by inefficient cache usage. The stride-N access pattern prevents the hardware prefetcher from working effectively.

3.4 Loop Permutation: j-k-i (Cluster)

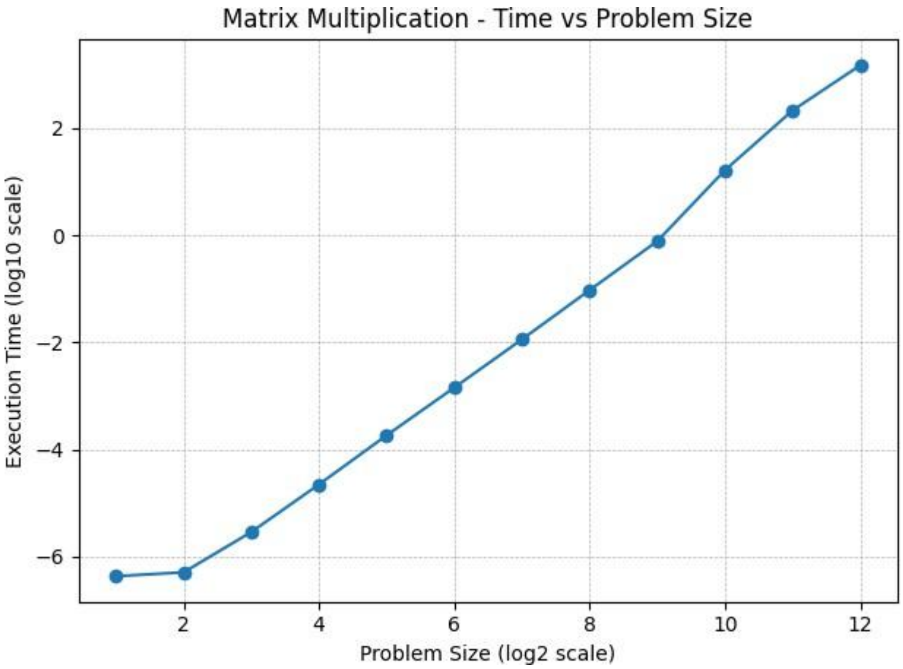

Figure 12: Performance of j-k-i loop order on HPC Cluster.

Observations: This ordering performs poorly on the Cluster. The larger cache of the Xeon processor cannot compensate for the extreme inefficiency of stride-N access in the inner loop. The resulting high cache miss rate makes the operation memory-bound rather than compute-bound.

3.5 Loop Permutation: k-i-j (Cluster)

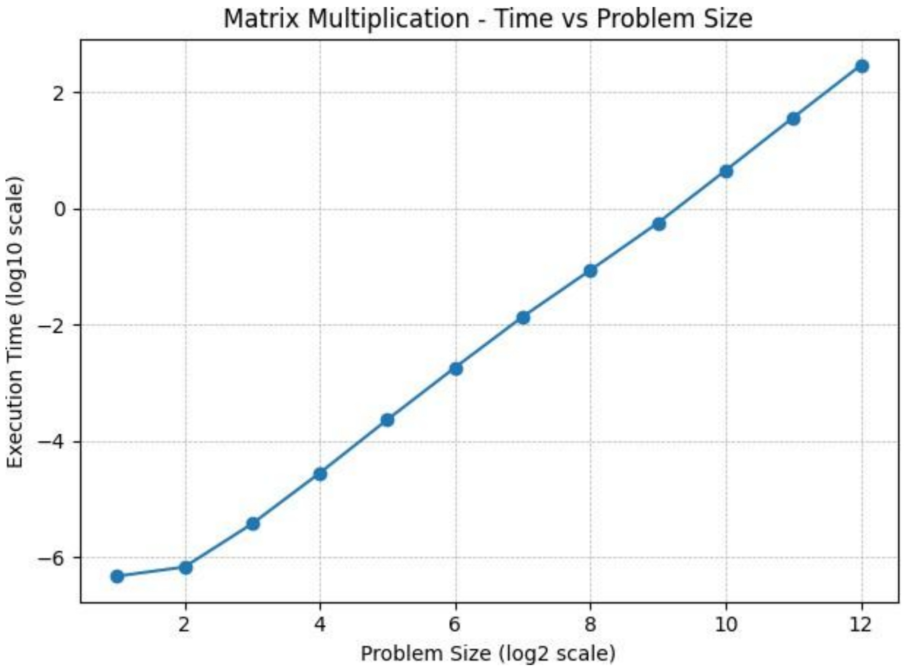

Figure 13: Performance of k-i-j loop order on HPC Cluster.

Observations: Matches the high performance of the $i - k - j$ permutation. The consistent low execution times demonstrate that arranging loops to access memory contiguously is the most critical optimization for matrix multiplication on this architecture.

3.6 Loop Permutation: k-j-i (Cluster)

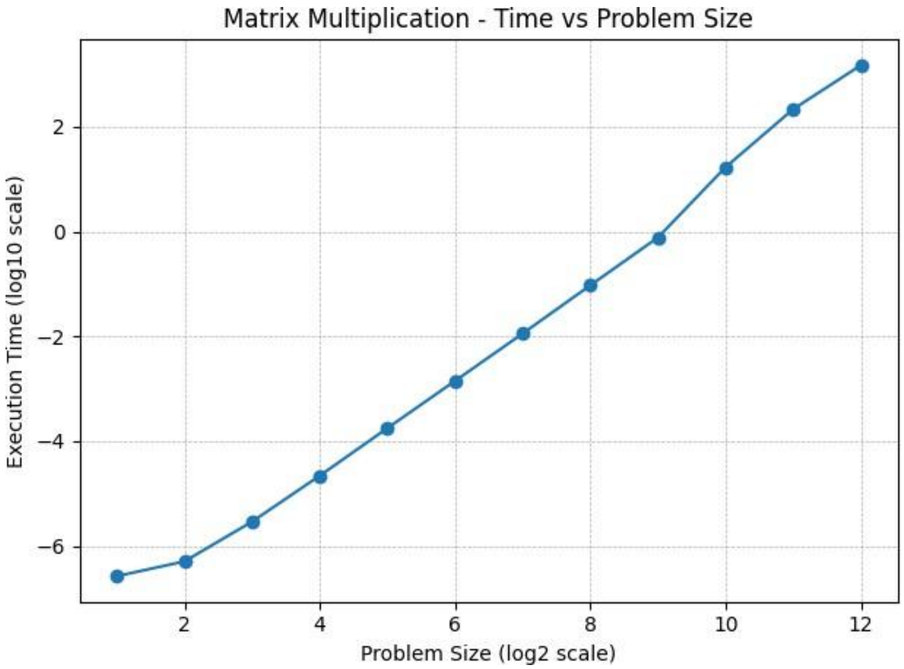

Figure 14: Performance of k-j-i loop order on HPC Cluster.

Observations: This permutation exhibits the worst-case behavior. The graph shows high execution times even for moderate problem sizes. The random-like access pattern (from the cache's perspective) leads to significant latency, severely underutilizing the Cluster's computational power.

3.7 Matrix Multiplication using Transpose (Cluster)

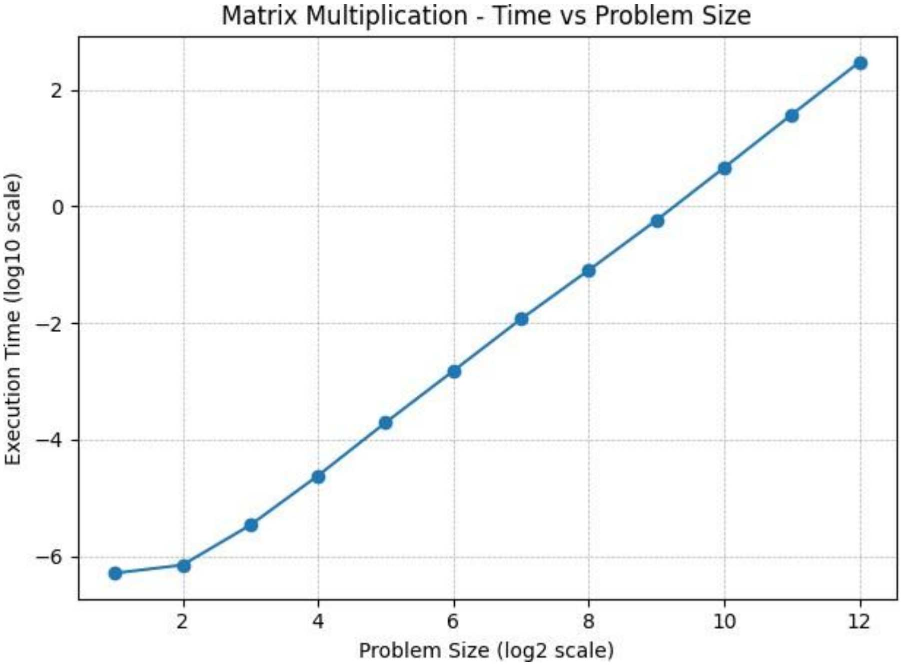

Figure 15: Performance of Matrix Multiplication with Transpose on HPC Cluster.

Observations: The transpose method performs efficiently on the Cluster. The initial overhead of the transposition step is negligible compared to the time saved during the $O(N^3)$ multiplication phase, where the row-major access pattern ensures high cache hit rates.

3.8 Block Matrix Multiplication (Cluster)

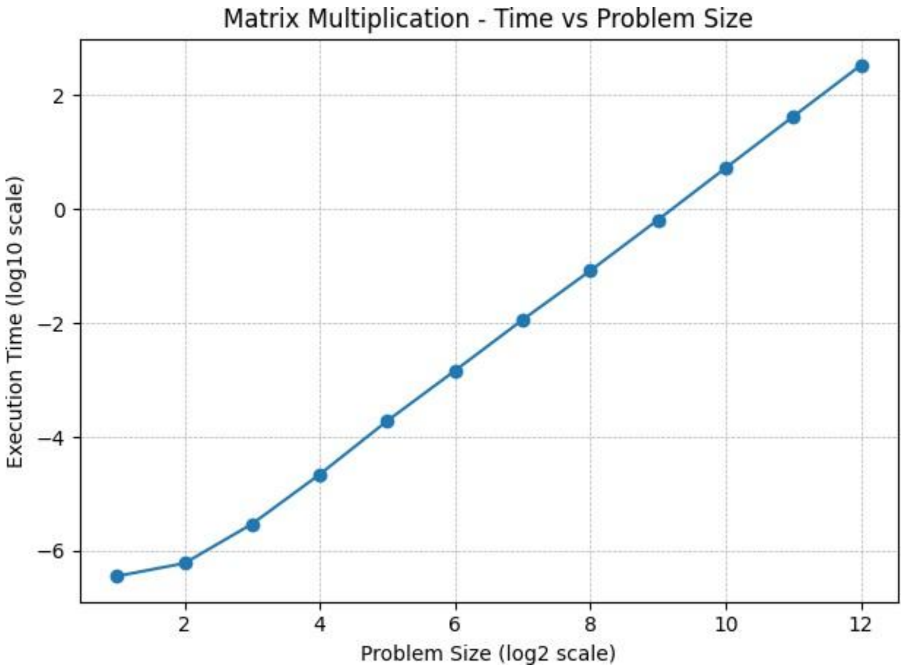

Figure 16: Performance of Block Matrix Multiplication on HPC Cluster.

Observations: Blocking proves effective on the Cluster, particularly for the largest problem sizes tested. By tuning the block size to fit the Xeon's cache hierarchy, we maintain a linear log-log slope even as the total matrix size exceeds the L3 cache capacity, demonstrating better scalability than non-blocked versions.

4 Conclusion

In this lab, we observed how memory access patterns significantly impact performance. The loop orderings that accessed memory contiguously (unit stride) performed much better than those with non-contiguous access. The Transpose method effectively improved spatial locality, while Block Matrix Multiplication improved temporal locality by utilizing the cache more efficiently. These trends were consistent across both the Lab PC and the HPC Cluster, though the specific crossover points varied due to differences in cache sizes.