CS 301
High-Performance Computing

Lab 02 - Matrix Multiplication

Visha Sitapara (202301414)
Dhruv Patel (202301095)

February 17, 2026

Contents

1 Introduction

This lab focuses on analyzing the performance of different matrix multiplication techniques by varying problem size and data types. The objective is to study runtime behavior, memory vs compute characteristics, and achieved performance, and compare them with the theoretical limits of the underlying hardware.

2 Hardware Details

2.1 Hardware Details for LAB207 PCs

- Architecture: x86_64
- CPU op-mode(s): 32-bit, 64-bit
- Address sizes: 46 bits physical, 48 bits virtual
- Byte Order: Little Endian
- CPU(s): 12
- On-line CPU(s) list: 0-11
- Thread(s) per core: 2
- Core(s) per socket: 6
- Socket(s): 1
- NUMA node(s): 1
- Vendor ID: GenuineIntel
- CPU family: 6
- Model: 151
- Model name: 12th Gen Intel(R) Core(TM) i5-12500
- Stepping: 5
- CPU max MHz: 4600.0000
- CPU min MHz: 800.0000
- BogoMIPS: 5990.40
- Virtualization: VT-x
- L1d cache: 288K
- L1i cache: 192K

- L2 cache: 7.5M
- L3 cache: 18M
- NUMA node0 CPU(s): 0-11
- Flags: fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36 clflush dts acpi mmx fxsr sse sse2 ss ht tm pbe syscall nx pdpe1gb rdtscp lm constant_tsc arch_perfmon pebs bts rep_good nopl xtopology nonstop_tsc aperfmperf eagerfpu pni pclmulqdq dtes64 monitor ds_cpl vmx smx est tm2 ssse3 fma cx16 xtpr pdcm pcid sse4_1 sse4_2 x2apic movbe popcnt tsc_deadline_timer aes xsave avx f16c rdrand lahf_lm abm epb invpcid_single tpr_shadow vnmi flexpriority ept vpid fsgsbase tsc_adjust bmi1 avx2 smep bmi2 erms invpcid xsaveopt dtherm ida arat pln pts

2.2 Hardware Details for HPC Cluster (Node gics0)

- Architecture: x86_64
- CPU op-mode(s): 32-bit, 64-bit
- Byte Order: Little Endian
- CPU(s): 24
- On-line CPU(s) list: 0-23
- Thread(s) per core: 2
- Core(s) per socket: 6
- 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: 2907.656
- BogoMIPS: 4804.69
- Virtualization: VT-x
- L1d cache: 32K
- L1i cache: 32K

- L2 cache: 256K
- L3 cache: 15360K
- NUMA node0 CPU(s): 0-5, 12-17
- NUMA node1 CPU(s): 6-11, 18-23

3 Problem Description

All runs need to be performed on Two machines → lab machine (207) and cluster. At least 5 times for statistical analysis of fluctuations in run-time.

Problem A-1 → Conventional matrix multiplication (interchange the loops - 6 possibilities)
Problem B-1 → Matrix multiplication using transpose
Problem C-1 → Block matrix multiplication (divide and conquer strategy)

Goal - measure performance/runtime as a function of problem size (2^2 to 2^{12}). Use integers (4 bytes) as well as floating points (8 bytes) for inputs.

4 Benchmarking Methodology

- Conventional matrix multiplication with all six loop orderings: ijk, ikj, jik, jki, kij, kji

$$C_{ij} = \sum_{k=1}^{n} A_{ik} B_{kj}$$

- Matrix multiplication using transpose of one input matrix.

$$C_{ij} = \sum_{k=1}^{n} A_{ik} B_{jk}^{T}$$

- Block (tiled) matrix multiplication using divide and conquer strategy.

$$C_{ij}^{(\text{block})} = \sum_{k=1}^{n/b} A_{ik}^{(\text{block})} B_{kj}^{(\text{block})}$$

5 Graphical Results

5.1 Conventional matrix multiplication (ijk loop)

5.1.1 Using Lab PC

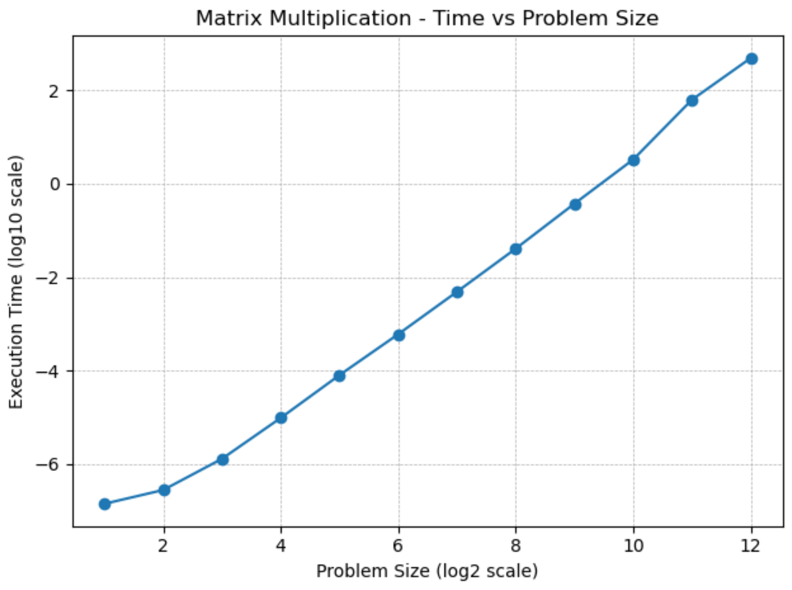

Figure 1: Time vs. Problem Size for the Conventional matrix multiplication (ijk loop) using Lab PC.

5.1.2 Using HPC Cluster

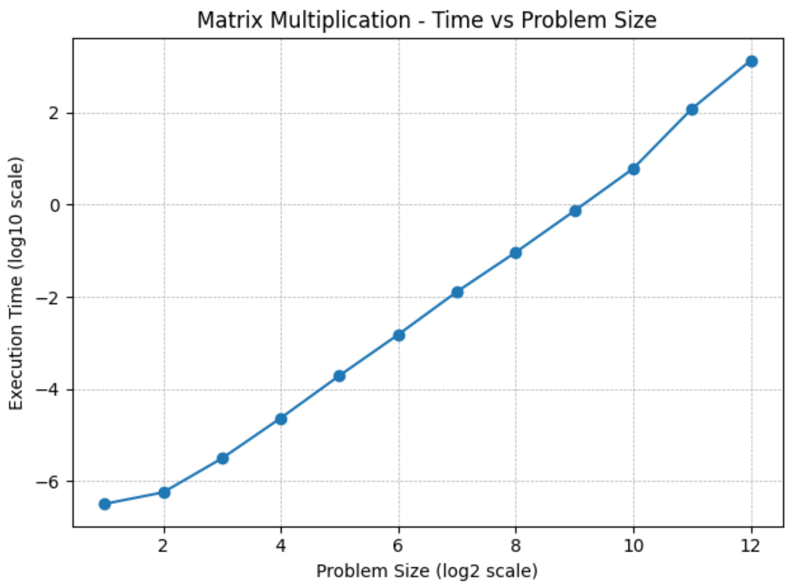

Figure 2: Time vs. Problem Size for the Conventional matrix multiplication (ijk loop) using HPC Cluster.

5.2 Conventional matrix multiplication (ikj loop)

5.2.1 Using Lab PC

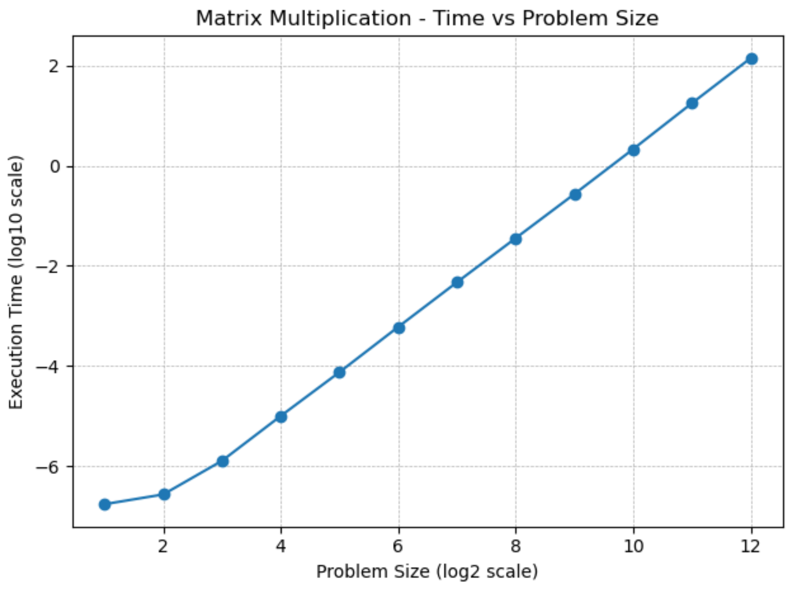

Figure 3: Time vs. Problem Size for the Conventional matrix multiplication (ikj loop) using Lab PC.

5.2.2 Using HPC Cluster

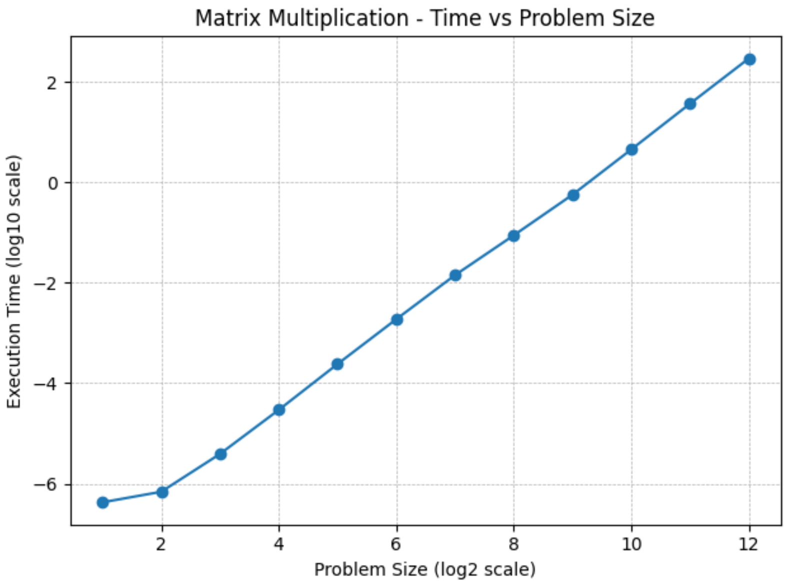

Figure 4: Time vs. Problem Size for the Conventional matrix multiplication (ikj loop) using HPC Cluster.

5.3 Conventional matrix multiplication (jik loop)

5.3.1 Using Lab PC

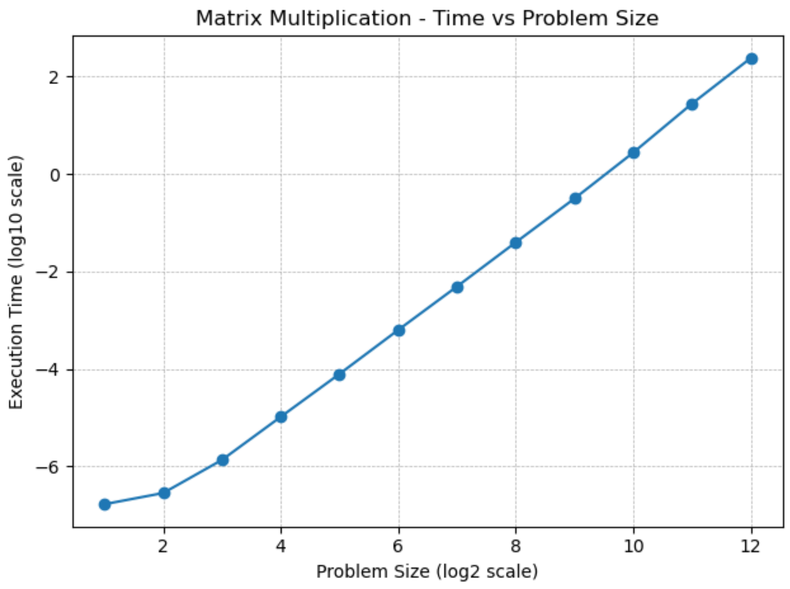

Figure 5: Time vs. Problem Size for the Conventional matrix multiplication (jik loop) using Lab PC.

5.3.2 Using HPC Cluster

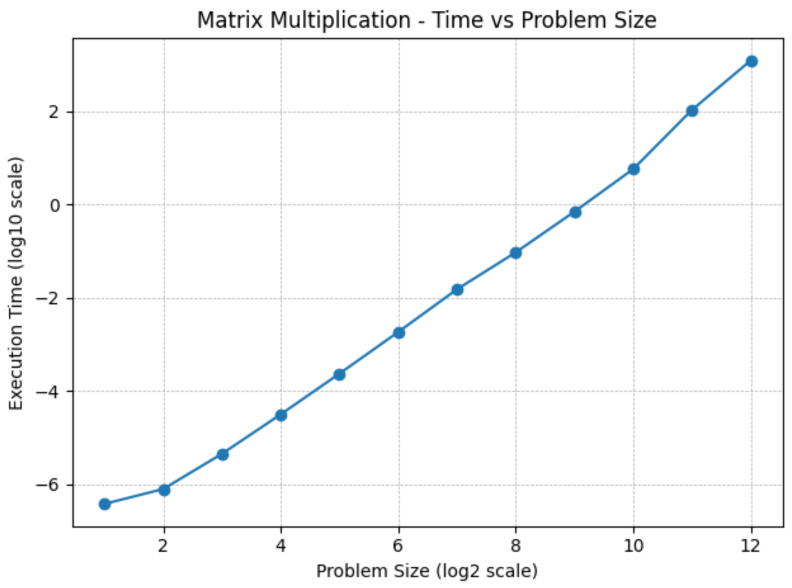

Figure 6: Time vs. Problem Size for the Conventional matrix multiplication (jik loop) using HPC Cluster.

5.4 Conventional matrix multiplication (jki loop)

5.4.1 Using Lab PC

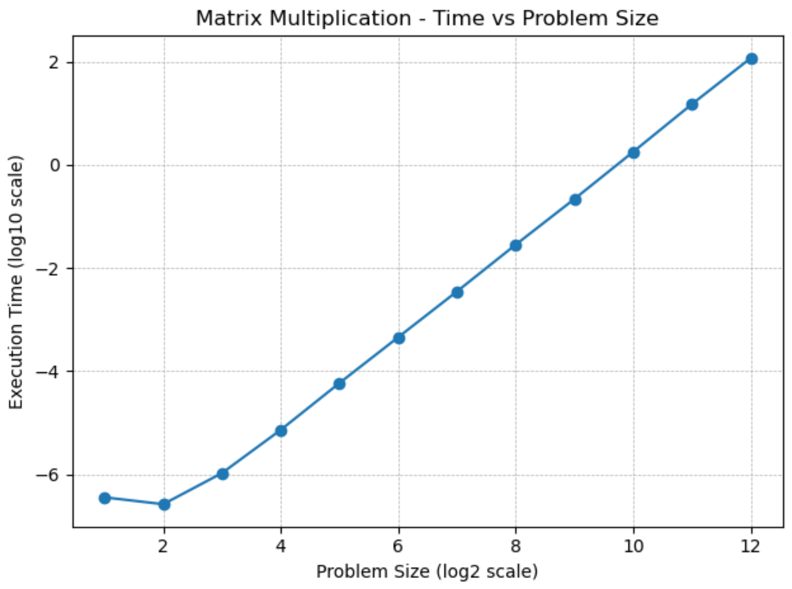

Figure 7: Time vs. Problem Size for the Conventional matrix multiplication (jki loop) using Lab PC.

5.4.2 Using HPC Cluster

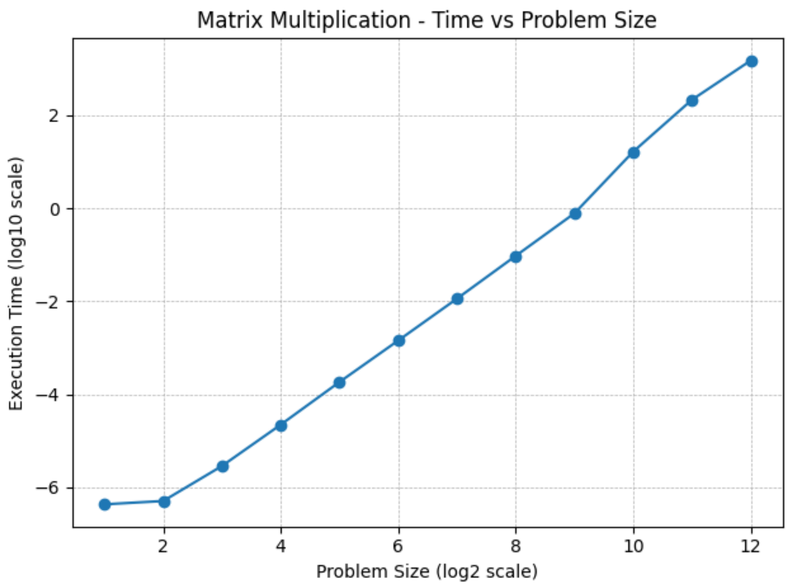

Figure 8: Time vs. Problem Size for the Conventional matrix multiplication (jki loop) using HPC Cluster.

5.5 Conventional matrix multiplication (kij loop)

5.5.1 Using Lab PC

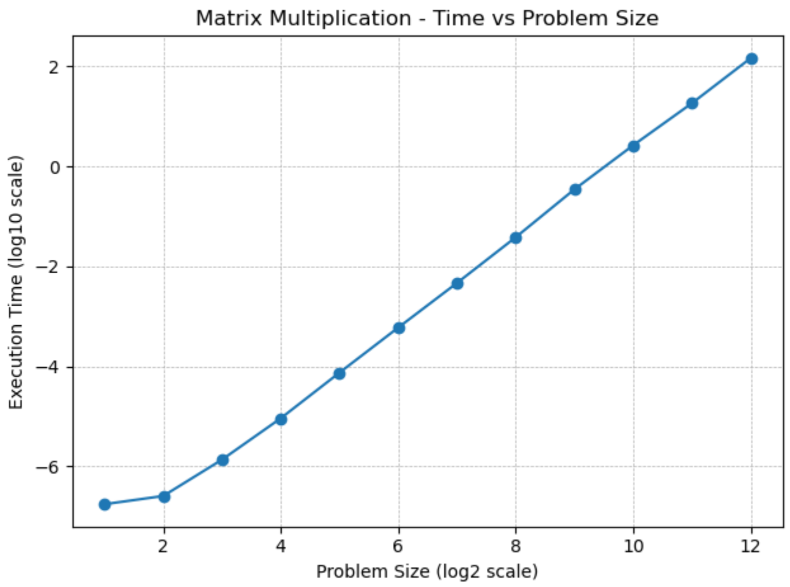

Figure 9: Time vs. Problem Size for the Conventional matrix multiplication (kij loop) using Lab PC.

5.5.2 Using HPC Cluster

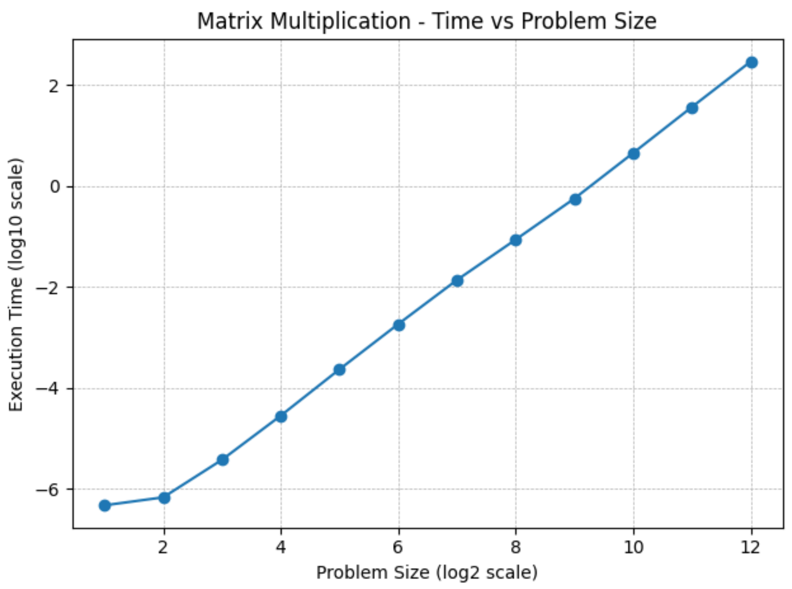

Figure 10: Time vs. Problem Size for the Conventional matrix multiplication (kij loop) using HPC Cluster.

5.6 Conventional matrix multiplication (kji loop)

5.6.1 Using Lab PC

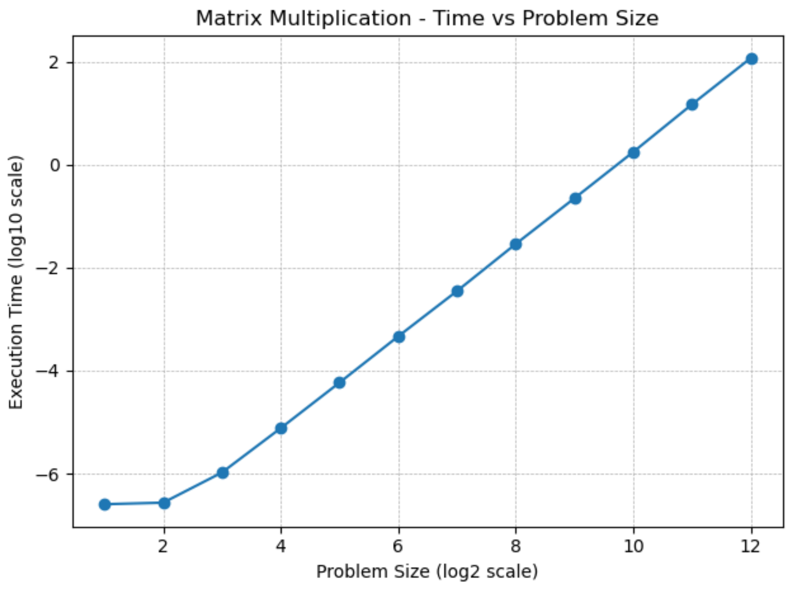

Figure 11: Time vs. Problem Size for the Conventional matrix multiplication (kji loop) using Lab PC.

5.6.2 Using HPC Cluster

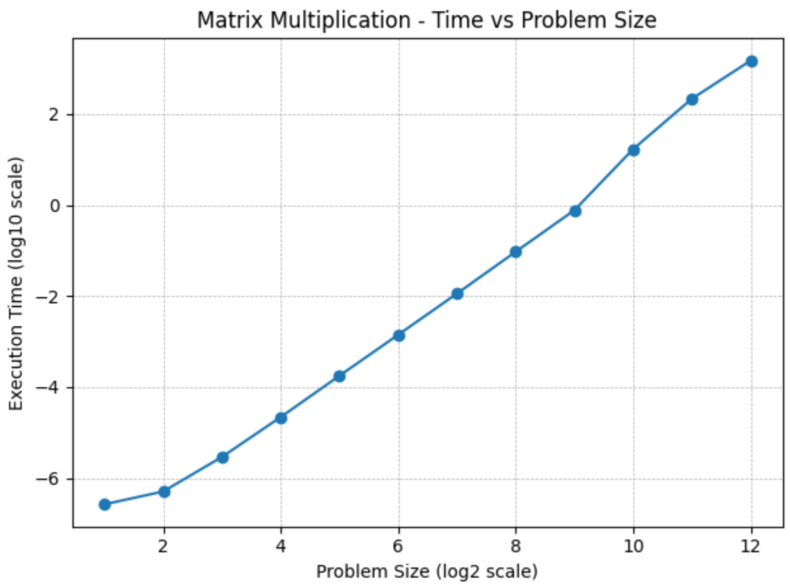

Figure 12: Time vs. Problem Size for the Conventional matrix multiplication (kji loop) using HPC Cluster.

5.7 Matrix multiplication using transpose of one input matrix.

$$C_{ij} = \sum_{k=1}^{n} A_{ik} B_{jk}^{T}$$

5.7.1 Using Lab PC

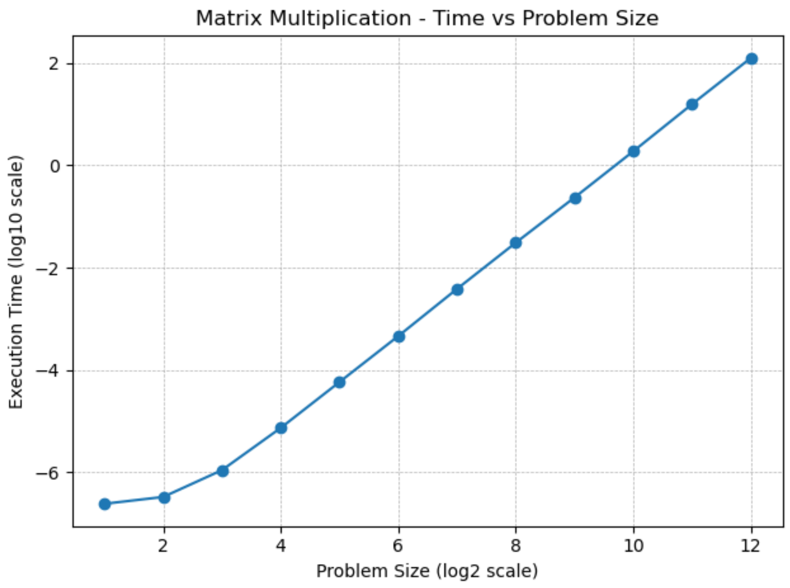

Figure 13: Time vs. Problem Size for the Matrix multiplication using transpose using Lab PC.

5.7.2 Using HPC Cluster

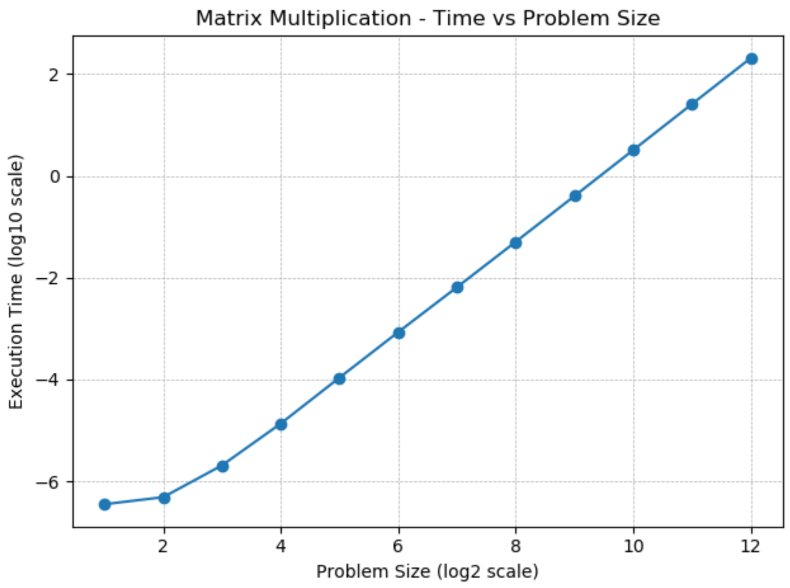

Figure 14: Time vs. Problem Size for the Matrix multiplication using transpose using HPC Cluster.

5.8 Block (tiled) matrix multiplication using divide and conquer strategy.

$$C_{ij}^{(\text{block})} = \sum_{k=1}^{n/b} A_{ik}^{(\text{block})} B_{kj}^{(\text{block})}$$

5.8.1 Using Lab PC

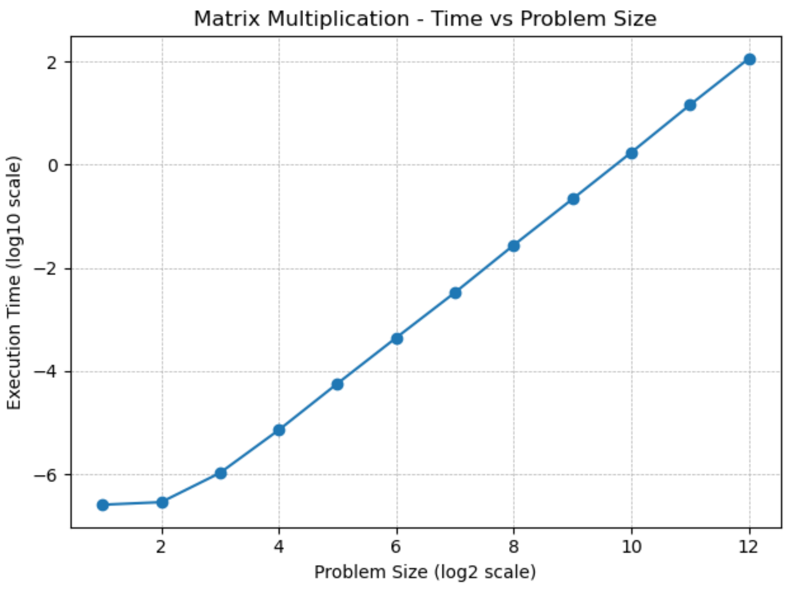

Figure 15: Time vs. Problem Size for the Block Matrix multiplication using Lab PC.

5.8.2 Using HPC Cluster

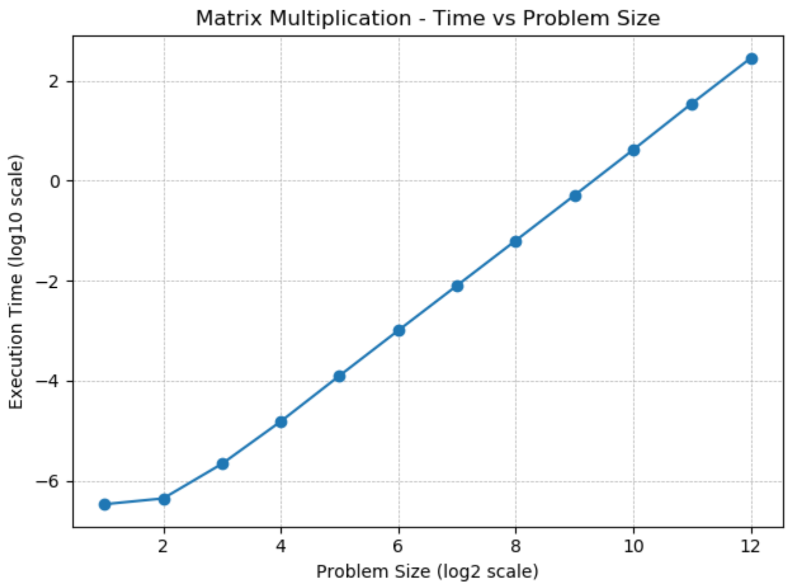

Figure 16: Time vs. Problem Size for the Block Matrix multiplication using HPC Cluster.

6 Conclusion

- The execution time increased approximately as $\mathcal{O}(n^3)$ with problem size, consistent with the theoretical complexity of matrix multiplication.
- Different loop orderings produced noticeable performance differences due to variations in **cache utilization** and **memory access patterns**.
- Implementing matrix multiplication using the transpose of one input matrix improved **spatial locality** and reduced cache misses, resulting in better runtime compared to naive implementations.
- Block (tiled) matrix multiplication further improved performance by enhancing **cache reuse** and reducing **memory traffic**.