CS 301
High-Performance Computing

Lab3 - Performance Analysis of various Matrix Multiplication Algorithms

Pratham Patel (202201485)
Ramkumar Patel (202201509)

Contents

1 Hardware Details

To compare the performance of different architectures, we analyzed two machines: Lab 207 PC and HPC Cluster. The key specifications are summarized in Table 1.

Specification	**Lab 207 PC**	**HPC Cluster**
Architecture	x86_64	x86_64
CPU Model	12th Gen Intel Core i5-12500	Intel Xeon E5-2620 v3 @ 2.40GHz
Clock Speed (GHz)	0.8 - 4.6	2.4
Cores / Threads	6 / 12	12 / 24
L1 Cache (KB)	288 (6 instances)	32 (data + instruction)
L2 Cache (KB)	7.5 MiB (6 instances)	256 KiB
L3 Cache (MB)	18	15
Memory (GB)	8 (DDR4)	64 (DDR4)
Memory Channels	2	2 (NUMA: 2 nodes)
Hyper-Threading	Yes	No

Table 1: Comparison of Hardware Architectures

2 Problem A-1: Conventional Matrix Multiplication

2.1 Description

Matrix multiplication is a fundamental operation in scientific computing and plays a crucial role in various high-performance computing applications. The conventional method follows:

$$C_{i,j} = \sum_{k=0}^{n-1} A_{i,k} B_{k,j} \tag{1}$$

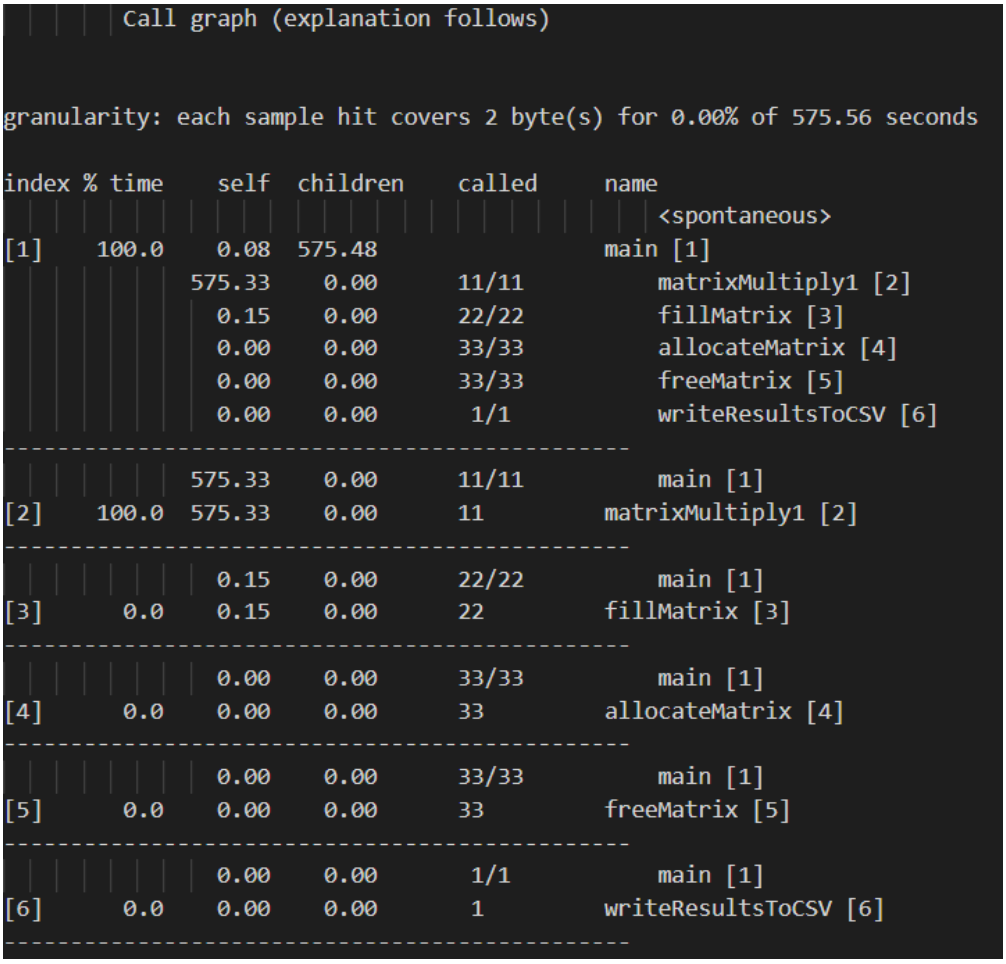
2.2 Complexity Analysis (Serial Implementation)

The naive implementation consists of three nested loops, leading to a time complexity of:

$$O(n^3) \tag{2}$$

For large matrices, this approach becomes inefficient. Enhancing memory access patterns can greatly improve performance in high-performance computing (HPC) environments.

2.3 Profiling Information



```
Call graph (explanation follows)


granularity: each sample hit covers 2 byte(s) for 0.00% of 575.56 seconds

index % time    self  children    called     name
                                             <spontaneous>
[1]    100.0    0.08  575.48                 main [1]
              575.33    0.00      11/11          matrixMultiply1 [2]
                0.15    0.00      22/22          fillMatrix [3]
                0.00    0.00      33/33          allocateMatrix [4]
                0.00    0.00      33/33          freeMatrix [5]
                0.00    0.00       1/1           writeResultsToCSV [6]
-----------------------------------------------
              575.33    0.00      11/11          main [1]
[2]    100.0  575.33    0.00      11           matrixMultiply1 [2]
-----------------------------------------------
                0.15    0.00      22/22          main [1]
[3]      0.0    0.15    0.00      22           fillMatrix [3]
-----------------------------------------------
                0.00    0.00      33/33          main [1]
[4]      0.0    0.00    0.00      33           allocateMatrix [4]
-----------------------------------------------
                0.00    0.00      33/33          main [1]
[5]      0.0    0.00    0.00      33           freeMatrix [5]
-----------------------------------------------
                0.00    0.00       1/1           main [1]
[6]      0.0    0.00    0.00       1           writeResultsToCSV [6]
-----------------------------------------------
```

Figure 1: Profiling Information for Problem - A1

2.4 Performance

2.4.1 Performance in Lab PC

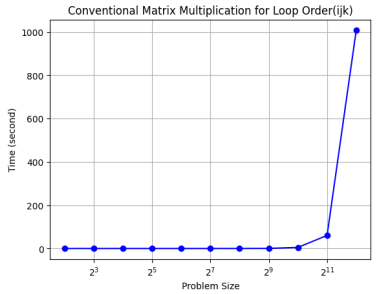


Figure 2: Time vs Problem Size on Lab PC for Loop Order(ijk)

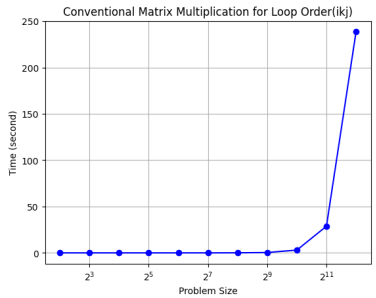


Figure 3: Time vs Problem Size on Lab PC for Loop Order(ikj)

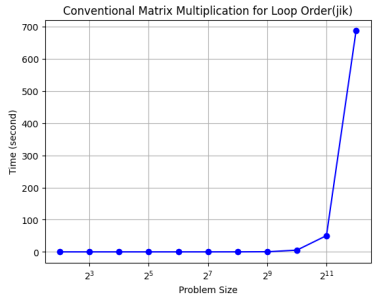


Figure 4: Time vs Problem Size on Lab PC for Loop Order(jik)

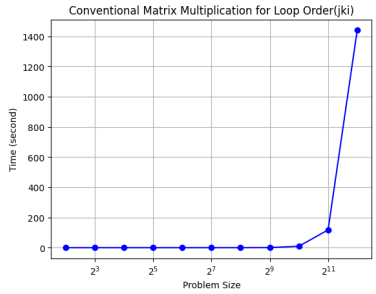


Figure 5: Time vs Problem Size on Lab PC for Loop Order(jki)

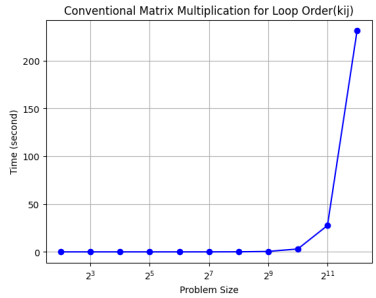


Figure 6: Time vs Problem Size on Lab PC for Loop Order(kij)

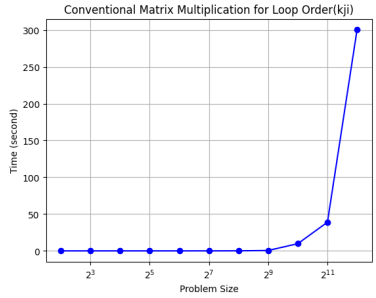


Figure 7: Time vs Problem Size on Lab PC for Loop Order(kji)

2.4.2 Performance in HPC Cluster

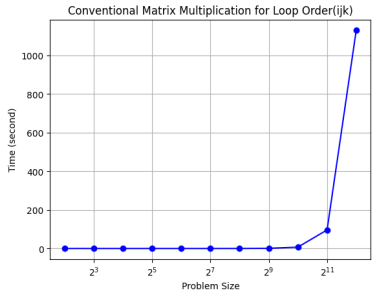


Figure 8: Time vs Problem Size on HPC Cluster for Loop Order(ijk)

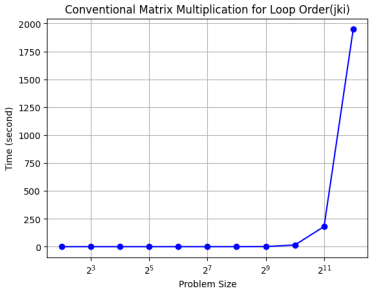


Figure 11: Time vs Problem Size on HPC Cluster for Loop Order(jki)

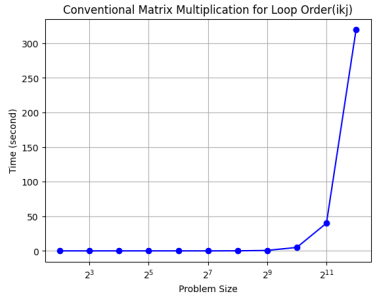


Figure 9: Time vs Problem Size on HPC Cluster for Loop Order(ikj)

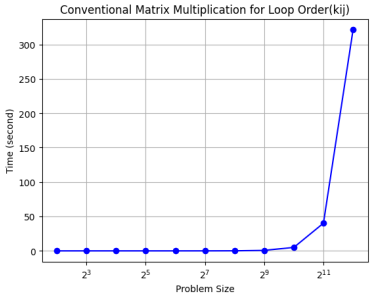


Figure 12: Time vs Problem Size on HPC Cluster for Loop Order(kij)

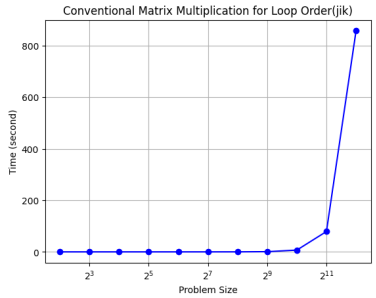


Figure 10: Time vs Problem Size on HPC Cluster for Loop Order(jik)

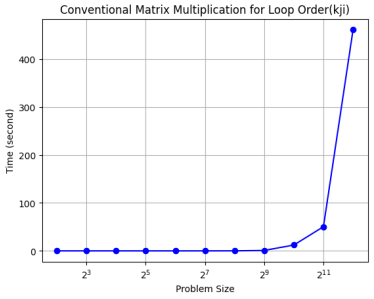


Figure 13: Time vs Problem Size on HPC Cluster for Loop Order(kji)

3 Problem B-1: Matrix Multiplication Using Transpose

3.1 Description

Optimizing matrix multiplication by transposing one of the input matrices improves memory locality and reduces cache misses. The optimized formula is:

$$C_{i,j} = \sum_{k=0}^{n-1} A_{i,k} B_{j,k}^{T} \tag{3}$$

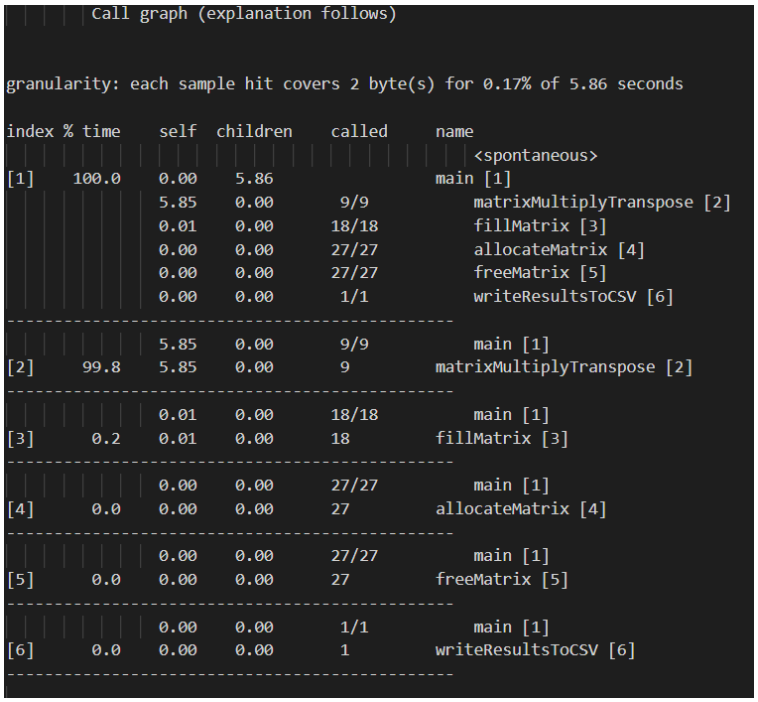
3.2 Complexity Analysis (Serial Implementation)

The time complexity remains:

$$O(n^3) \tag{4}$$

but the constant factors are improved due to better cache utilization. The space complexity is also $O(n^2)$, as it requires storing the transpose of matrix B in addition to the input matrices.

3.3 Profiling Information



```
     Call graph (explanation follows)


granularity: each sample hit covers 2 byte(s) for 0.17% of 5.86 seconds

index % time    self  children    called     name
                                             <spontaneous>
[1]    100.0    0.00    5.86                 main [1]
                5.85    0.00       9/9           matrixMultiplyTranspose [2]
                0.01    0.00      18/18          fillMatrix [3]
                0.00    0.00      27/27          allocateMatrix [4]
                0.00    0.00      27/27          freeMatrix [5]
                0.00    0.00       1/1           writeResultsToCSV [6]
-----------------------------------------------
                5.85    0.00       9/9           main [1]
[2]     99.8    5.85    0.00       9         matrixMultiplyTranspose [2]
-----------------------------------------------
                0.01    0.00      18/18          main [1]
[3]      0.2    0.01    0.00      18         fillMatrix [3]
-----------------------------------------------
                0.00    0.00      27/27          main [1]
[4]      0.0    0.00    0.00      27         allocateMatrix [4]
-----------------------------------------------
                0.00    0.00      27/27          main [1]
[5]      0.0    0.00    0.00      27         freeMatrix [5]
-----------------------------------------------
                0.00    0.00       1/1           main [1]
[6]      0.0    0.00    0.00       1         writeResultsToCSV [6]
-----------------------------------------------
```

Figure 14: Profiling Information for Problem - B1

3.4 Performance

3.4.1 Performance in Lab PC

- Plot of problem size versus time for various values of n.

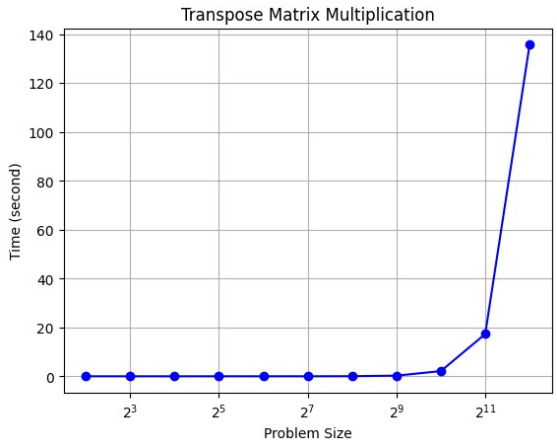


Figure 15: Time vs Problem Size on Lab PC

3.4.2 Performance in HPC Cluster

- Plot of problem size versus time for various values of n.

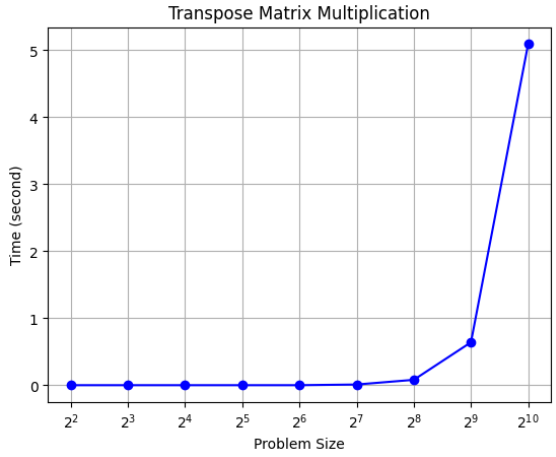


Figure 16: Time vs Problem Size on HPC Cluster

4 Problem C-1: Block Matrix Multiplication (Divide and Conquer Strategy)

4.1 Description

Block matrix multiplication is a divide-and-conquer strategy that partitions the input matrices into smaller blocks. The multiplication is then performed on these blocks, which improves memory locality and cache efficiency. The formula for block matrix multiplication is:

$$C = \begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix} = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \times \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}$$

Each block multiplication is computed recursively until the blocks are small enough to be multiplied directly. This approach is particularly effective in high-performance computing, as it allows for parallel processing and improved cache utilization.

4.2 Time Complexity

Using the recurrence relation:

$$T(n) = 8T(n/2) + O(n^2) \tag{5}$$

we obtain:

$$O(n^3) \tag{6}$$

which remains same as conventional matrix multiplication but the divide-and-conquer approach improves cache utilization, leading to better practical performance. The space complexity is also $O(n^2)$, as it requires storing the input matrices and the resultant matrix.

4.3 Profiling Information

```
Call graph (explanation follows)


granularity: each sample hit covers 2 byte(s) for 0.15% of 6.54 seconds

index % time    self  children    called     name
                                             <spontaneous>
[1]    100.0    0.00    6.54                 main [1]
                6.53    0.00       9/9           blockMatrixMultiply [2]
                0.01    0.00      18/18          fillMatrix [3]
                0.00    0.00      27/27          allocateMatrix [4]
                0.00    0.00      27/27          freeMatrix [5]
                0.00    0.00       1/1           writeResultsToCSV [6]
-----------------------------------------------
                6.53    0.00       9/9           main [1]
[2]     99.8    6.53    0.00       9         blockMatrixMultiply [2]
-----------------------------------------------
                0.01    0.00      18/18          main [1]
[3]      0.2    0.01    0.00      18         fillMatrix [3]
-----------------------------------------------
                0.00    0.00      27/27          main [1]
[4]      0.0    0.00    0.00      27         allocateMatrix [4]
-----------------------------------------------
                0.00    0.00      27/27          main [1]
[5]      0.0    0.00    0.00      27         freeMatrix [5]
-----------------------------------------------
                0.00    0.00       1/1           main [1]
[6]      0.0    0.00    0.00       1         writeResultsToCSV [6]
-----------------------------------------------
```

Figure 17: Profiling Information for Problem - C1

4.4 Performance

4.4.1 Performance in Lab PC

- Plot of problem size versus time for various values of n.

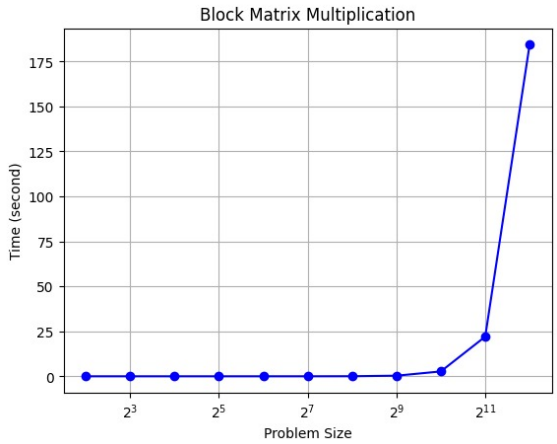


Figure 18: Time vs Problem Size on Lab PC

4.4.2 Performance in HPC Cluster

- Plot of problem size versus time for various values of n.

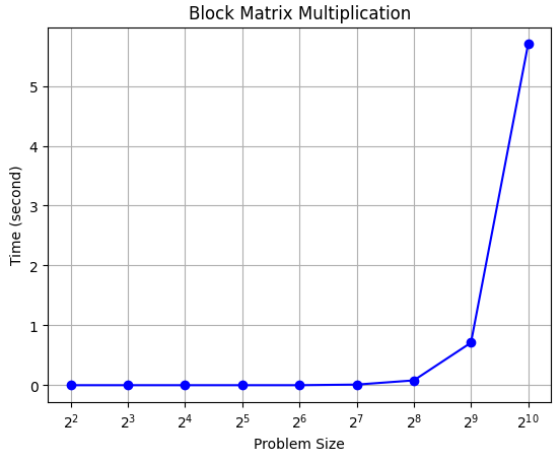


Figure 19: Time vs Problem Size on HPC Cluster