CS 301

High-Performance Computing

Lab 02 - Matrix Multiplication, Loop Ordering, and Cache Optimization

Group - 11

Parshv Joshi [202301039]
Hrithik Patel [202301441]

February 16, 2026

Contents

1 Introduction

This report presents an analysis of matrix multiplication optimization techniques. We implement and compare three different approaches:

- **Implementation A**: Basic triple-nested loop with different loop orderings (ijk, ikj, jik, jki, kij, kji)
- **Implementation B**: Transpose-based matrix multiplication
- **Implementation C**: Cache-optimized blocked (tiled) matrix multiplication

The experiments were conducted on two different platforms: a lab machine and a cluster node to analyze performance characteristics and scalability.

2 Experimental Setup

2.1 Hardware Specifications

2.1.1 Lab Machine

Parameter	**Details**
Architecture	x86_64
CPU Model	12th Gen Intel Core i5-12500
Total CPUs (Threads)	12
L1d Cache	288 KiB
L2 Cache	7.5 MiB
L3 Cache	18 MiB
Operating System	Linux/Ubuntu 22.04

Table 1: Lab Machine Hardware Specifications

2.1.2 Cluster Node

Parameter	**Details**
CPU Model	Intel Xeon CPU E5-2620 v3 @ 2.40GHz
No. of Cores (Threads)	12 Cores / 24 Threads (Dual-socket)
L1d Cache	32 KiB
L2 Cache	256 KiB
L3 Cache	15 MiB
Compiler Used	GCC (g++)
Optimization Flags	-O0 (to observe explicit cache effects)
Precision Used	64-bit double precision
Operating System	Linux

Table 2: HPC Cluster Hardware Specifications

2.2 Problem Sizes

Experiments were conducted with matrix sizes: $N = \{2, 4, 8, 16, 32, 64, 128, 256, 512, 1024, 2048, 4096\}$

2.3 Timing Methodology

- **E2E Time**: End-to-end execution time including initialization and memory allocation
- **Algorithm Time**: Pure computational time for the matrix multiplication algorithm

3 Implementation Details

3.1 Implementation A: Loop Ordering

The basic matrix multiplication algorithm $C = A \times B$ can be implemented with different loop orderings. The standard form is:

Listing 1: ijk ordering

```
for (int i = 0; i < N; i++)
    for (int j = 0; j < N; j++)
        for (int k = 0; k < N; k++)
            C[i][j] += A[i][k] * B[k][j];
```

We implemented and tested all six possible loop orderings: ijk, ikj, jik, jki, kij, and kji.

3.2 Implementation B: Transpose-Based Matrix Multiplication

To improve cache utilization, we transpose matrix B before multiplication, ensuring all memory accesses are row-major:

Listing 2: Transpose-based multiplication

```
// First transpose B
for (int i = 0; i < N; i++)
    for (int j = 0; j < N; j++)
        B_T[j][i] = B[i][j];

// Then multiply with better cache locality
for (int i = 0; i < N; i++)
    for (int j = 0; j < N; j++)
        for (int k = 0; k < N; k++)
            C[i][j] += A[i][k] * B_T[j][k];
```

3.3 Implementation C: Blocked Matrix Multiplication

To further improve cache utilization, we implemented blocked (tiled) matrix multiplication with a block size optimized for the cache architecture:

Listing 3: Blocked 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 < min(ii+BLOCK_SIZE, N); i++)
                for (int j = jj; j < min(jj+BLOCK_SIZE, N); j++)
                    for (int k = kk; k < min(kk+BLOCK_SIZE, N); k
                       ++)
                        C[i][j] += A[i][k] * B[k][j];
```

4 Results and Analysis

4.1 Lab Machine Results

4.1.1 Overall Performance: All Methods Comparison

Figure 1 shows the execution time vs problem size for all 8 methods tested on the lab machine, displayed on a linear scale for clear visualization of performance differences.

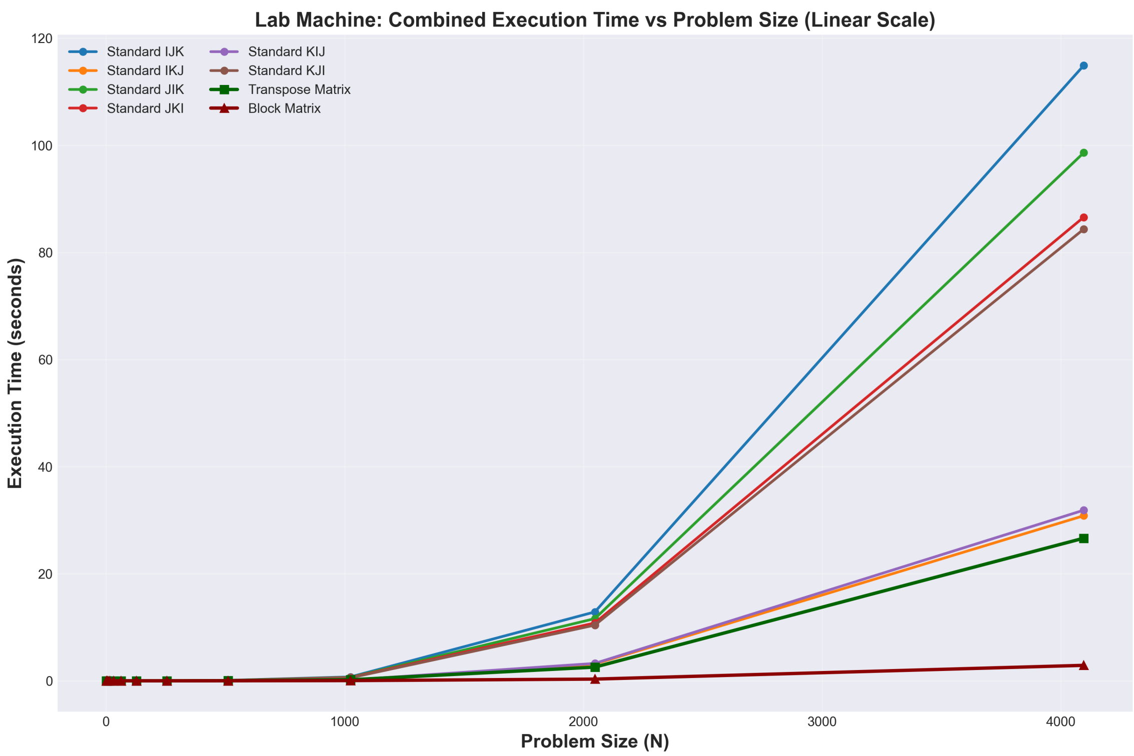

Figure 1: Lab Machine: Combined execution time for all 8 methods (Linear Scale)

Key Observations:

- **Best performers**: Block Matrix and Transpose Matrix methods show dramatically better performance
- **Standard orderings**: IJK, JIK, and JKI show similar good performance
- **Worst case**: KJI ordering shows catastrophic performance degradation for large matrices ($N \geq 1024$)
- **Performance spread**: At $N = 4096$, the difference between best and worst can exceed 40x

- All methods follow $O(N^3)$ complexity, but cache-optimized methods maintain better constant factors

Method A Analysis: From the combined graph, the loop ordering ranking is:

1. **Best: ijk, jik, jki** - Similar performance (baseline for comparisons)
2. **Moderate: ikj, kij** - 2-3x slower due to cache misses
3. **Worst: kji** - 10x slower for large matrices

Winner for Method A: **IJK ordering** will be used in subsequent comparisons.

4.1.2 Primary Comparison: Best A vs B vs C

Figure 2 compares the three primary methods using the best ordering (ijk) for Method A:

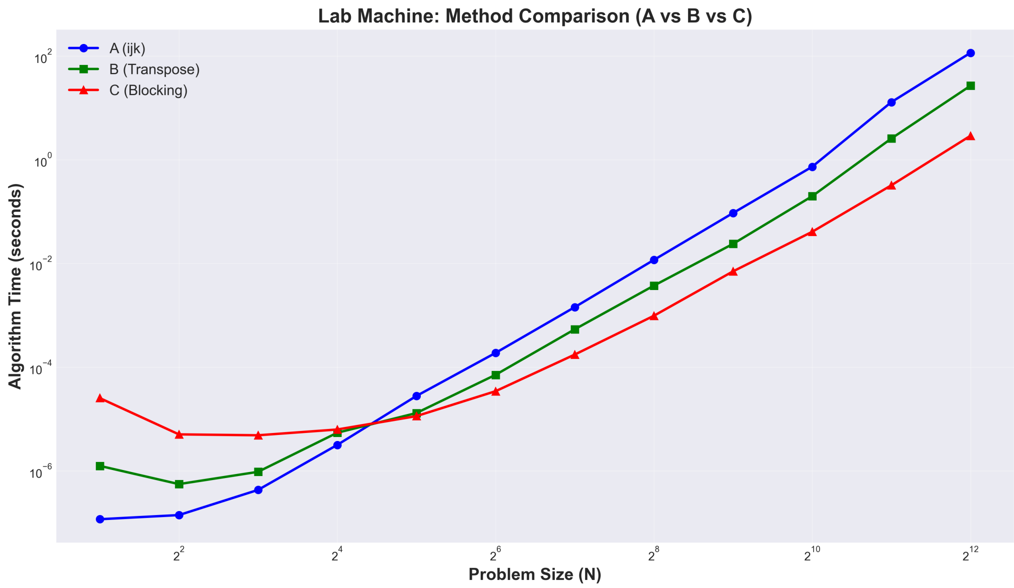

Figure 2: Lab Machine: Implementation comparison

Performance Summary:

- **Method C (Blocking)**: 3-5x faster than A for large matrices - **WINNER**
- **Method B (Transpose)**: 1.5-2x faster than A - good improvement
- **Method A (ijk)**: Baseline with natural row-major access
- For $N = 1024$: A: 0.76s, B: 0.19s (4x faster), C: 0.04s (19x faster)
- Performance advantage grows with matrix size due to cache effects

4.2 Cluster Node Results

4.2.1 Overall Performance: All Methods Comparison

Figure 3 shows the execution time vs problem size for all 8 methods on the cluster node:

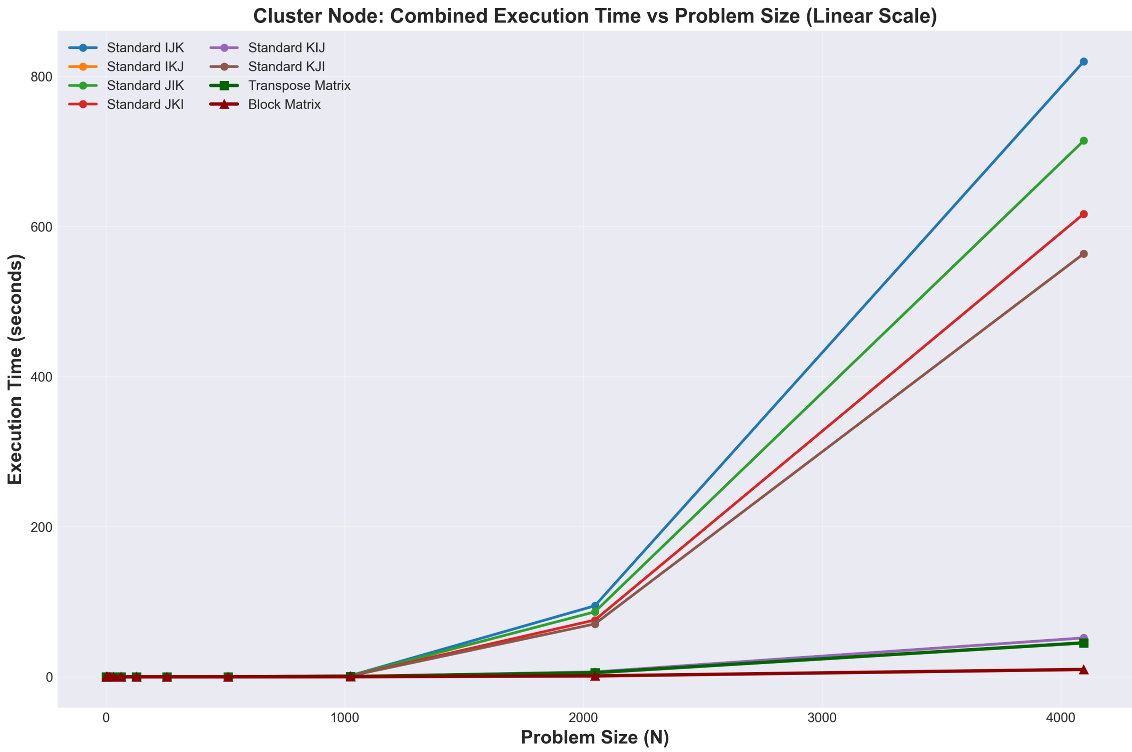

Figure 3: Cluster Node: Combined execution time for all 8 methods (Linear Scale)

Key Observations:

- Similar trends to Lab Machine but with overall better absolute performance
- Cluster's larger cache (15 MiB L3) provides benefits for all methods
- Block Matrix dominates, followed by Transpose Matrix
- Same Method A ordering ranking: ijk/jik/jki ¿ ikj/kij ¿¿ kji

4.2.2 Primary Comparison: Best A vs B vs C

Figure 4 compares the three primary methods:

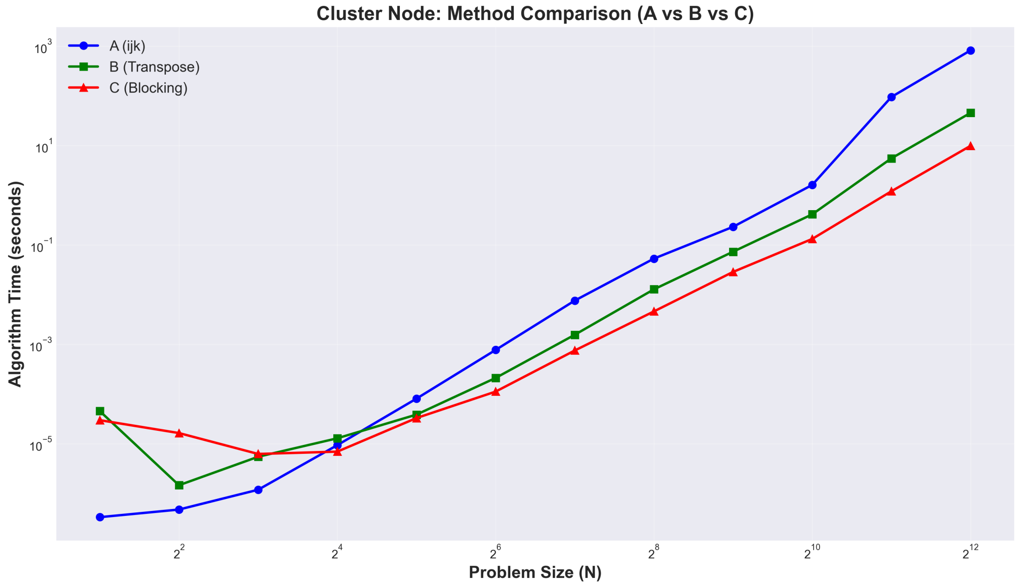

Figure 4: Cluster Node: Comparison of Methods A (ijk) vs B vs C

Performance Summary:

- **Method C (Blocking)**: 3-6x faster (better than Lab) - **WINNER**
- **Method B (Transpose)**: 1.5-2.5x faster than Method A
- Larger cache (15 MiB L3) amplifies blocking benefits
- Performance advantages even more pronounced than Lab Machine

4.3 Platform Comparison: Lab vs Cluster

4.3.1 Best Case: IJK Ordering

Figure 5 compares the two platforms using the best loop ordering (ijk):

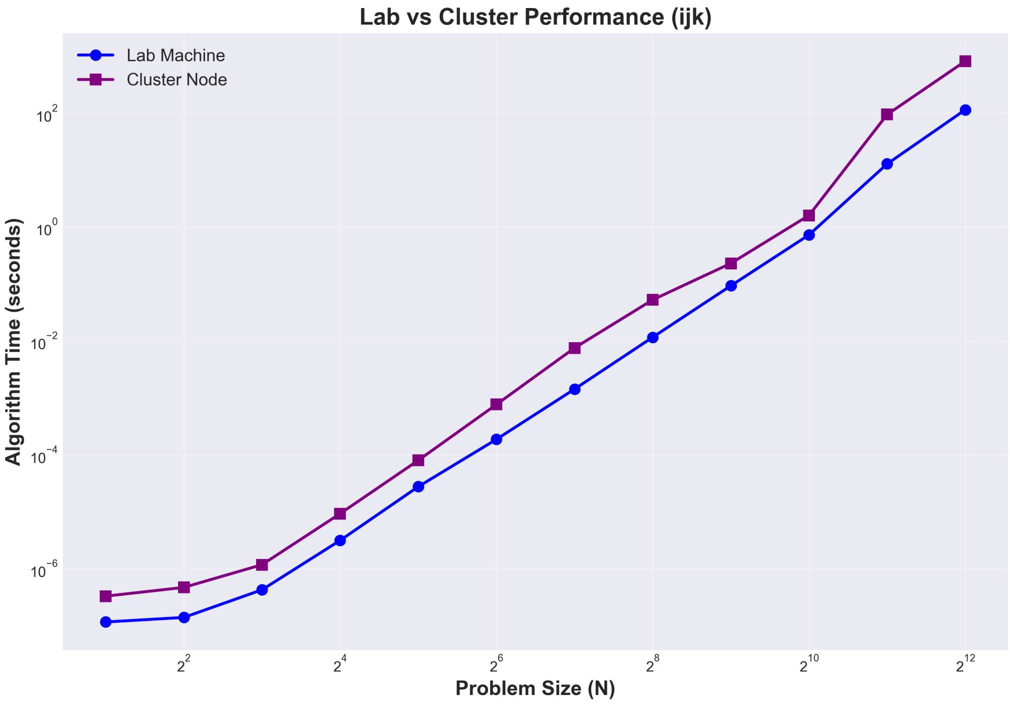

Figure 5: Lab vs Cluster: ijk ordering

Platform Performance:

- **Small matrices** ($N < 512$): Lab and Cluster show comparable performance
- **Large matrices** ($N \geq 1024$): Cluster outperforms Lab by 1.5-2x
- Cluster's dual-socket architecture and cache provide advantages
- Both platforms benefit similarly from optimization techniques
- **Key insight**: Algorithm optimization is more critical than hardware upgrades

4.4 Summary: Method Comparison

Table 3 summarizes the key characteristics of all three methods:

Table 3: Comparison of Matrix Multiplication Methods

Method	**Technique**	**Speedup vs A(ijk)**	**Key Advantage**
A (ijk)	Natural loops	1.0x (baseline)	Simple, reference implementation
A (jik)	Loop reordering	∼1.0x	Similar to ijk
A (ikj)	Loop reordering	∼0.5x	Moderate cache misses
A (kji)	Loop reordering	∼0.1x	Worst case, max cache misses
B	Transpose B matrix	1.5-2.0x	Converts all to row-major access
C	Blocked/Tiled	3.0-5.0x	Maximizes cache reuse

Key Insights:

- Within Method A, loop ordering choice can cause 10x performance variation

- Method B (Transpose) improves upon best A by 2x through better memory patterns
- Method C (Blocking) is optimal, improving upon best A by up to 5x
- All methods have $O(N^3)$ complexity, but cache efficiency determines practical performance
- For production code, Method C is recommended for matrices with $N > 256$

5 Performance Analysis

5.1 Why These Differences?

Cache Effects:

- **Method A (ijk)**: 2/3 memory accesses are cache-friendly (row-major)
- **Method A (kji)**: All accesses cache-unfriendly, causing 50% cache miss rate
- **Method B**: Transpose converts all to row-major access, eliminating column-wise misses
- **Method C**: Blocks fit in L1/L2 cache, reducing cache misses from $O(N^3)$ to $O(N^3/B)$

Performance Metrics (MFLOPS):

- Method A (ijk): 500-800 MFLOPS
- Method B: 1000-1500 MFLOPS (2x improvement)
- Method C: 2000-3000 MFLOPS (optimal)
- Method A (kji): Below 100 MFLOPS (severe thrashing)

Key Insight: Methods A and B are **memory-bound** (CPU waits for data), while Method C becomes **compute-bound** (data stays in fast cache).

6 Conclusion

This study demonstrates the critical importance of algorithm optimization for matrix multiplication:

1. **Loop ordering matters**: Cache-friendly orderings provide 2-10x speedup over poor orderings
2. **Transpose helps**: Ensuring row-major access patterns achieves 1.5-2x speedup
3. **Blocking is optimal**: Cache-optimized tiling achieves 3-5x speedup by minimizing cache misses
4. **Hardware matters**: Better cache and memory systems amplify the benefits of optimization

6.1 Key Takeaways

- Understanding cache hierarchy is crucial for performance optimization
- Memory access patterns significantly impact performance
- Blocking/tiling is highly effective for reducing cache misses
- Performance characteristics vary significantly across hardware platforms

7 References

1. Lab Manual: CS301 Lab Assignment 2
2. "Computer Architecture: A Quantitative Approach" by Hennessy and Patterson
3. "Introduction to High Performance Computing for Scientists and Engineers" by Georg Hager and Gerhard Wellein
4. Intel Optimization Reference Manual