CS 301
High-Performance Computing

Lab 3 - Matrix Multiplication Performance

Rakshit Pandhi (202201426)
Kalp Shah (202201457)

February 26, 2025

Contents

1 Problem A - Conventional matrix multiplication

1.1 Brief description of the problem

The problem of conventional matrix multiplication with loop interchange involves reordering the three nested loops in the standard algorithm, resulting in six possible execution orders. Each ordering affects cache efficiency, memory access patterns, and overall performance.

1.2 Description of Algorithm

1. **Initialization**: Start with an empty result matrix C (typically initialized to zero).
2. **Outer Loop (i-loop)**: Iterates over the rows of matrix C.
3. **Middle Loop (j-loop)**: Iterates over the columns of matrix C.
4. **Inner Loop (k-loop)**: Performs element-wise multiplication and accumulation to compute each entry $C[i][j]$.

$$C[i][j] = \sum_{k=0}^{N-1} A[i][k] \times B[k][j] \tag{1}$$

This is the first and the basic possibility of i-j-k loop order. We can interchange the order of the as described in the question and also get to the desired answer.

1.3 Complexity of the Algorithm

The algorithm consists of three nested loops, each iterating N times. The number of operations performed (multiplications and additions) is proportional to:

$$N \times N \times N = N^3 \tag{2}$$

Thus, the time complexity is:

$$O(N^3) \tag{3}$$

1.4 Compute to Memory Access Ratio

For each element to be calculated in C we perform N multiplications and N additions, making 2N operations and there are N^2 such elements thus flops are $2N^3$. And in total there are $3N^2$ elements making compute to memory access ratio as $2N/3$.

1.5 Memory Bound versus Compute Bound

From calculation we can see that the CMA increases as N increases and also if we see the order and compare computations performed are more than the memory accesess.

1.6 Hardware Details

1.6.1 Hardware Details for LAB207 PCs

- Architecture: x86_64
- CPU op-mode(s): 32-bit, 64-bit
- Byte Order: Little Endian
- CPU(s): 12
- On-line CPU(s) list: 0-3
- Thread(s) per core: 2
- Core(s) per socket: 6
- Socket(s): 1
- NUMA node(s): 1
- Vendor ID: GenuineIntel
- CPU family: 6
- Model: 60
- Model name: Intel Core i5-12500
- Stepping: 3
- CPU MHz: 799.992
- CPU max MHz: 4.6G
- CPU min MHz: 800.0000
- BogoMIPS: 6584.55
- Virtualization: VT-x
- L1d cache: 288K
- L1i cache: 192K
- L2 cache: 7.5M
- L3 cache: 18M
- NUMA node0 CPU(s): 0-3

1.6.2 Hardware Details for HPC Cluster

- 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 E5-2620 v3 @ 2.40GHz
- Stepping: 2
- CPU MHz: 1419.75
- 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

1.7 Input Parameters,Output,Accuracy

1. **Matrix** A: A 2D array of size $N \times N$, representing the first matrix.
2. **Matrix** B: A 2D array of size $N \times N$, representing the second matrix.
3. **Matrix Size** N: An integer representing the dimension of the square matrices.

Output The result is a matrix C of size $N \times N$, where each element is computed as:

$$C[i][j] = \sum_{k=0}^{N-1} A[i][k] \times B[k][j] \tag{4}$$

Accuracy

- Accuracy is proportional to the number of computations performed.
- Therefore for smaller n accuracy may come low and vice versa.
- We can measure the accuracy by calculating the error if we are known with the exact value of the final matrix.

1.8 Algorithm Time(serial) versus Problem Size

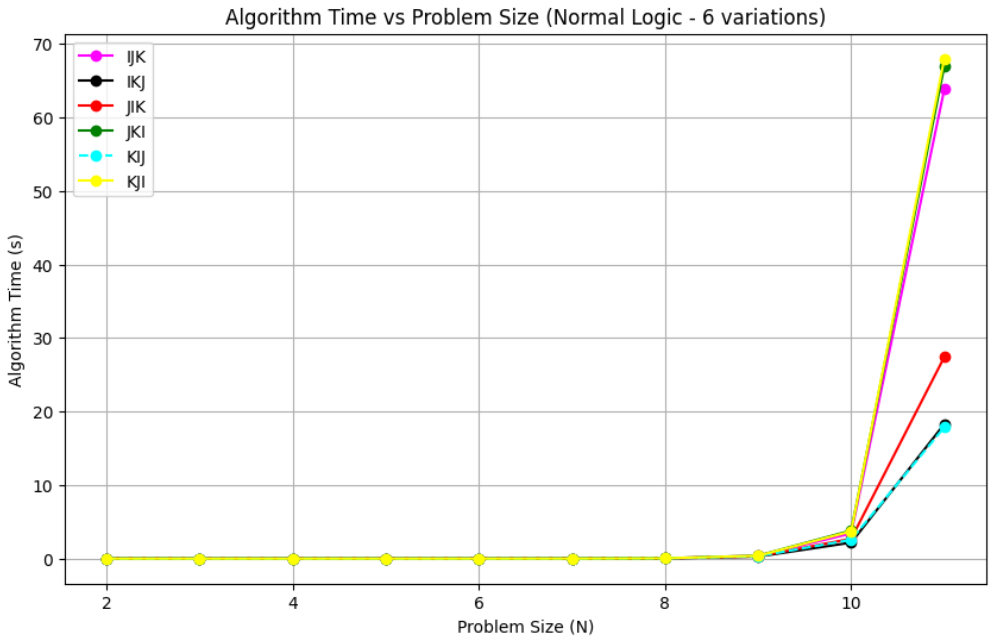

Figure 1: Screenshot from LAB207 PC

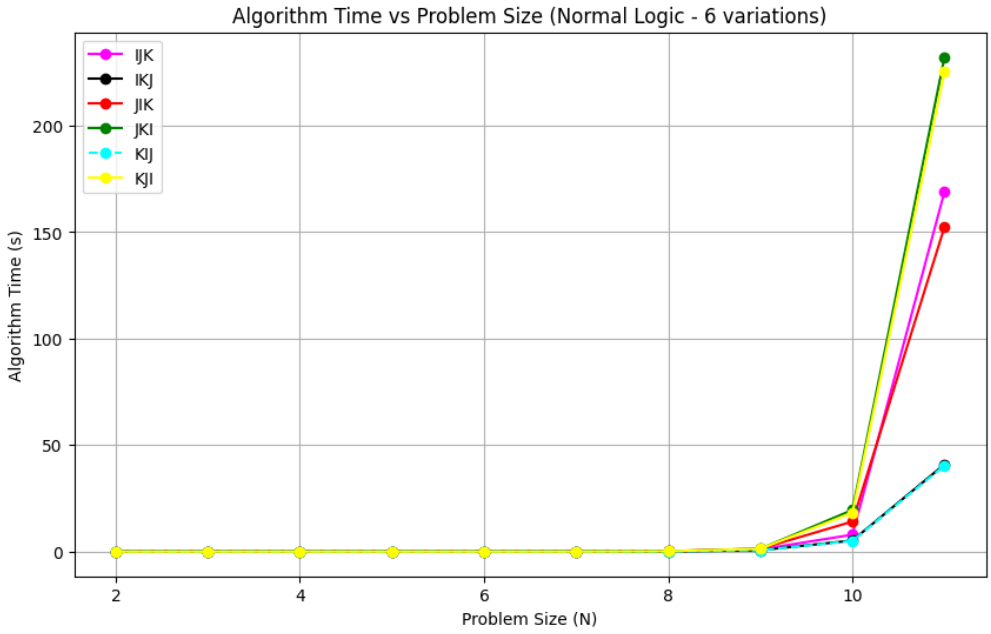

Figure 2: Screenshot from HPC Cluster

1.9 Inference

From the graphs we can see that HPC Cluster takes more time than Lab PC because of the relatively smaller cache size of HPC Cluster. Moreover, the variant I-K-J performs the best out of the all 6 variants because of the row major property of C++. In this variant all the 3 loops run in a row major fashion making cache hit more efficient and improving the algorithm time.

2 Problem B - Matrix multiplication using transpose

2.1 Brief description of the problem

The problem involves computing the product of the 2 matrices by transposing one of the matrices to exploit the property of row major so that the cache hit efficiency is increased.

$$C_{m\times p} = A_{m\times n} \times B^T_{n\times p}$$

2.2 Description of Algorithm

If A is of size $(m \times n)$ and B is of size $(p \times n)$, then:

- The transpose of B, denoted as B^T, will have a size of $(n \times p)$.
- The resulting matrix $C = A \times B^T$ will have a size of $(m \times p)$.

Computation Formula: Each element of C is calculated as:

$$C[i][j] = \sum_{k=0}^{n-1} A[i][k] \times B[j][k]$$

(since $B^T[j][k] = B[k][j]$)

2.3 Complexity of the Algorithm

The algorithm consists of three nested loops, each iterating N times. The number of operations performed (multiplications and additions) is proportional to:

$$N \times N \times N = N^3 \tag{5}$$

Thus, the time complexity is:

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

2.4 Compute to Memory Access Ratio

For each element to be calculated in C we perform N multiplications and N additions, making 2N operations and there are N^2 such elements thus flops are $2N^3$. And in total there are $3N^2$ elements making compute to memory access ratio as $2N/3$.

2.5 Memory Bound versus Compute Bound

From calculation we can see that the CMA increases as N increases and also if we see the order and compare computations performed are more than the memory accesses, hence compute bound.

2.6 Hardware Details

2.6.1 Hardware Details for LAB207 PCs

- Architecture: x86_64
- CPU op-mode(s): 32-bit, 64-bit
- Byte Order: Little Endian
- CPU(s): 12
- On-line CPU(s) list: 0-3
- Thread(s) per core: 2
- Core(s) per socket: 6
- Socket(s): 1
- NUMA node(s): 1
- Vendor ID: GenuineIntel

- CPU family: 6
- Model: 60
- Model name: Intel Core i5-12500
- Stepping: 3
- CPU MHz: 799.992
- CPU max MHz: 4.6G
- CPU min MHz: 800.0000
- BogoMIPS: 6584.55
- Virtualization: VT-x
- L1d cache: 288K
- L1i cache: 192K
- L2 cache: 7.5M
- L3 cache: 18M
- NUMA node0 CPU(s): 0-3

2.6.2 Hardware Details for HPC Cluster

- 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 E5-2620 v3 @ 2.40GHz

- Stepping: 2
- CPU MHz: 1419.75
- 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

2.7 Input Parameters,Output,Accuracy

Input Parameters

- A: $m \times n$ matrix
- B: $p \times n$ matrix
- m, n, p: Dimensions of matrices

Output

- C: $m \times p$ matrix, computed as:

$$C[i][j] = \sum_{k=0}^{n-1} A[i][k] \times B[j][k]$$

Accuracy Check

- Compare with direct multiplication:

$$C_{\text{expected}} = A \times B^T$$

2.8 Algorithm Time(serial) versus Problem Size

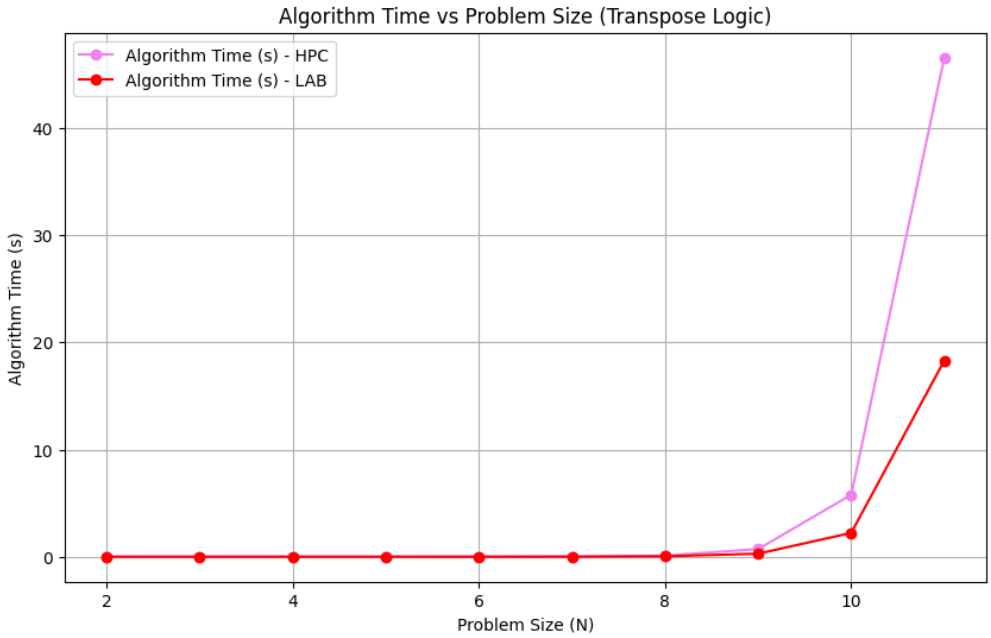

Figure 3: Screenshot from LAB PC

2.9 Inference

We can see that HPC cluster takes more time than LAB PC because of the smaller cache size it has. Moreover, we can also compare this plot with the previous plot and see that these 2 plots are very similar to the i-k-j plot because ultimately transpose way of multiplication is the variant i-k-j for the matrix multiplication.

3 Problem C-Block matrix multiplication

3.1 Brief description of the problem

Block matrix multiplication is an optimized approach to multiplying large matrices by dividing them into smaller submatrices (blocks). This technique improves cache efficiency and can leverage parallel processing.

3.2 Description of Algorithm

Divide Given matrices A $(m \times n)$ and B $(n \times p)$, divide them into smaller blocks (submatrices) of size $b \times b$, where b is a predefined block size.

For example, split matrix A into submatrices:

$$A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}$$

Similarly, split B into blocks:

$$B = \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}$$

Multiply Using Blocks Compute each block of the result matrix C using:

$$C_{ij} = \sum_k A_{ik} \times B_{kj}$$

Each C_{ij} is computed using smaller matrix multiplications.

Combine The resulting submatrices are combined to form the final output matrix C:

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

3.3 Complexity of the Algorithm

Each level of recursion does b^3 multiplications, and the recursion continues until matrices are small enough.

This results in an overall complexity of:

$$O(N^3)$$

(same as standard multiplication). However, better memory access patterns make it faster in practice.

3.4 Compute to Memory Access Ratio

For matrix multiplication of size $N \times N$, the total number of multiplications and additions required is:

$$\text{FLOPs} = 2N^3$$

Blocked multiplication reuses elements within cache, reducing memory transfers. The total memory traffic (loads/stores) is approximately:

$$\text{Memory Accesses} = N^2$$

Therefore the CMA is approximately 2N.

3.5 Memory Bound versus Compute Bound

Blocked matrix multiplication is **compute-bound** for large matrices because the ratio of computations to memory accesses is high. However, for small matrices that do not fully utilize the CPU, it may become **memory-bound**.

3.6 Hardware Details

3.6.1 Hardware Details for LAB207 PCs

- Architecture: x86_64
- CPU op-mode(s): 32-bit, 64-bit
- Byte Order: Little Endian
- CPU(s): 12
- On-line CPU(s) list: 0-3
- Thread(s) per core: 2
- Core(s) per socket: 6
- Socket(s): 1
- NUMA node(s): 1
- Vendor ID: GenuineIntel
- CPU family: 6
- Model: 60
- Model name: Intel Core i5-12500
- Stepping: 3
- CPU MHz: 799.992
- CPU max MHz: 4.6G
- CPU min MHz: 800.0000
- BogoMIPS: 6584.55
- Virtualization: VT-x
- L1d cache: 288K
- L1i cache: 192K
- L2 cache: 7.5M
- L3 cache: 18M
- NUMA node0 CPU(s): 0-3

3.6.2 Hardware Details for HPC Cluster

- 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 E5-2620 v3 @ 2.40GHz
- Stepping: 2
- CPU MHz: 1419.75
- 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

3.7 Input Parameters,Output,Accuracy

3.8 Algorithm Time(serial) versus Problem Size

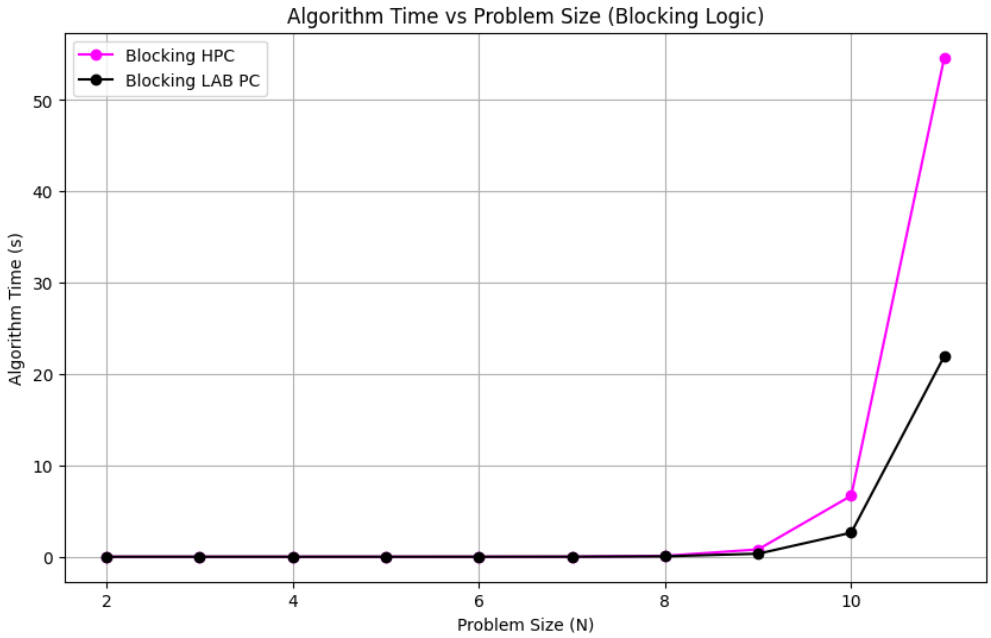

Figure 4: Screenshot from LAB PC

3.9 Inference

First of all for LAB PC blocksize has been kept 32 which came by solving

$$24 \cdot b^2 < \text{Cache size}$$

This blocksize gave the smallest time. Also, for HPC Cluster the optimal blocksize came around 16 and was kept this and plotted. Here also we can see that LAB-PC performs better than HPC Cluster due to bigger cache size .

4 Conclusions

- In this lab we performed matrix multiplication in several different ways and compared their running time.
- Firstly we used basic 3 nested loop technique with all the 6 variants and compared the time between them and understood the concept of locality and row-column major thing.
- We also used transpose technique which came out to be similar to the i-k-j variant due to exploitation of the row major property in C/C++.
- Lastly we performed Block multiplication which improves the cache reuse efficiency but still the i-k-j variant performed well due to row majorness.