

---

---

# CS 301

## High-Performance Computing

---

---

### Lab 3 - Comparison in Matrix Multiplication

---

Problem 1: Conventional matrix multiplication

Jay Dobariya (202101521)  
Akshar Panchani (202101522)

February 14, 2024

# Contents

<b>1</b>	<b>Introduction</b>	<b>3</b>
<b>2</b>	<b>Hardware Details</b>	<b>3</b>
2.1	Hardware Details for LAB207 PCs . . . . .	3
2.2	Hardware Details for HPC Cluster . . . . .	4
<b>3</b>	<b>Problem 1</b>	<b>4</b>
3.1	Brief description of the problem . . . . .	4
3.1.1	Algorithm description . . . . .	5
3.2	The complexity of the algorithm . . . . .	5
3.3	Profiling using HPC Cluster (with gprof) . . . . .	5
3.4	Graph of Problem Size vs Runtime . . . . .	6
3.4.1	Graph of Problem Size vs Runtime for LAB207 PCs . . . . .	6
3.4.2	Graph of Problem Size vs Runtime for HPC Cluster . . . . .	7
<b>4</b>	<b>Conclusions</b>	<b>7</b>

# 1 Introduction

A basic operation in linear algebra and many other computer tasks is matrix multiplication. There are a number of effective matrix multiplication techniques, each having pros and cons. Three approaches are known to us and we first discuss about the conventional method.

## 2 Hardware Details

### 2.1 Hardware Details for LAB207 PCs

- Architecture: x86\_64
- CPU op-mode(s): 32-bit, 64-bit
- Byte Order: Little Endian
- CPU(s): 6
- On-line CPU(s) list: 0-5
- Thread(s) per core: 1
- Core(s) per socket: 6
- Socket(s): 1
- NUMA node(s): 1
- Vendor ID: GenuineIntel
- CPU family: 6
- Model: 155
- Model name: Intel(R) Core(TM) i5-8500 CPU @ 3.00GHz
- Stepping: 10
- CPU MHz: 799.992
- CPU max MHz: 4100.0000
- CPU min MHz: 800.0000
- BogomIPS: 6000.00
- Virtualization: VT-x
- L1d cache: 192KB
- L1i cache: 192KB
- L2 cache: 1.5MB
- L3 cache: 9MB
- NUMA node0 CPU(s): 0-5

## 2.2 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: 2642.4378
- 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 1

### 3.1 Brief description of the problem

In traditional matrix multiplication, the products of related elements from the row of the first matrix and the column of the second matrix are added to determine each element of the final matrix.

### 3.1.1 Algorithm description

1. Initialize the resulting matrix to zeros.
2. For each row  $i$  of the first matrix and each column  $j$  of the second matrix:
  - (a) Compute the dot product of the  $i$ th row of the first matrix and the  $j$ th column of the second matrix.
  - (b) Assign the result to the corresponding element in the resulting matrix.

### 3.2 The complexity of the algorithm

The time complexity of conventional matrix multiplication is  $O(n^3)$ , where  $n$  is the size of the matrices. This is because there are  $n^2$  elements in the resulting matrix, and each element requires  $n$  multiplications and additions.

### 3.3 Profiling using HPC Cluster (with gprof)

The screenshots of profiling using the HPC Cluster are given below

```
granularity: each sample hit covers 2 byte(s) no time propagated

index % time    self  children   called    name
[1]      0.0      0.00    0.00      2/2      main [7]
-----
[1]      0.0      0.00    0.00      2      diff [1]
```

-----

This table describes the call tree of the program, and was sorted by the total amount of time spent in each function and its children.

Each entry in this table consists of several lines. The line with the index number at the left hand margin lists the current function. The lines above it list the functions that called this function, and the lines below it list the functions this one called. This line lists:

index	A unique number given to each element of the table. Index numbers are sorted numerically. The index number is printed next to every function name so it is easier to look up where the function is in the table.
% time	This is the percentage of the 'total' time that was spent in this function and its children. Note that due to different viewpoints, functions excluded by options, etc, these numbers will NOT add up to 100%.
self	This is the total amount of time spent in this function.
children	This is the total amount of time propagated into this function by its children.
called	This is the number of times the function was called. If the function called itself recursively, the number

Figure 1: Screenshot of the terminal from HPC Cluster

### 3.4 Graph of Problem Size vs Runtime

#### 3.4.1 Graph of Problem Size vs Runtime for LAB207 PCs

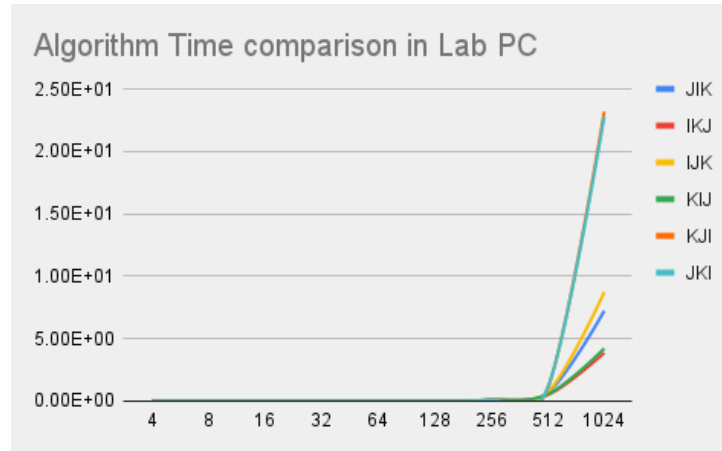


Figure 2: Total mean execution time (Algorithm time) vs Problem size plot for **problem size  $2^{10}$**  (Hardware: LAB207 PC, Problem: Conventional Matrix Multiplication).

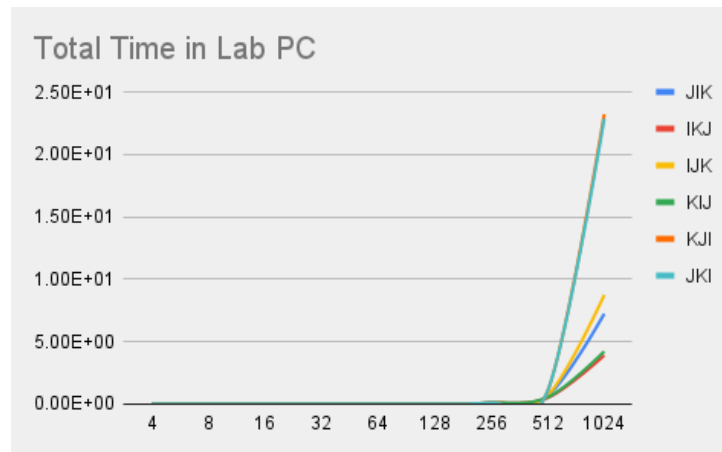


Figure 3: Total mean execution time (Algorithm time) vs Problem size plot for **problem size  $2^{10}$**  (Hardware: HPC Cluster, Problem: Conventional Matrix Multiplication).

### 3.4.2 Graph of Problem Size vs Runtime for HPC Cluster

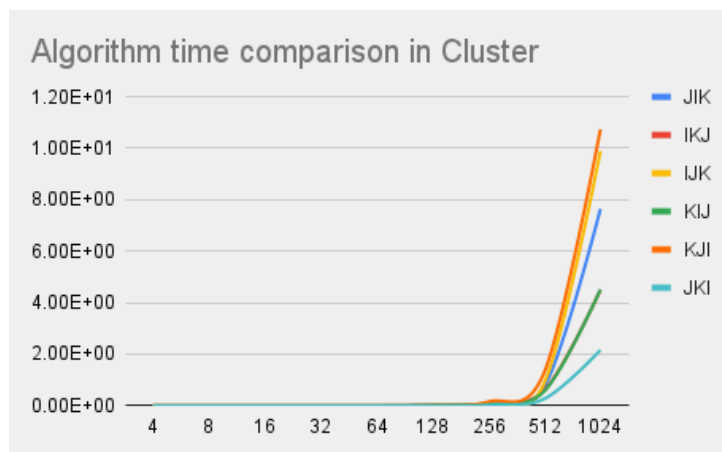


Figure 4: Total mean execution time (Algorithm time) vs Problem size plot for **problem size  $2^{10}$**  (Hardware: LAB207 PC, Problem: Conventional Matrix Multiplication).

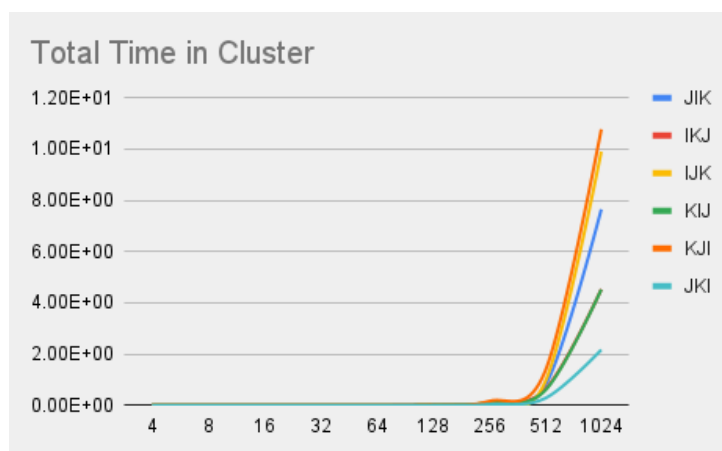


Figure 5: Total mean execution time (Algorithm time) vs Problem size plot for **problem size  $2^{10}$**  (Hardware: HPC Cluster, Problem: Conventional Matrix Multiplication).

## 4 Conclusions

- Conventional matrix multiplication is straightforward to implement and is suitable for small to medium-sized matrices. However, its time complexity grows cubically with the size of the matrices, limiting its scalability for large-scale computations.