Load Balancing Problem 2 report

submission date: 20.04.2025

Student ID	50251141
Student name	Hamidreza Rahimian

Contents

1 Introduction

1.1 Motivation

This project is a project to see usage of Load Balancing in real java project, given is a java script doing Matrix multiplication.

1.2 Goal

The Goal of the project is to use multi-threading via Load balancing methods and see if the execution time and performance will be better doing that

There are 2 different Load balancing methods that we can use there:

1. **Static load balancing based on block decomposition**
2. **Static load balancing based on cyclic decomposition**

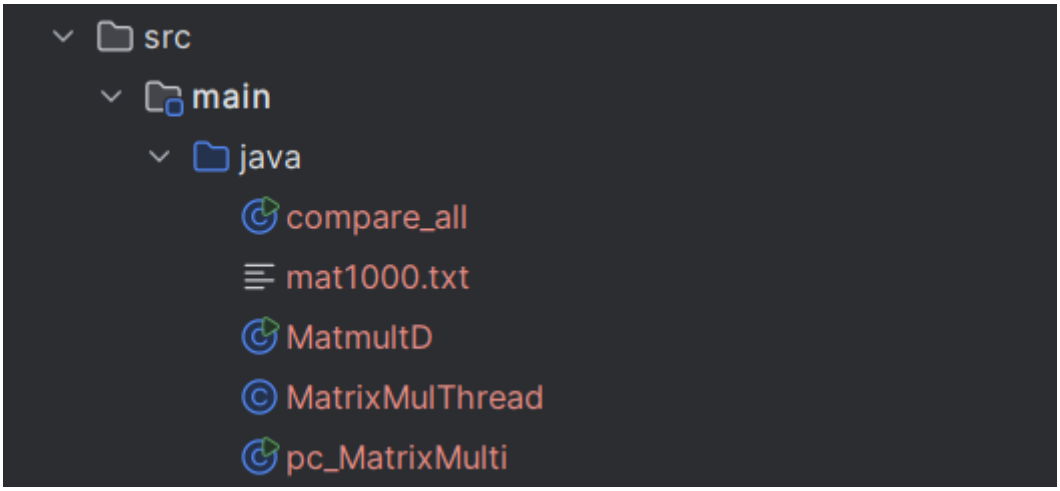
   FOR SOME REASON id like to continue this project with Block methods so lets do it with Block Method

1.3 Approach

Through this project, I aim make 3 java classes **MatrixMulThread** contains the thread , **pc_MatrixMulti** is doing the matrix multiplication and the last one **compare_all** is comparing the program with different numbers of Threads.

2 File organization

And this is how all files together look like, u can run MatrixmultiThread but u can call it



3 Expriment

3.1 CPU type

As I mentioned before I made 2 java classes in each directory, one of them has the general function (methods) for the type of load balancing that going to be used in that directory. For example, here there are 2 classes:

Device specifications

Device name	DELAA-25S9HX3
Full device name	DELAA-25S9HX3.emrsn.org
Processor	13th Gen Intel(R) Core(TM) i5-1345U 1.60 GHz
Installed RAM	16.0 GB (15.7 GB usable)
Device ID	80453336-5C1F-47BD-AEBE-128782DEB07B
Product ID	00330-80000-00000-AA180
System type	64-bit operating system, x64-based processor
Pen and touch	Pen and touch support with 10 touch points

After checking it online , I realized that my pc is not really quadcore or clock speed but Hybrid.

> **quad-core?quad-core?**
>
> **Nope! It's better than that.**
>
> It has **10 physical cores** — but 2 of them are heavy-duty and 8 are optimized for efficiency. Intel calls this a **hybrid architecture**.

3.2 Running the program!

If we run the basic java scrip given by the Problem2 the result will be

```
1033 1024 1032 1037 1049  987  988 1020 1064  962
 972  996  960 1021 1053  974  992  973  991  947
 954  962  973 1000  988  963  974  980 1009  929

Matrix Sum = 1001895301

[thread_no]: 1 , [Time]:1570 ms

Process finished with exit code 0
```

in this case there is only one thread, that's why its so slow!

Now we do use our multi threaded class, which is fully explain in video , and running it, I put for example with 5 thread and we will see the result is immediately will be better

```
Matrix[1000][1000]
Sum = 1001895301
Thread 0: 374 ms
Thread 1: 300 ms
Thread 2: 292 ms
Thread 3: 249 ms
Thread 4: 336 ms
Total time: 374 ms

Process finished with exit code 0
```

Its almost 5 times better than that , and we only used 5 threads haha,

How time to run the GOAT , which is **compare_all.java** , it will run the program , with different thread numbers , and write us all ther total execution times , so its make out job easier to compare ;D

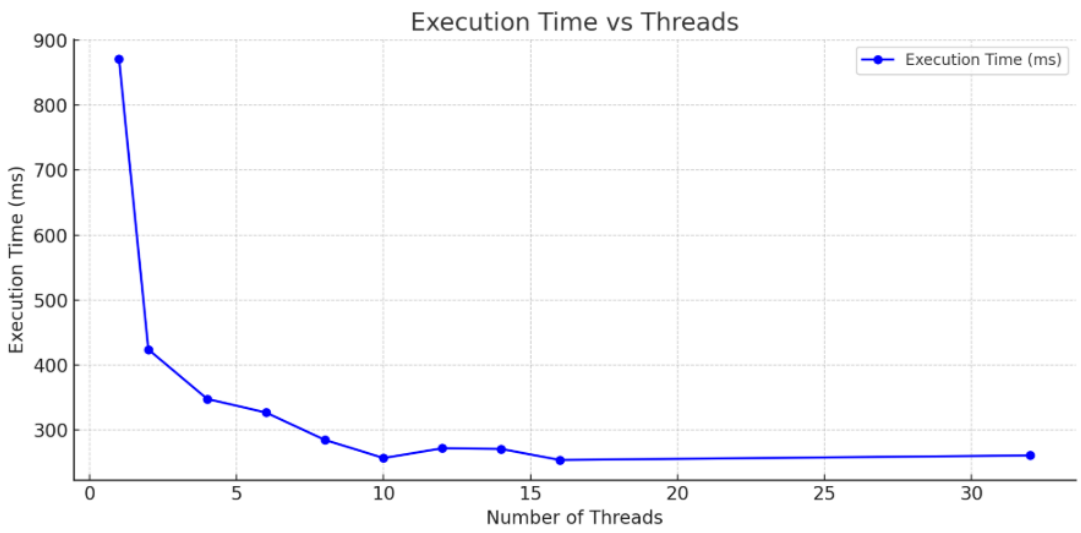
```
Threads:  1 => Time:  871 ms
Threads:  2 => Time:  424 ms
Threads:  4 => Time:  348 ms
Threads:  6 => Time:  327 ms
Threads:  8 => Time:  285 ms
Threads: 10 => Time:  257 ms
Threads: 12 => Time:  272 ms
Threads: 14 => Time:  271 ms
Threads: 16 => Time:  254 ms
Threads: 32 => Time:  261 ms

Process finished with exit code 0
```

And ladies and gentelmens are u see , the Runtime has decrease in every step , so lets put all these data in a table :D

Threads	1	2	4	6	8	10	12	14	16	32
Static Block (ms)	871	424	348	327	285	257	272	271	254	261

And now we use our python script , to generate us photo of graph with this numbers as array input :D

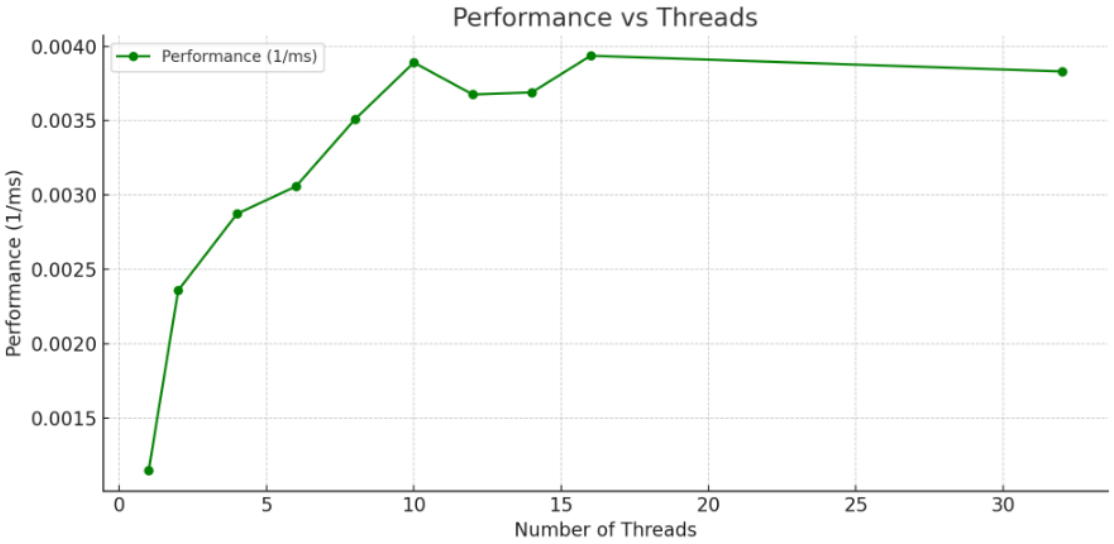


As its clearly visible with increasing the number of threads the runtime of program will decrease. In the beginning this changes are a lot and more eye catching , special from 1 to 4 threads , and then this trend slows down , and as its visible after the 10 threads the changes is not really that much . how ever it still become better.

Performance is (literally 1/exec time) depending on the execution time , the faster device works the more performance we will receive from that , first we need to calculate the performance of threads based on the exec times that we have in the last table .

Threads	1	2	4	6	8	10	12	14	16	32
Static Block Performance (1/ms)	0.001148	0.002358	0.002874	0.003058	0.003509	0.003891	0.003676	0.00369	0.003937	0.003831

Well I can say its not the beautifulest table I ever made , sorry about that Microsoft excel is killing me , how ever it has all the data it must have so that's fine. Now lets check the graph pf performance .



Now as we can observe the performance raised after increasing number of threats. In the begging this change was more , but after around 10 threads performance increasing slows and the changes are not that crazy anymore , how ever there are still some improvements. The experiment clearly shows that increasing the number of threads significantly reduces execution time in matrix multiplication using static block decomposition. The best performance is observed between 10 to 16 threads, aligning with the CPU's capabilities, while using more than 16 threads offers minimal improvement due to overhead and hardware limitations.