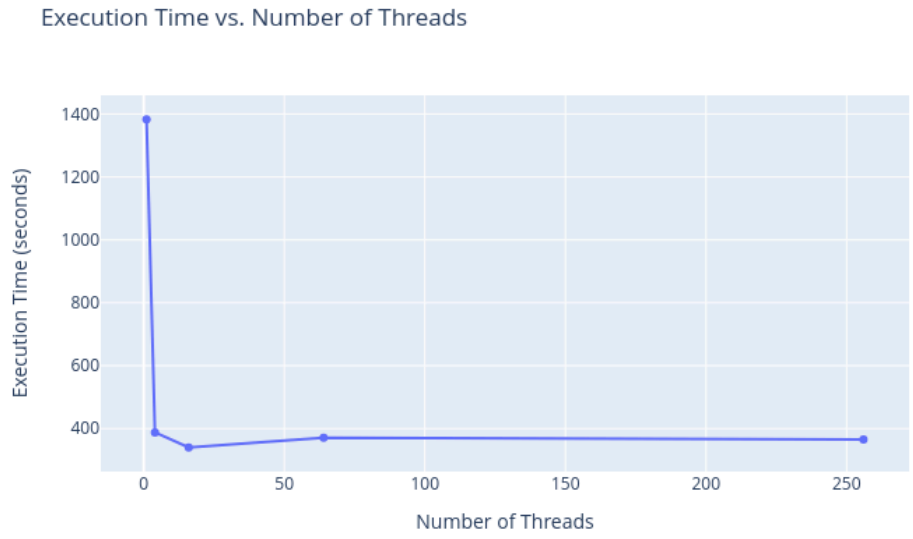
Name: Rohit Pujar Nagraj
USC ID: 4664 4939 18

# Programming Homework 1 - Report

## 1. Matrix Multiplication

### 1.1 Local Output Variables

| No. of Threads | Execution Time | Performance |
|---|---|---|
| 1 | 1,383,238,290,000 ns<br>(1,383.238290 sec) | 99,360,287 FLOPS<br>(99.360287 MFLOPS) |
| 4 | 388,641,673,000 ns<br>(388.641673 sec) | 353,639,260 FLOPS<br>(353.639260 MFLOPS) |
| 16 | 340,028,312,000 ns<br>(340.028312 sec) | 404,198,558 FLOPS<br>(404.198558 MFLOPS) |
| 64 | 371,307,126,000 ns<br>(371.307126 sec) | 370,148,978 FLOPS<br>(370.148978 MFLOPS) |
| 256 | 366,939,739,000 ns<br>(366.939739 sec) | 374,554,563 FLOPS<br>(374.554563 MFLOPS) |



Name: Rohit Pujar Nagraj
USC ID: 4664 4939 18

**Observations:** The execution time decreases from 1 thread to 4 and further to 16 threads, and then it increases slightly. The code ran on a 4-core CPU, and additional threads cannot be executed parallelly. Additionally, we see a slight increase in execution time at 64 threads. This could be attributed to the overhead of creating threads, while there is no speed benefit due to the limited CPU parallelization.

**Partitioning:** Each block of size n/p in the output matrix is handled by one thread, and the same data element in the output matrix is never updated by two different threads thus, we do not use mutex in this implementation.

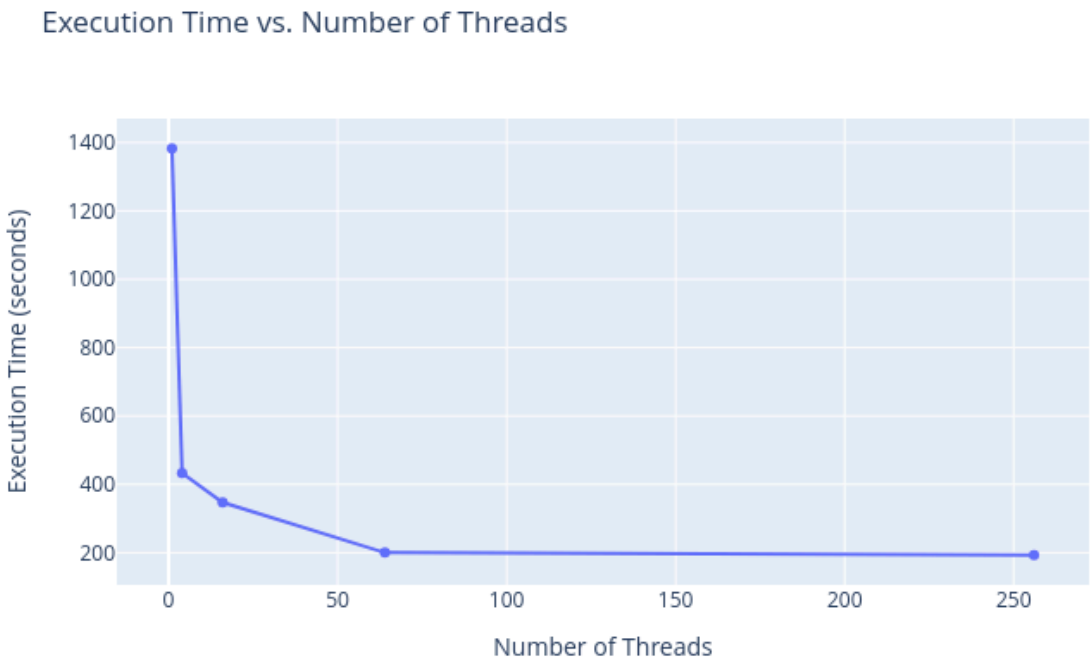
**Terminal Output:**

```
Lab/Programming HW 2 on   main [?] via   v14.2.1-gcc took 6m2s
❯ ./p1a 2
Number of FLOPs = 137438953472, Execution time = 388.641673 sec,
353.639260 MFLOPs per sec
C[100][100]=879616000.000000

Lab/Programming HW 2 on   main [?] via   v14.2.1-gcc took 6m28s
❯ ./p1a 4
Number of FLOPs = 137438953472, Execution time = 340.028312 sec,
404.198558 MFLOPs per sec
C[100][100]=879616000.000000

Lab/Programming HW 2 on   main [?] via   v14.2.1-gcc took 5m40s
❯ ./p1a 8
Number of FLOPs = 137438953472, Execution time = 371.307126 sec,
370.148978 MFLOPs per sec
C[100][100]=879616000.000000

Lab/Programming HW 2 on   main [?] via   v14.2.1-gcc took 6m11s
❯ ./p1a 16
Number of FLOPs = 137438953472, Execution time = 366.939739 sec,
374.554563 MFLOPs per sec
C[100][100]=879616000.000000
```

### 1.2 Shared Output Variables

Name: Rohit Pujar Nagraj
USC ID: 4664 4939 18

| No. of Threads | Execution Time | Performance |
|---|---|---|
| 1 | 1,383,238,290,000 ns<br>(1,383.238290 sec) | 99,360,287 FLOPS<br>(99.360287 MFLOPS) |
| 4 | 433,491,396,000 ns<br>(433.491396 sec) | 317,051,168 FLOPS<br>(317.051168 MFLOPS) |
| 16 | 347,791,980,000 ns<br>(347.791980 sec) | 395,175,741 FLOPS<br>(395.175741 MFLOPS) |
| 64 | 200,550,032,000 ns<br>(200.550032 sec) | 685,310,055 FLOPS<br>(685.310055 MFLOPS) |
| 256 | 193,415,494,000 ns<br>(193.415494 sec) | 710,589,159 FLOPS<br>(710.589159 MFLOPS) |



Name: Rohit Pujar Nagraj
USC ID: 4664 4939 18

**Terminal output:**

```
❯ ./p1b 2
Number of FLOPs = 137438953472, Execution time = 433.491396 sec,
317.051168 MFLOPs per sec
C[100][100]=879616000.000000

Lab/Programming HW 2 on   main [?] via   v14.2.1-gcc took 7m13s
❯ ./p1b 4
Number of FLOPs = 137438953472, Execution time = 347.791980 sec,
395.175741 MFLOPs per sec
C[100][100]=879616000.000000

Lab/Programming HW 2 on   main [?] via   v14.2.1-gcc took 5m48s
❯ ./p1b 8
Number of FLOPs = 137438953472, Execution time = 200.550032 sec,
685.310055 MFLOPs per sec
C[100][100]=879616000.000000

Lab/Programming HW 2 on   main [?] via   v14.2.1-gcc took 3m21s
❯ ./p1b 16
Number of FLOPs = 137438953472, Execution time = 193.415494 sec,
710.589159 MFLOPs per sec
C[100][100]=879616000.000000
```

**Observations:** We observe a decaying performance improvement with an increase in the number of threads. It stagnates after 64 threads because the CPU has a limited number of cores, and any more threads cannot run in parallel to speed up the execution.

**Partitioning:** In this algorithm, each thread (i,j) is responsible for one block A(i,j). So, all the values of C(i,:) and C(:, j) are updated by this thread (i,j). Thus, we need a mutex lock since multiple threads are simultaneously updating the values of C.

**Problem 1.1 vs. 1.2:** We observe that the best execution time of 1.2 is better than that of 1.1. This is because of the overhead of creating threads every time in 1.1. However, for a lower number of threads, 1.1 performs better as it does not lock a large chunk of the output matrix, unlike in 1.2.

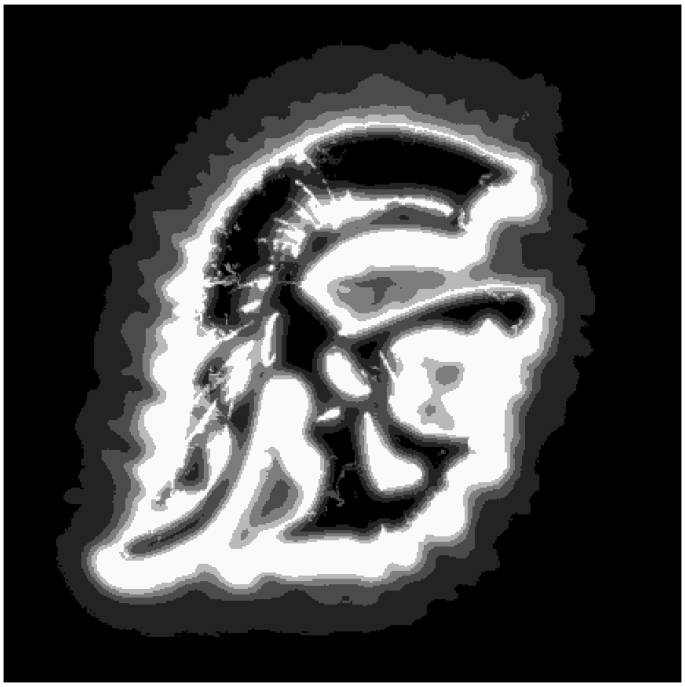
Name: Rohit Pujar Nagraj
USC ID: 4664 4939 18

# 2. Parallel K-Means

## 2.1 Iteratively creating and joining threads

| No. of Threads | Execution Time |
|---|---|
| 1 | 2,395,388,000 ns<br>(2.395388 sec) |
| 2 | 802,675,000 ns<br>(0.802675 sec) |
| 4 | 1,244,380,000 ns<br>(1.244380 sec) |
| 8 | 1,073,809,000 ns<br>(1.073809 sec) |

**Output Image:**



Name: Rohit Pujar Nagraj
USC ID: 4664 4939 18

**Terminal Output:**

```
Lab/Programming HW 2 on   main [?] via   v14.2.1-gcc
❯ ./p2a 1
Execution time = 0.746170 sec

Lab/Programming HW 2 on   main [?] via   v14.2.1-gcc
❯ ./p2a 2
Execution time = 0.391537 sec

Lab/Programming HW 2 on   main [?] via   v14.2.1-gcc
❯ ./p2a 4
Execution time = 0.669615 sec

Lab/Programming HW 2 on   main [?] via   v14.2.1-gcc
❯ ./p2a 8
Execution time = 0.496173 sec
```

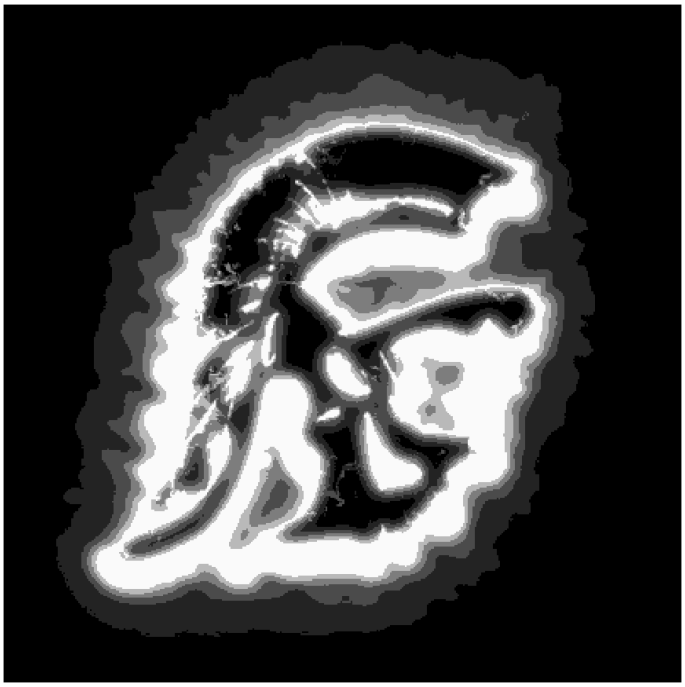
**Observations:** The execution time decreases as we create more threads. However, the overhead of creating threads quickly outweighs the performance gains.

### 2.2 One-time Thread Creation

| No. of Threads | Execution Time |
|---|---|
| 1 | 756,933,000 ns<br>(0.756933 sec) |
| 2 | 823,925,000 ns<br>(0.823925 sec) |
| 4 | 225,577,000 ns<br>(0.225577 sec) |
| 8 | 553,521,000 ns<br>( 0.553521 sec) |

Name: Rohit Pujar Nagraj
USC ID: 4664 4939 18

**Output Image:**



**Terminal Output**

```
Lab/Programming HW 2 on   main [?] via   v14.2.1-gcc
❯ ./p2b 1
Execution time = 0.756933 sec

Lab/Programming HW 2 on   main [?] via   v14.2.1-gcc
❯ ./p2b 2
Execution time = 0.823925 sec

Lab/Programming HW 2 on   main [?] via   v14.2.1-gcc
❯ ./p2b 4
Execution time = 0.225577 sec

Lab/Programming HW 2 on   main [?] via   v14.2.1-gcc
❯ ./p2b 8
Execution time = 0.553521 sec
```

**Observations:** We observe that the time decreases and then starts increasing. I'm running this on a 4-core machine, and any more than four threads will not yield better performance but will add overhead.

Name: Rohit Pujar Nagraj
USC ID: 4664 4939 18

**Problem 2.1 vs. 2.2:** The runtime of 2.2 is consistently lower than 2.1. This can be attributed to the lack of thread creation and synchronization overhead in 2.2. Surprisingly, both 2.1 and 2.2 are slower than naive K-means from PHW1 (0.27 sec) (except 2.2 with four threads), showing the overhead of thread creation and synchronization.