Homework 2

Name: Anand Ravi

System Properties:

Processor : Intel Core i3-6100U
Number of Cores: 4
Operating System: Ubuntu 16.04

Problem 1. Parallel Matrix Multiplication

How I implemented the code:

In this implementation of parallel multiplication, I am passing matrix A and matrix B to all my threads created. Each thread has n/p columns of output matrix which every thread needs to compute using naive multiplication approach. I have created a thread structure which knows in which range it is supposed to compute the output for using start and end index.

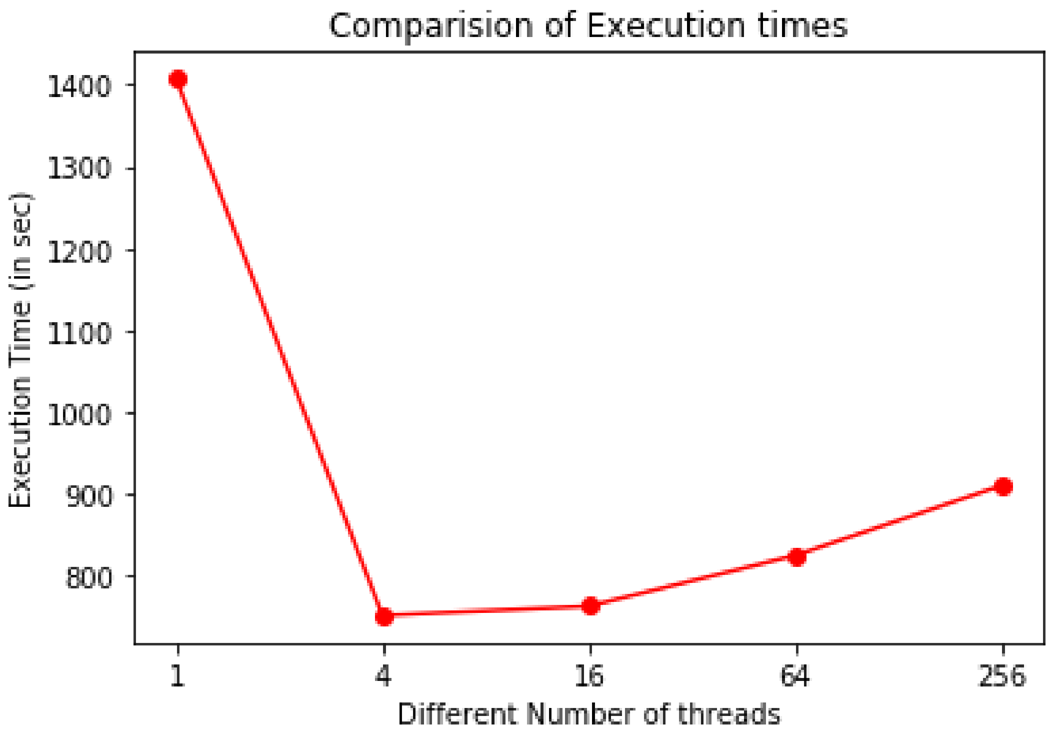
```
Thus in each thread
For i in 0 - num_of_threads - 1
Start_index = (i * n /num_of_threads)
End_index = (i + 1) * n/num_of_threads

And inside each thread
for (int i = 0; i  < n; i++ )
for( int j = start; j < end; j++)
        C[i][j] = C[i][j] + (A[i][k] * B[k][j])
```

Number of Threads	Execution Time
1	1407
4	752.188985
16	763.620165
64	825.512159
256	910.963923

Intuition:

The number of cores in my laptop is 4. Thus it makes sense that the least number of time taken for this parallel naive multiplication is 4. When the number of threads are way more that number of cores then the execution time goes on increasing. The reason is when we go on increasing the number of threads, physically it can only work on 4 threads at a time. More thread means there is more overhead due to context switching and thus the overall execution time decreases.



Problem 2. Parallel K Means

In parallel k means implementation the image is broken down and each thread is incharge of a vertical region in the image. In that region, the thread performs k means clustering and keeps on storing the values of the pixels belonging in a cluster. When all the threads are done with the work, we simply update the value of the cluster mean by considering the sum of pixels for each cluster in all the threads and dividing it my total number.

The execution time where as follows:

Number of Threads	Execution Time in secs
1	3.13
4	1.6488
8	1.64011

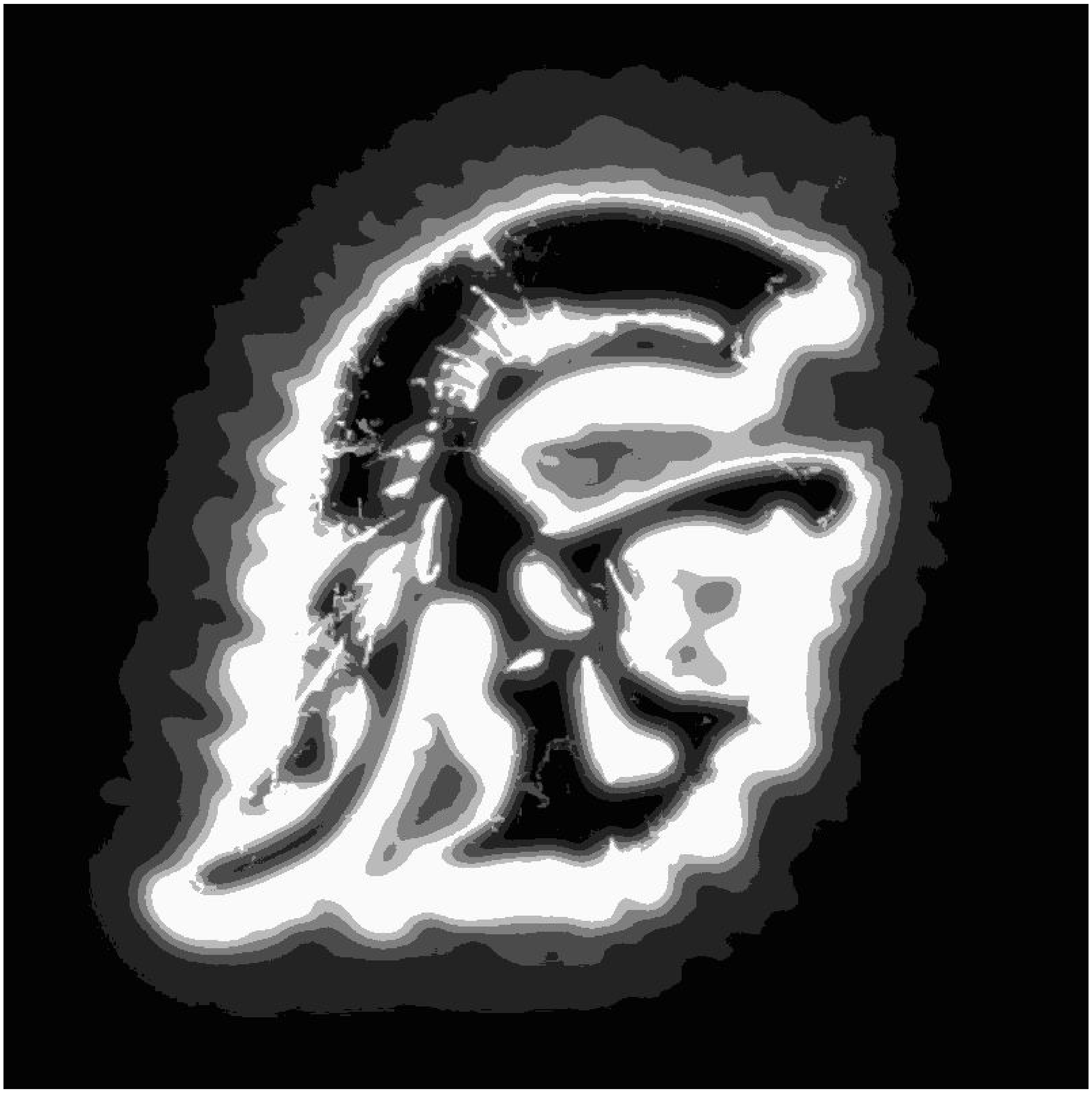
We can see that by distributing the work, we are able to bring down the execution time.

The means are:

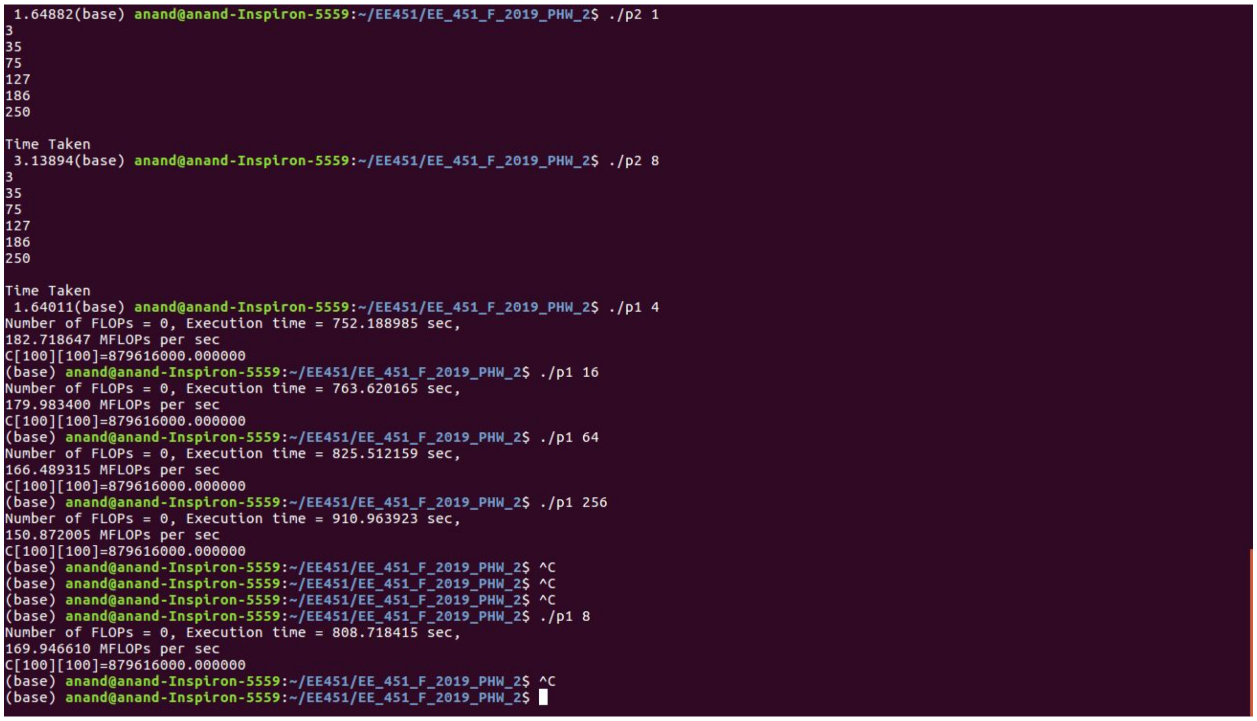
3
35
75
127
186
250

Output:

Input:



Appendix:



```
 1.64882(base) anand@anand-Inspiron-5559:~/EE451/EE_451_F_2019_PHW_2$ ./p2 1
3
35
75
127
186
250

Time Taken
 3.13894(base) anand@anand-Inspiron-5559:~/EE451/EE_451_F_2019_PHW_2$ ./p2 8
3
35
75
127
186
250

Time Taken
 1.64011(base) anand@anand-Inspiron-5559:~/EE451/EE_451_F_2019_PHW_2$ ./p1 4
Number of FLOPs = 0, Execution time = 752.188985 sec,
182.718647 MFLOPs per sec
C[100][100]=879616000.000000
(base) anand@anand-Inspiron-5559:~/EE451/EE_451_F_2019_PHW_2$ ./p1 16
Number of FLOPs = 0, Execution time = 763.620165 sec,
179.983400 MFLOPs per sec
C[100][100]=879616000.000000
(base) anand@anand-Inspiron-5559:~/EE451/EE_451_F_2019_PHW_2$ ./p1 64
Number of FLOPs = 0, Execution time = 825.512159 sec,
166.489315 MFLOPs per sec
C[100][100]=879616000.000000
(base) anand@anand-Inspiron-5559:~/EE451/EE_451_F_2019_PHW_2$ ./p1 256
Number of FLOPs = 0, Execution time = 910.963923 sec,
150.872005 MFLOPs per sec
C[100][100]=879616000.000000
(base) anand@anand-Inspiron-5559:~/EE451/EE_451_F_2019_PHW_2$ ^C
(base) anand@anand-Inspiron-5559:~/EE451/EE_451_F_2019_PHW_2$ ^C
(base) anand@anand-Inspiron-5559:~/EE451/EE_451_F_2019_PHW_2$ ^C
(base) anand@anand-Inspiron-5559:~/EE451/EE_451_F_2019_PHW_2$ ./p1 8
Number of FLOPs = 0, Execution time = 808.718415 sec,
169.946610 MFLOPs per sec
C[100][100]=879616000.000000
(base) anand@anand-Inspiron-5559:~/EE451/EE_451_F_2019_PHW_2$ ^C
(base) anand@anand-Inspiron-5559:~/EE451/EE_451_F_2019_PHW_2$ 
```