EE451 Programming HW #2

Siddhant S. Nadkarni

2596-6618-89

System Information:

OS: macOS Mojave (v 10.14)
Processor: 1.6 GHz Intel Core i5
Memory: 8 GB 1600 MHz DDR3
Graphics: Intel HD Graphics 6000 1536 MB

Q 1. Parallel Matrix Multiplication

Num threads = 1
Execution time (s) = 2061.79

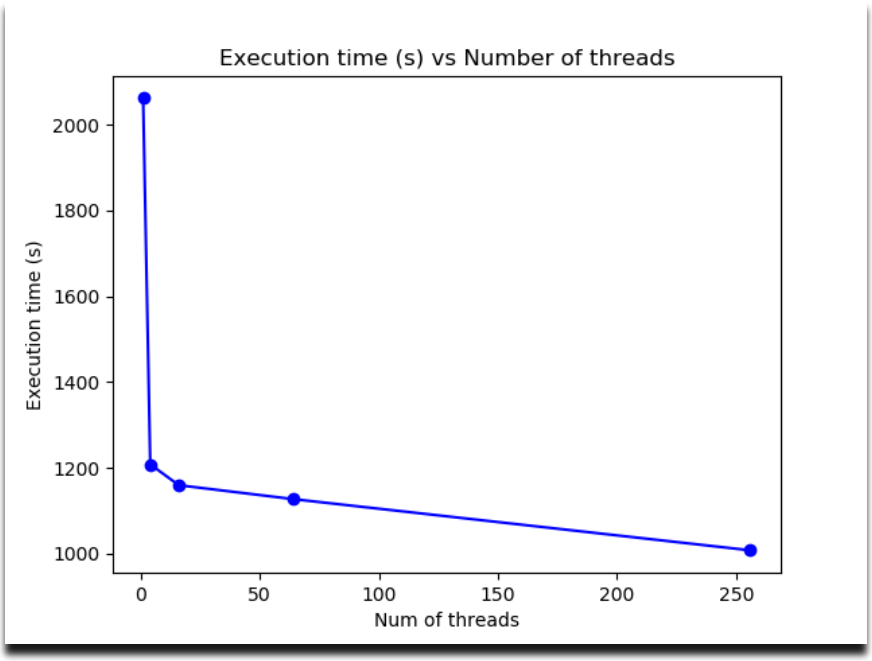
Num threads = 4
Execution time (s) = 1208.32

Num threads = 16
Execution time (s) = 1159.71

Num threads = 64
Execution time (s) = 1127.31

Num threads = 256
Execution time (s) = 1007.93

Plot:



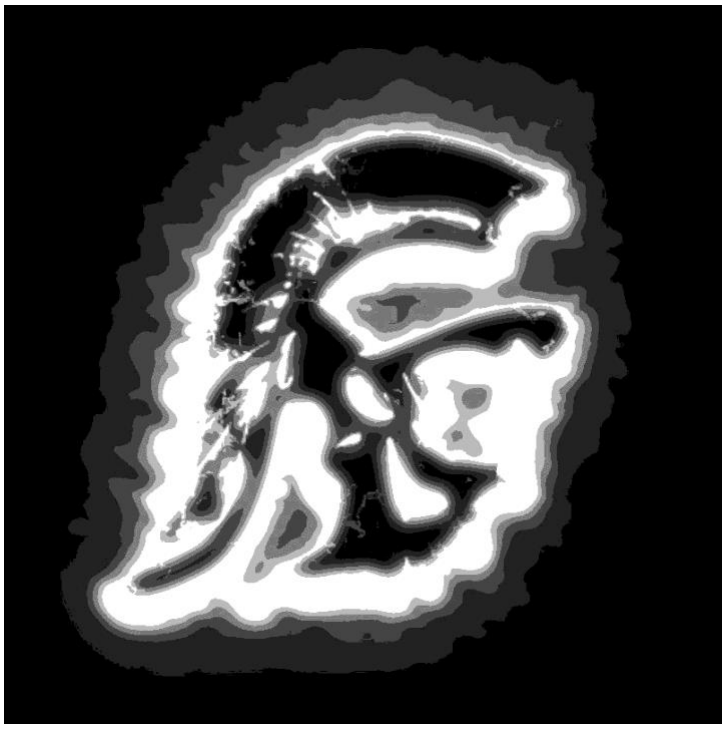
Computation of output matrix working:

Depending on the number of threads, I divide the output matrix into equal regions along the columns between them. For e.g. if num threads = 4, then output matrix will be divided into 4 sections along the columns, and thread 1 will compute between 0 to n/4 - 1, thread 2 will compute between n/4 to 2n/4 – 1, thread 3 will computer between 2n/4 to 3n/4 – 1 , thread 4 will compute between 3n/4 to n - 1. Each thread has an argument which consists of start and end limits of the regions of the output matrix it is responsible for. Each thread contains pointers to each matrix i.e. A, B, C. Finally, the matMultiply function in my program performs naive matrix multiplication for each thread i.e. for the output matrix section each thread is responsible for. All these threads operate asynchronously and are finally joined by using a barrier.

Q 2. Parallel K-Means

Input Image:

Output Image:



For threads = 8:

Execution time (s) = 3.85

For threads = 4:

Execution time (s) = 4.16

Observation:

In Pthreads parallel programming model, we use multiple threads to perform k-means clustering. Each thread divides the task of clustering elements between them, in comparison to serial version in which only one thread operates on the entire image. All threads in parallel programming model operate asynchronously and they are finally synchronized using a barrier. There is no requirement of sharing variables and thus no requirement of locking variables as each thread operates on independent summations for calculating means of clusters. The execution time was improved by 90.47 % compared to serial k-means due to multithreading. Also, as we increase number of threads, the task of clustering gets divided between them, thus improving time.