Fall 2018 Semester

CS 6068 Parallel Computing

Project Report on Parallel K-means Clustering

Nithyasri Babu

(M12506236)

Introduction

Data Mining is a huge field of study in current times. The applications are in every industry and so very popular. The word 'Mining' generally means the process extracting something of value from a given source. Data sources in the current are growing exponentially every day. Some companies collect and explore the data without the knowledge of what the results will be. This type of mining is known as Unsupervised Data Mining where the data is used to get inferences without any previous results that it can be compared to check for correctness.

This is a constantly growing field and has an influx of algorithms being invented for faster and more reliable results. But the K-Means Clustering algorithm has been used as it is and as a part of many of the new algorithms. K-Means a popularly used clustering algorithm known for its simplicity and correctness. The main part of K-Means is to calculate distances of each point to a selected number of centroids.

Though it does not have any complex calculations, with the increase in the size of data, the amount of computation increases decreasing the efficiency of the algorithm. But at the same time since the distance calculations of each point are independent of each other, they could be computed concurrently.

The time complexity of the K-Means is O(nkt), where n is the number of data points, k is the number of clusters, t is the number of iterations it takes to complete the cluster assignments. The number of iterations could be limited to a maximum value or when there are no more changes to cluster assignments in given iterations.

Space complexity is limited to the amount of space required to store the data points, centroids, and the cluster assignments. Given that there are m data points with n features, and k clusters need to be formed, the space required for the data points is O(m*n) and the centroids O(k*n). There is only 1 cluster assignment for each data point in K-Means for which the space requirement is O(n). Because of these are fixed values the Space complexity of the K-Means algorithm would be O(((m+k) * n) + n).

The goal of this project is to achieve execution time improvements of K-Means algorithm for by parallelizing the distance computations and have 100% CPU Utilization. I have implemented a

naïve K-Means algorithm where the distance between each data point and each centroid is computed sequentially. Then, using the multiprocessing module in python, I have divided the tasks between all the processors available in the system to parallelize the distance computation and utilizing the full computing power of the CPU.

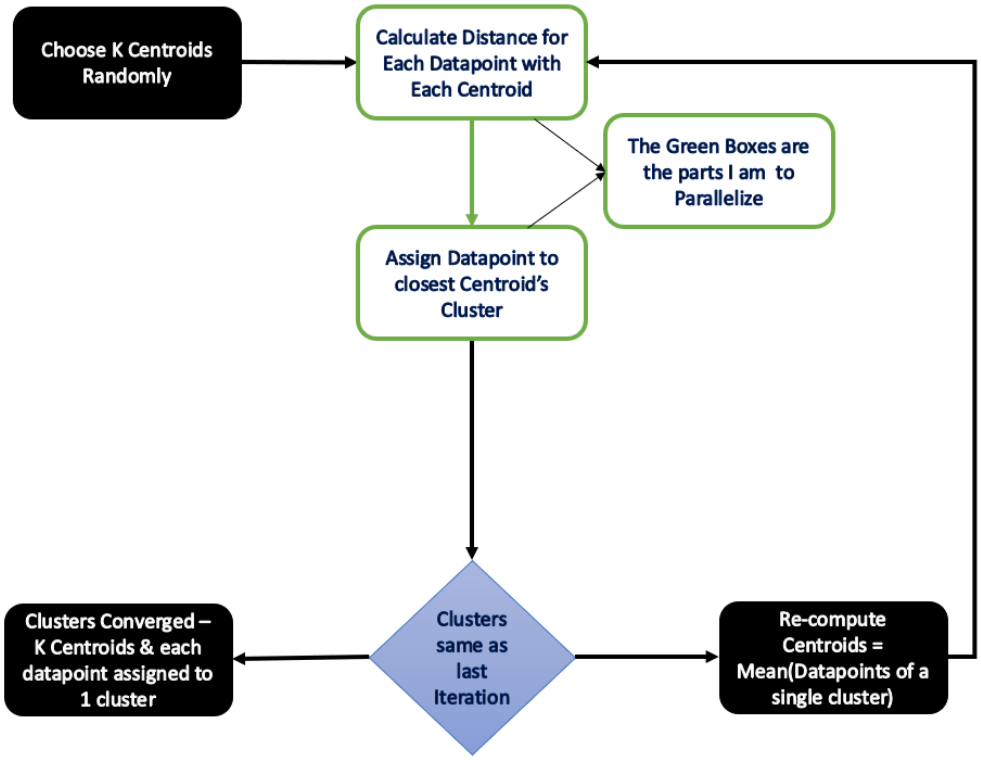
Execution time improvements are not achieved until the computations are significantly high. We can see that the improvements for a larger dataset are clearly visible in the graphs that are provided in the below segments.

Sequential Implementation Design

Steps of K-Means Clustering Algorithm:

1. Pick K centroids randomly
2. Compute Distance between each data point with k centroids
3. Assign Cluster based on the minimum distance from the centroid
4. If Clusters are not the same as previous iteration:
    a. Recompute Centroids with new cluster Assignments
    b. Repeat from Step 2
5. If no, End Process

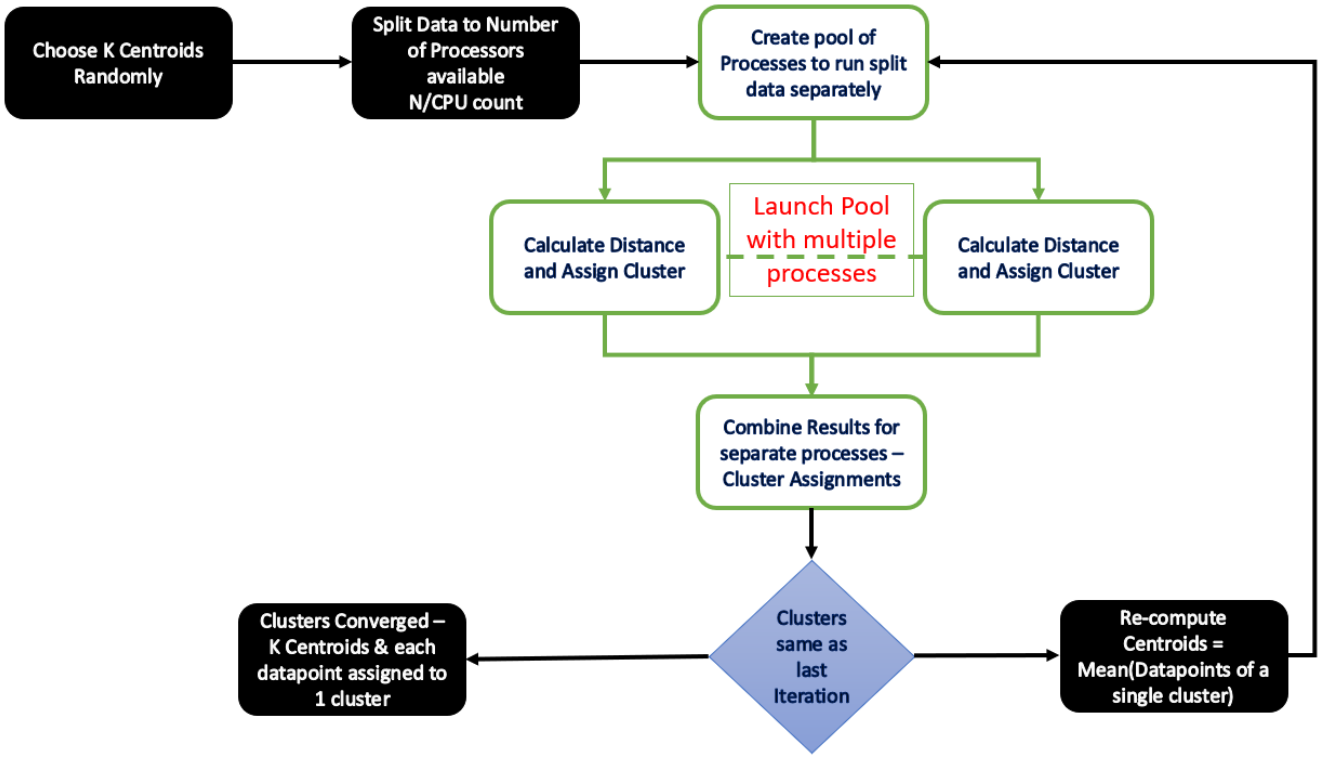
High-Level Design Flow Chart of the Sequential Implementation:



Parallel Implementation Design & Optimization

Data Parallelism is used such that the N data points are divided into multiple partitions for the number of processors available in the system. Distance calculations run concurrently on different processors. The cluster assignments from each of the partitions are later combined are to check if the next iteration is required. If it is required, then it is centroids are recomputed and the same steps are repeated.

High-Level Design Flow Chart for Parallel Implementation:



Extra steps that are taken to Parallelize:

1. Partitioning the Data points to CPU counts
2. Launching the Pool of processes for distance computations
3. Combining the results from each partition

The Parallelized implementation should ideally be faster than the sequential implementation. I have chosen to implement this in Python. Python being an Interpreted Language is not usually used for Parallel Computing. Using the Multiprocessing module, I have implemented a version of K-Means that will use 100% of the CPU.

Since the Processors are sharing the data and computing the distances the time taken should be divided by the number of the processors involved in the computation. The parallel implementation is expected to perform well for datasets that are huge with many attributes. Speedups are significantly high when the computation required is also high.

Performance Analysis:

System Configuration used for testing:

- Intel® Xeon® W-2123 CPU @ 3.60GHz Processor
- 16.0 RAM
- 64-Bit Windows Operating System

Code Results:

The program was tested on 2 Datasets with varying shape:

- Wholesale Customer Data: 440 Data points × 8 Attributes
- Wine Quality Data: 4898 Data points x 12 Attribute
- **The number of attributes was reduced to 8 in the wine quality to highlight on improvement in performance based on the computation quantity.**
- Source: UCI Machine Learning

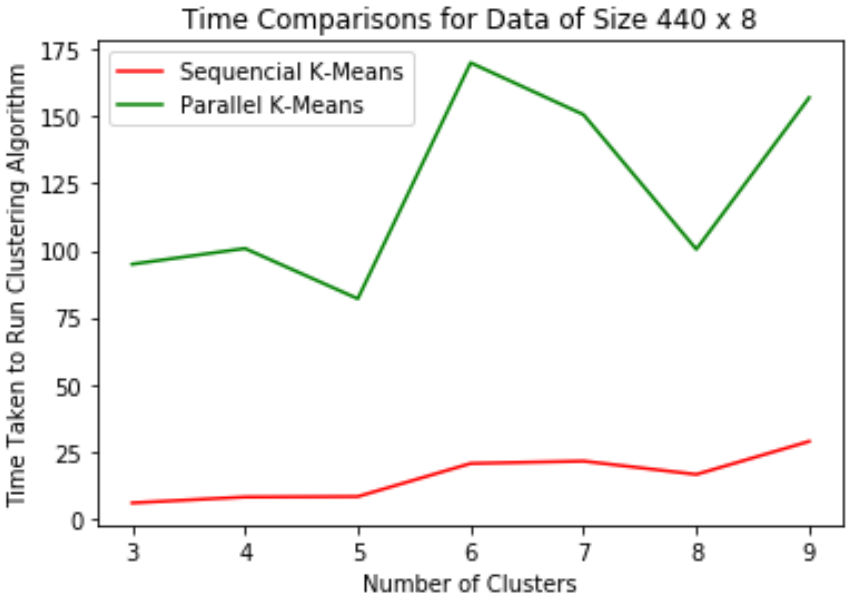
Both the sequential and parallel implementations were run for the datasets mentioned above.

The Total Time taken and the Mean Time Taken for a single iteration is recorded once it's been executed for different number of clusters.

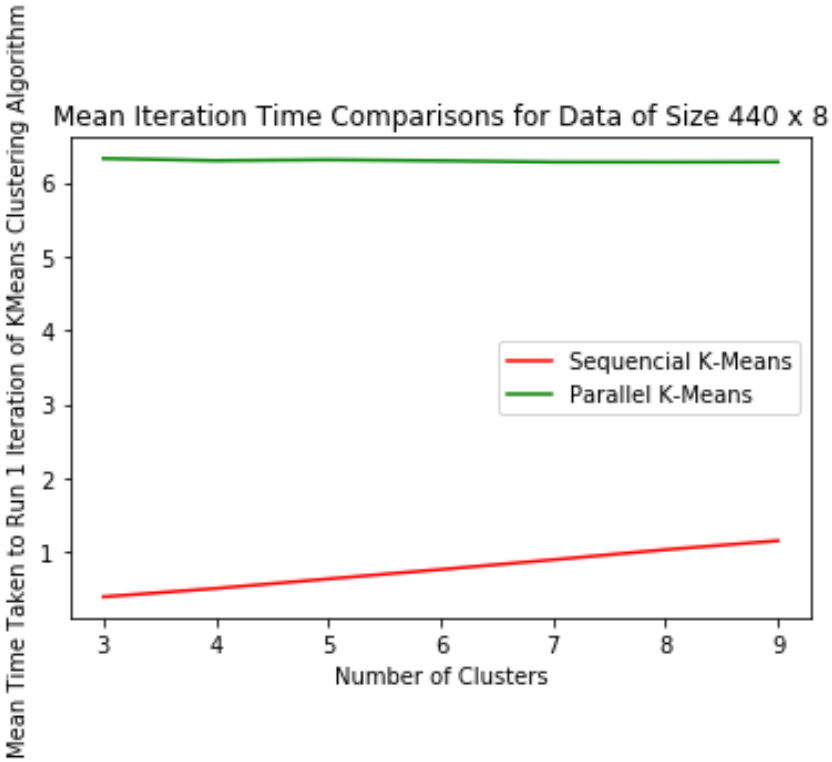
The correctness of both the implementations is checked by comparing the final cluster assignments for the different number of clusters its executed for. The difference is calculated after the execution of both the implementations and if it is not the same, the difference is shown.

Results for Whole Sale Customer Data

Plot for Total Execution Time:

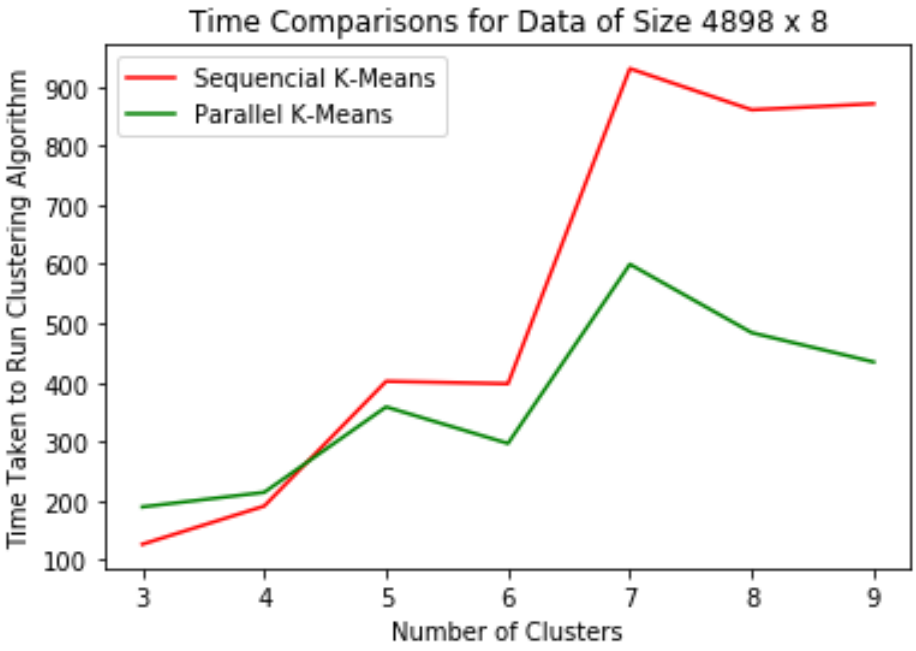


Plot for Mean Iteration Time:

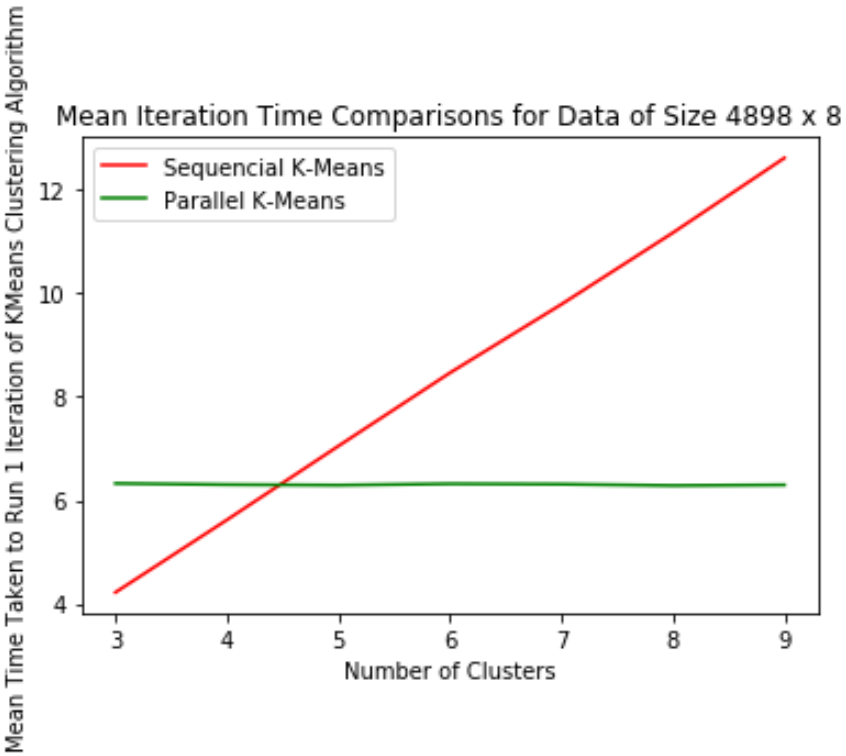


Results for Wine Quality Data

Plot for Total Execution Time:

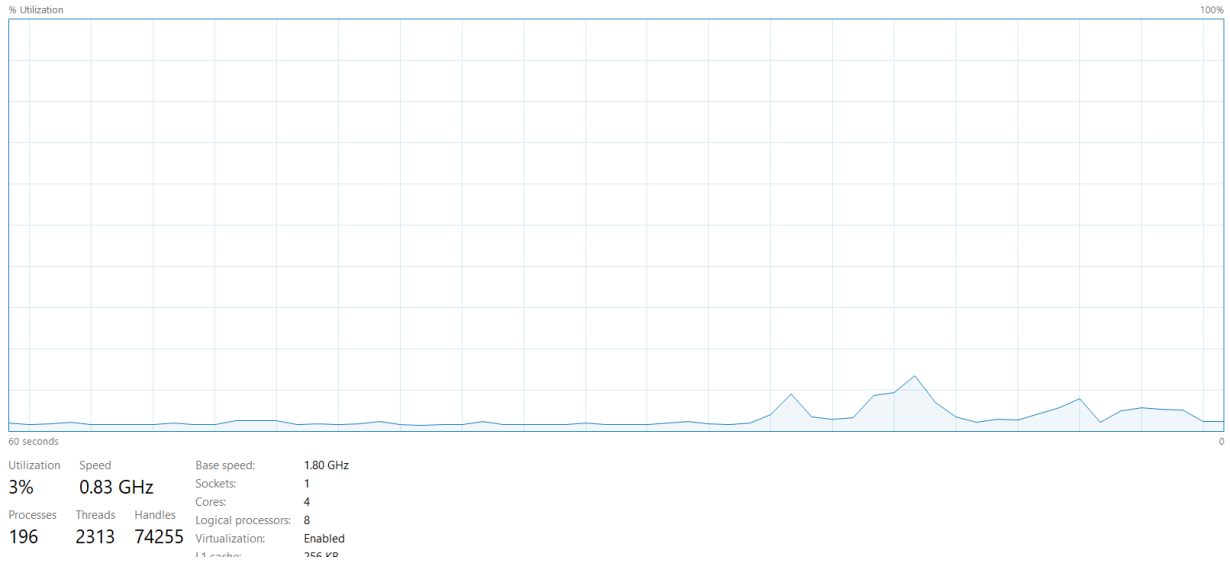


Plot for Mean Iteration Time:

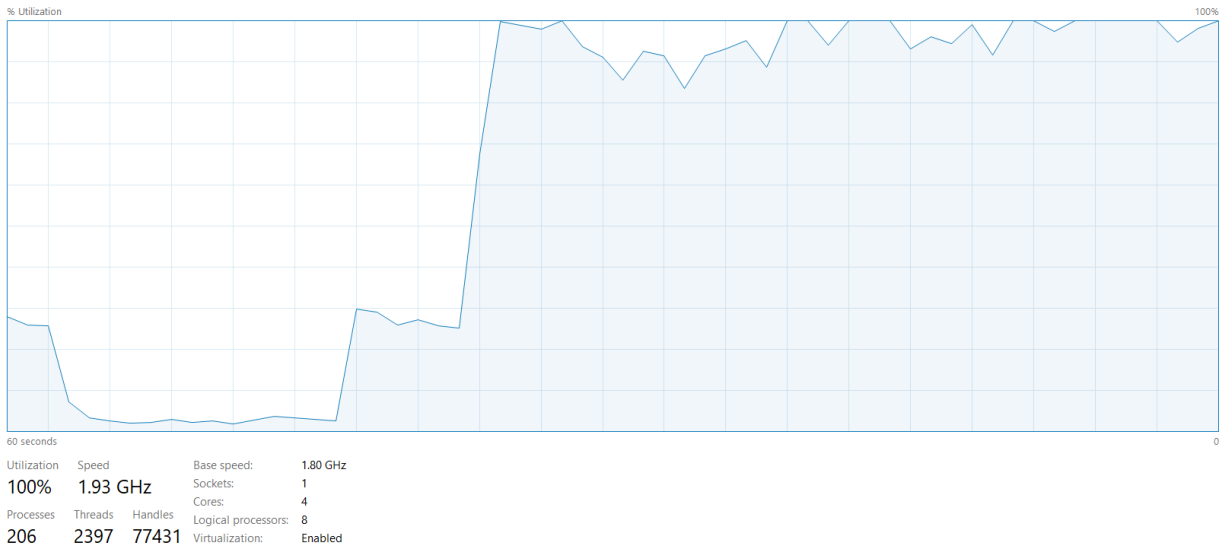


CPU Utilization Before, During and After running of the program

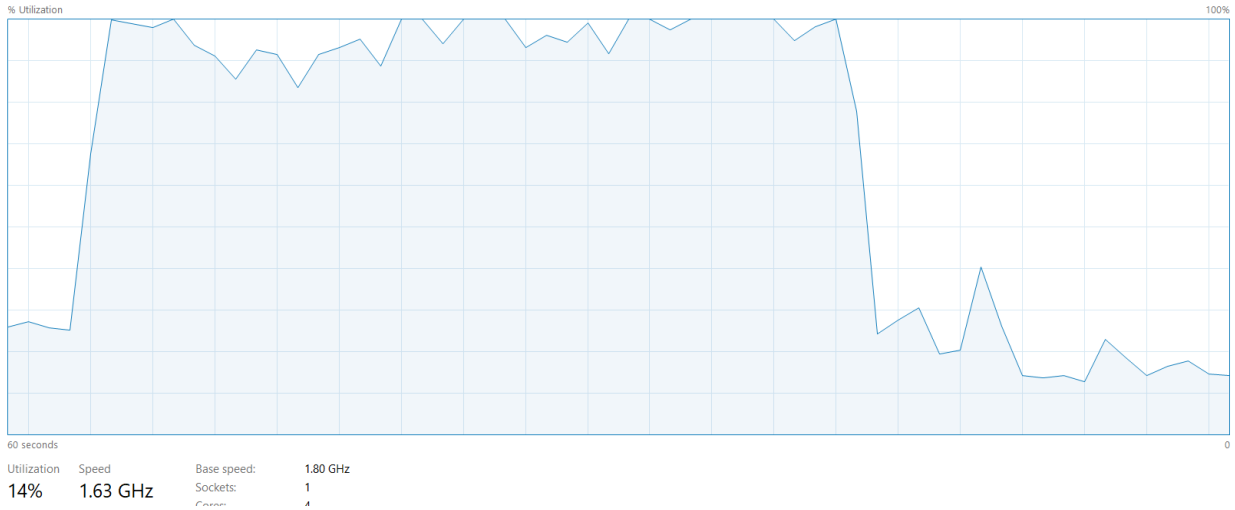
Before:



During:



After:



Observations on Performance Metrics:

Here we can see that the sequential program runs faster than the parallel program. The distance calculations are done for 440 instances. This N is relatively very low and does not really require parallelism for speedups.

One thing we can notice is that the mean time taken to run a single iteration is almost constant and gradually decreasing for the parallel implementation. At the same time for sequential implementation, as the number of clusters increase the mean time increases. This suggests that increase in any of the 3 parameters in the O(nkt) would increase the execution time for the sequential K-Means algorithm.

```
In [10]: diff_mp_km
Out[10]: array([0, 0, 0, 0, 0, 0, 0])
```

Here it shows that there is no difference in results of the sequential and parallel implementations.

Future Improvements:

- Implement PyCUDA solution on a CUDA capable system to utilize GPU Computing power
- Implement Parallel computation to Centroid Re-Computation

Source Code:

The code is included in the below git public repository.

https://github.uc.edu/babuni/CS6068-PKmeans

Please note that the Presentation & this report is included in the repository for your reference.

Bibliography:

1. Dataset Source: UCI Machine Learning:
    a. https://archive.ics.uci.edu/ml/datasets/Wholesale%20customers
    b. http://archive.ics.uci.edu/ml/datasets/Wine+Quality?iframe=true&width=95%&height=95%
2. Chunfei Zhang, Zhiyi Fang, An Improved K-means Clustering Algorithm, 2013, Journal of Information & Computational Science, 10, 193–199
3. Kittisak Kerdprasop, Nittaya Kerdprasop, A lightweight method to parallel k-means clustering, 2010, International Journal of Mathematics and Computers in Simulation, Issue 4, Vol 4, 144-153