Project Report

Topic: Parallel-K-means-Clustering

Project Code: https://github.iu.edu/avipawar/hpc_project

- **The problem domain and motivation**

  K-means is a classification problem. Clustering by K-means is an unsupervised learning technique that is used when unlabeled data is present. Hence it is from the Machine learning domain.

  The primary motivation for this project is to write a parallelized algorithm for K-means which I use mostly in Machine learning problems. As this task is very resource intensive it always takes 10 to 20 minutes on large datasets to produce results. Hence I want to parallelize it and see how much speedup can be achieved.

- **A detailed problem description**

  https://en.wikipedia.org/wiki/K-means_clustering

  K-means is a classification problem in which we try to partition the dataset into different K subsets or clusters. A data point only belongs to only one set. The data points are classified such that the euclidean distance (sum of squared distance between data points is lower).

  There are a lot of iterations that have to be done to compute the centroid of data points and classify the data points. We iterate till there is no variation in clusters.

- **Your input/output details**

  The input and output are very simple for this algorithm.

  The Data file mentioned here contains cartesian points.

  Input: 1. Data file 2. The number of clusters to classify data into 3. No of worker threads

  Output: 1. Centroid output file 2. Cluster output file 3. Compute time

- **parallelization strategy/ Algorithm**

  Rather than sketching, I will explain the algorithm.

  1. Assign K=no of clusters to be classified as
  2. Initialize random K clusters
  3. Split Data by **No of data points/No of threads**

Parallel Loop // Iterate till we don't gate the same class as of the last iteration

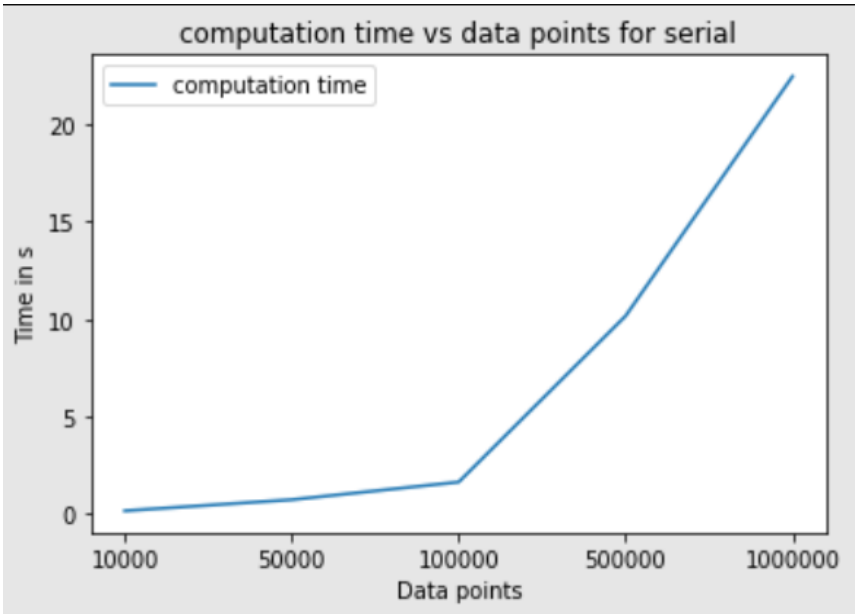
1. Find the Distance of each data point from all centroids
2. Find which centroid is nearest to the datapoint Assign that centroid as the class of that data point.
3. Recalculate the centroid of the new class by taking the mean of data points in the cluster.

Hence for the parallelization we already Split Data into **No of data points/No of threads/Nodes.** Also, we need to keep track of the coordinates of the centroids of the clusters And need to share the centroid data between the treads/nodes.

Stopping Condition: We are calculating the delta between centroids of each iteration at every loop. I have taken the threshold as 0.00001

- **Results**

As expected the computation time decreases as we use threads in our parallel program. The overall computation time for most of the dataset is near zero.



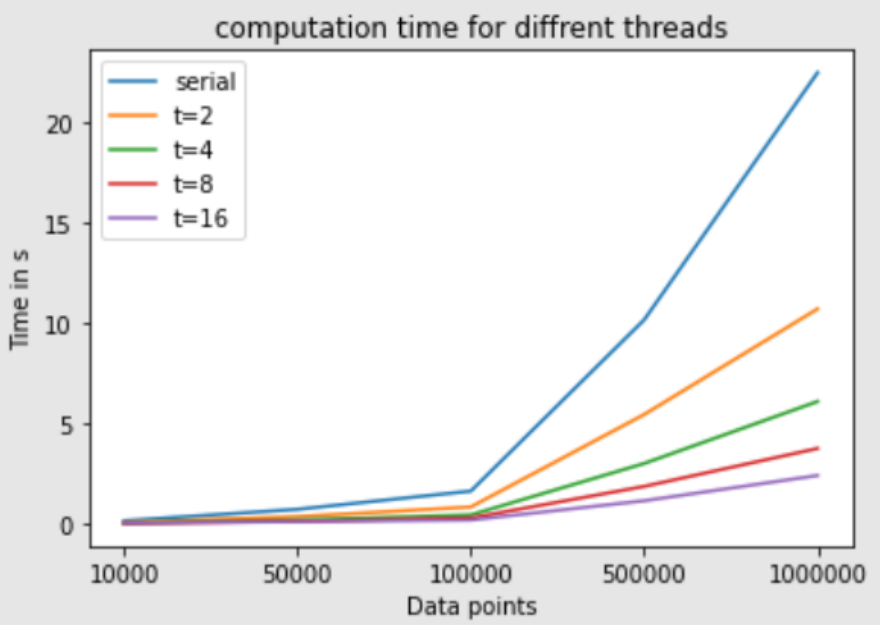
When we compare parallel implementation between the different number of threads then we get to know that the implementation does scale well.

Both the strong and weak scaling of the implementation is significant.

Near linear speedup can be seen if you compare the serial implementation and parallel implementation with two threads across the size of the dataset.

	10000	50000	100000	500000	1000000
Normal	0.164673	0.732804	1.638986	10.166147	22.471914
T=2	0.083916	0.367379	0.849792	5.453538	10.710458
T=4	0.043264	0.204285	0.452777	3.01576	6.100979
T=8	0.03313	0.136421	0.299451	1.872624	3.763004
T=16	0.042691	0.120351	0.199977	1.158823	2.416237

But the speedup does not happen linearly if you consider any other threads.
I think the problem size is too low for the computation hence the speedup decreases as the number of threads increases. This is because the cost of overheads will be far more than the cost of computations.
I think for the larger datasets than 100000 and with a larger thread pool we can achieve much more speedup as problem size will increase hence the threads will have problems for computation and the overheads will be negligible hence best speedup can be achieved.



- **Limitations**
  1. Currently we are taking 3 dimensional data and multidimensional data can be processed.
  2. I have tested with a constant cluster size of 5. But varying cluster size can also affect our computation times.
  3. I'm taking the threshold as 0.00001 which can be more fine tuned.

- **Future Scope**

  As the current implementation is on 3 Dimensional data I would like to work on multidimensional data. Also there is no dimensional reduction algorithm implemented now which can also be implemented.

  To make the processing faster I would also like to implement this using CUDA or any other graphics accelerated computing.

- **Refances**

1. https://towardsdatascience.com/k-means-clustering-algorithm-applications-evaluation-methods-and-drawbacks-aa03e644b48a
2. https://www.analyticsvidhya.com/blog/2021/11/understanding-k-means-clustering-in-machine-learningwith-examples/
3. https://en.wikipedia.org/wiki/K-means_clustering