CS 6068 - Parallel Computing Final Project Proposal

Project Title: Data Parallelism in K-Means Clustering

Group Member: Nithyasri Babu

#M12506236

Part 1 - Background and Motivation:

K-Means Clustering is one of the popularly used unsupervised learning algorithms because of its simplicity and versatility. It's called simple as it has an iterative set of steps that is done until the results converge at an acceptable point.

I am a machine learning enthusiast & have used this algorithm in a few of my projects before. I always thought that K-Means is a simple but versatile clustering algorithm with a decent time complexity. Given a dataset of n data points, the time complexity is O(nkt) where n is the number of data points, k is number of clusters formed and t is the number of iterations it takes for convergence.

The problem is that it works slowly for a data set which is huge and/or has a relatively high number of clusters. To solve this issue, using data parallelism to calculate distances may result in better runtime.

Part 2 - Application-level Objectives:

The sequential K-Means algorithms runs iterations of the same set of steps until it reaches and acceptable convergence. Initially k cluster centroids are chosen from the datapoints randomly/ using some criteria. Enhancing choosing methods for the centroids is not a part of this project.

K-Means Algorithm Steps:

1) Find the distance of each data point to all centroids.
2) Assign each data point to cluster with closest centroid.
3) Update Centroid Values

Repeat until there is no difference between the consecutive iteration results.

Out of the above-mentioned steps, maximum time taken is to calculate the distance from each centroid. The same operation is done on each datapoint though at each iteration there is a variation in the centroids used.

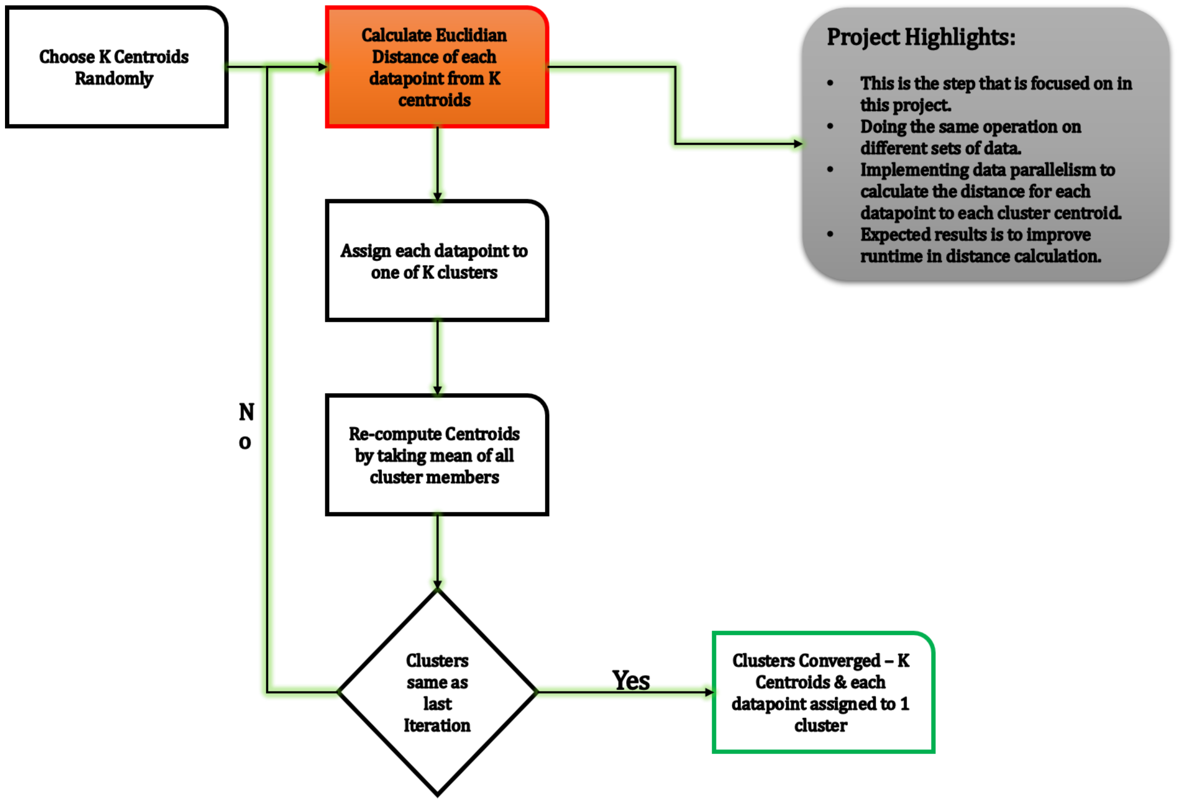
The main goal for this project is to improve run time for large datasets with more than 3 attributes by parallelizing the distance calculation for each datapoint.

Other features could be to include parallelism in calculation of mean to find new centroids.

Part 3 - Design Overview:

Block Diagram for K-Means Algorithm:

It's a repeated 3 step process and this project focuses on improving the time taken for distance calculation.


Choose K Centroids Randomly
Calculate Euclidian Distance of each datapoint from K centroids
Assign each datapoint to one of K clusters
Re-compute Centroids by taking mean of all cluster members
Clusters same as last Iteration
No
Yes
Clusters Converged – K Centroids & each datapoint assigned to 1 cluster
Project Highlights:
This is the step that is focused on in this project.
Doing the same operation on different sets of data.
Implementing data parallelism to calculate the distance for each datapoint to each cluster centroid.
Expected results is to improve runtime in distance calculation.

The code will be written in Python and numbapro python library is used to implement CUDA for parallelizing the K-Means algorithm.

Host:

- Choosing initial centroids
- Recomputing new centroids after each iteration of distance calculation
- Checking if re-computation is required.

Device:

- Kernel - Calculating Euclidian distance for each data point to each centroid of K clusters.
- Each data point is to be run on a single thread to achieve maximum parallelism.
- Input - Data point, list of all cluster centroids.
- Output - Cluster Number assigned to the datapoint.
- Distance value is intermediate data which is not stored for consequent iterations.

Part 4 - Performance Goals and Validation:

The kernel is to calculate the distance between a given datapoint and k centroids and pick the least distance out of all the k distances. The speedups depend on the number of blocks and threads per block that are launched to calculate the distances.

Given that we have N Datapoints, suppose we run P blocks of the kernel.

Then each block will run N/P threads. It means that the program is cutting down the time taken by P. The minimum performance that we expect is that the runtime is at least reduced P times the time taken to run the program sequentially.

We also need to note that the performance can change based on the parameters passed to the kernels and the number of cores in the system.

Part 5 - Schedule and Division of Work:

Basic plans for the project:

1. All code is managed on GitHub for version control and multiple solutions
2. Since my personal laptop does not contain an NVIDIA Graphics Card, I will be testing all CUDA code on the Ohio Supercomputer.
3. The initial code and testing for the sequential K-Means will be done in my personal laptop or any computer with Python3 installation.
4. I will be writing the sequential K-Means algorithm and test it from scratch.
5. To verify the answers, I will use the sckit-learn version of K-Means.

Schedule:

No division of work – Single person on the team.

Deadlines for different stages of the project:

• Sequential K-Means Implementation:	Nov 16
• Testing and Verification of Results:	Nov 23
• Implementing Data Parallelism for the Distance Calculation:	Nov 30
• Testing and Verifying Results & Speed Ups:	Dec 7

If there the schedule is moved up, then the below things may be tackled:

- Plot performance variations for different input sizes
- Implement & Test CUDA code for Centroid Update.
- Verify Speedups for the Parallel Centroid Update feature.