15-640 Project 4
Parallel K-means with OpenMPI

Report

Udbhav Prasad //udbhavp
Krishna Aditya Gabbita //kgabbita

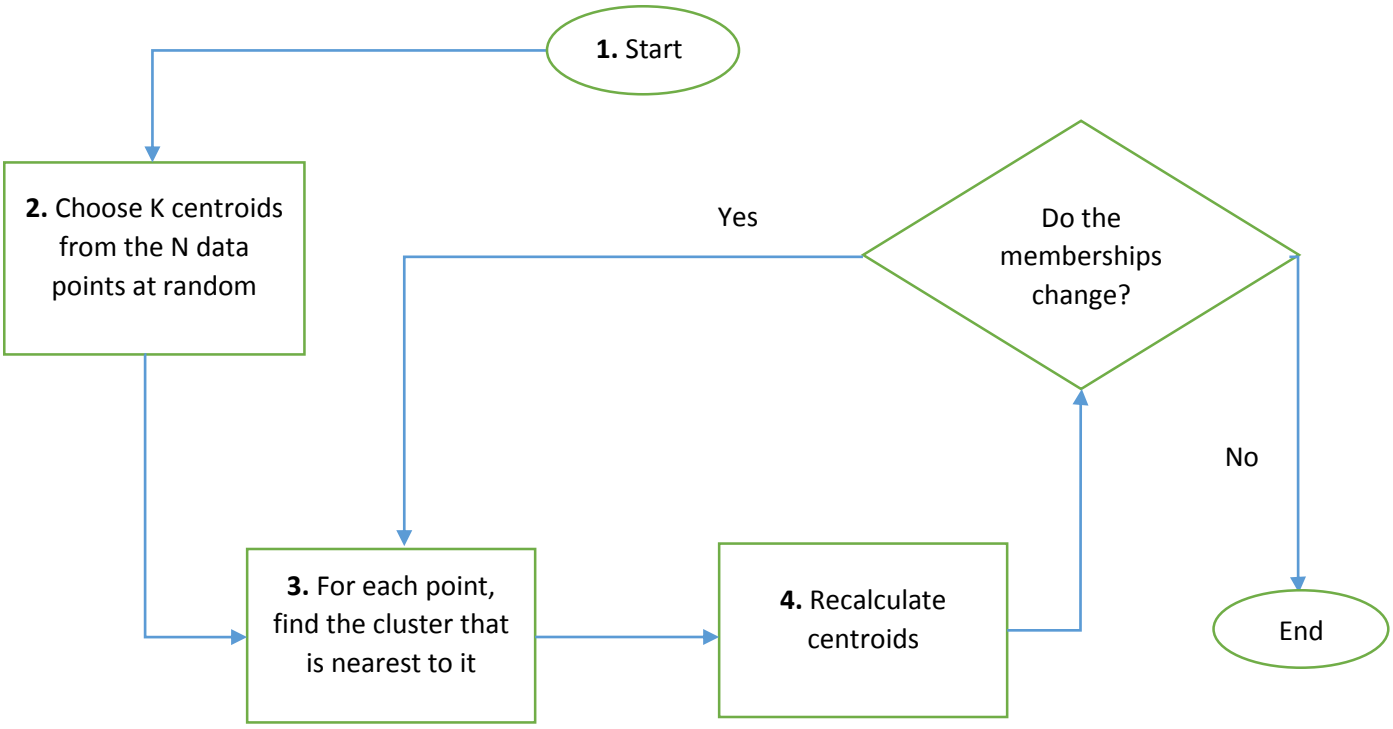
I. Introduction

In this assignment, we apply a clustering algorithm, K-means, to two sets of data, one which consists of points in a *d*-dimensional space, and the other which consists of DNA sequences. We apply both sequential and parallel K-means algorithms on two types of data, and compare the results.

We use a modified version of the given point dataset generator. We have also written a DNA dataset generator, which works along the same lines as the point set generator. Below, we describe the changes we made to the given generator, and how our DNA generator generates data.

II. Sequential K-means

Given a dataset D which has a set of points D_1, D_2… D_N, we want to split it into "clusters", i.e. groups whose elements have some sort of similarity between them. The sequential K-means algorithm can be explained with a flowchart:



III. Parallel K-means

The most expensive step in sequential K-means is step **3**, where we go through all the points, and for each points, calculate the distance between that point and all the clusters. This calculates N*K distances, where N is the number of data points and K is the number of cluster centroids. The distance calculation itself may not be trivial, as in the case of DNA sequences, where we have to go through the entire DNA strand to find the number of different bases.

We parallelize step **3**. If we assume that there are P WORKER machines, we divide our data into P chunks of N/P points each, and make the MASTER send each of the WORKER machines N/P data points and the K cluster centroids. Each machine then separately calculates the new membership of each of the data points that have been sent to it, and return the membership to the MASTER.

Let's look at the ideal case in this parallel algorithm: when we have zero network overhead (of sending and receiving all the data points and centroids). Then in each iteration, the MASTER sends the data chunks to the respective nodes, the nodes calculate the memberships, which now takes (N*K)/P distance calculations, instead of N*K. Thus, in the ideal case, we achieve a speedup of P *in the membership calculation step*.

Our observed conditions, as will become clearer, are much less ideal, and we provide plots comparing the running times of the sequential and parallel algorithms on K-means applied to the *d*-dimensional data points and the DNA sequences.

IV. Data Generation

1. Points in 2-dimensional real space
   We use the data generator provided on the project website to generate a set of 2-d points

2. DNA sequences
   We made some changes to the 2-d points generator.
   a. We changed the command line argument for the maximum coordinate to now represent the length of the DNA sequences to be generated.
   b. We changed the *euclideanDistance* function to now return the number of different characters between two strings that represent DNA sequences.
   c. To generate the cluster centroids, we now sample T real values randomly in the range [0, 4) and take the floor of all the real values, so that we get T values that either 0, 1, 2 or 3 (which could be taken to represent A, T, G and C). Now we generate the points around the centroids by using NumPy's Gaussian normal random generator with the mean as the centroids and the variance as 0.5. This means that there is a danger of sequences being generated with values that are -1. We take care of this by taking the absolute value of all the elements in the generate sequences.

V. Recalculating the cluster centroids

A point to note is the way we calculate the cluster centroids for the two types of data.

1. 2-d data points
   For the 2-d data points, we calculate the cluster centroid by averaging the points that belong to one cluster.
2. DNA sequences
   For DNA sequences of length T, the centroid sequence has, at index *i,* the majority of the i^{th} element of all DNA sequences that belong to the same cluster.

VI. Evaluation

We have evaluated the performance of our serial and parallel implementations of the K-means clustering algorithm for the 2-d points and DNA datasets. For each of the plots below, we have run the algorithm on the dataset 5 times and averaged the results to lessen the influence of outliers.

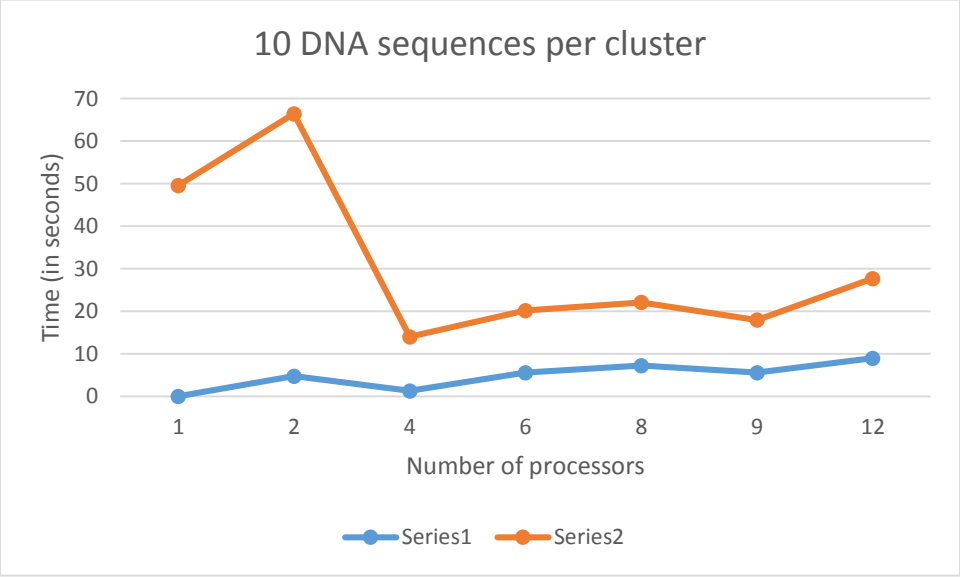


Figure 1
Series 1 – System time
Series 2 – User + System time

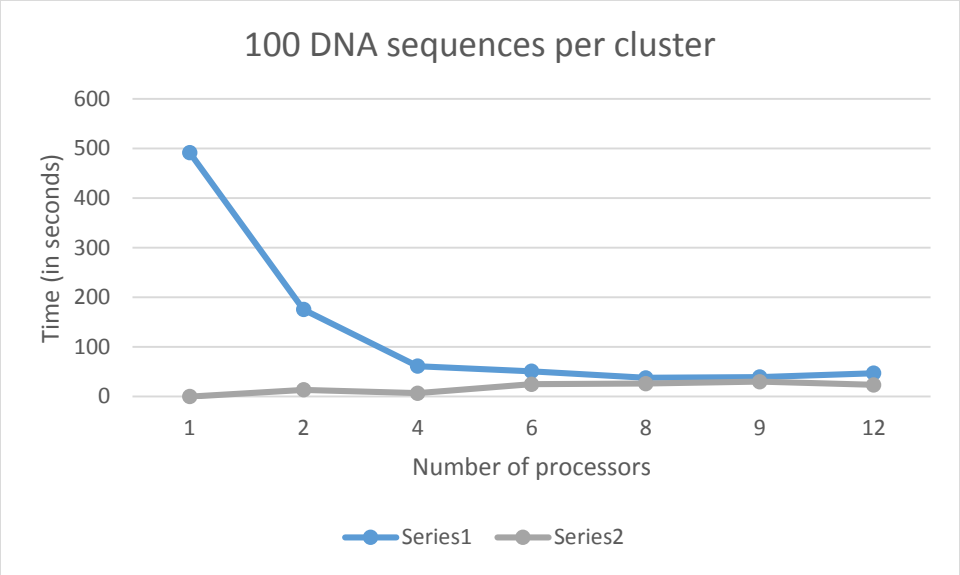


Figure 2
Series 1 – User + System time
Series 2 – System time

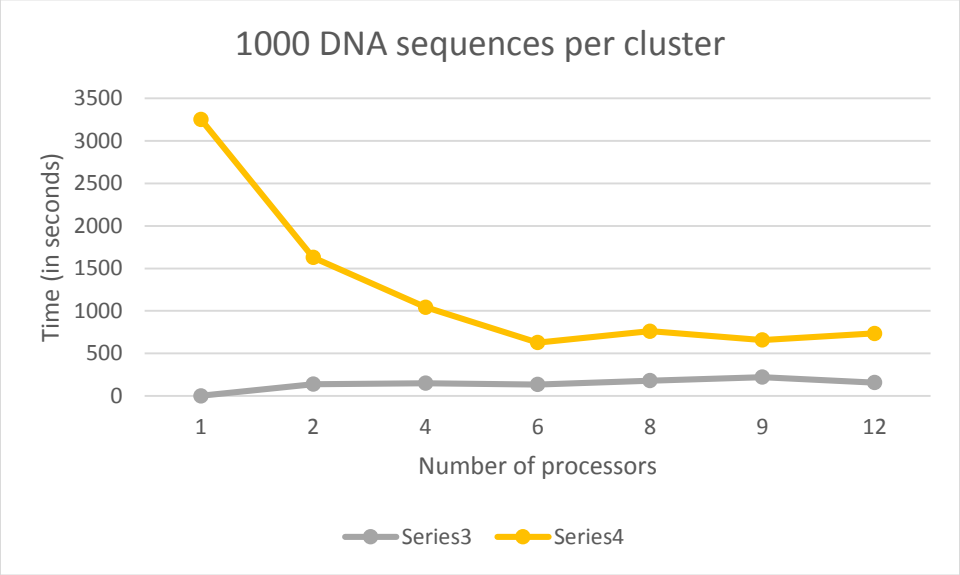


Figure 3
Series 1 – System time
Series 2 – User + System time

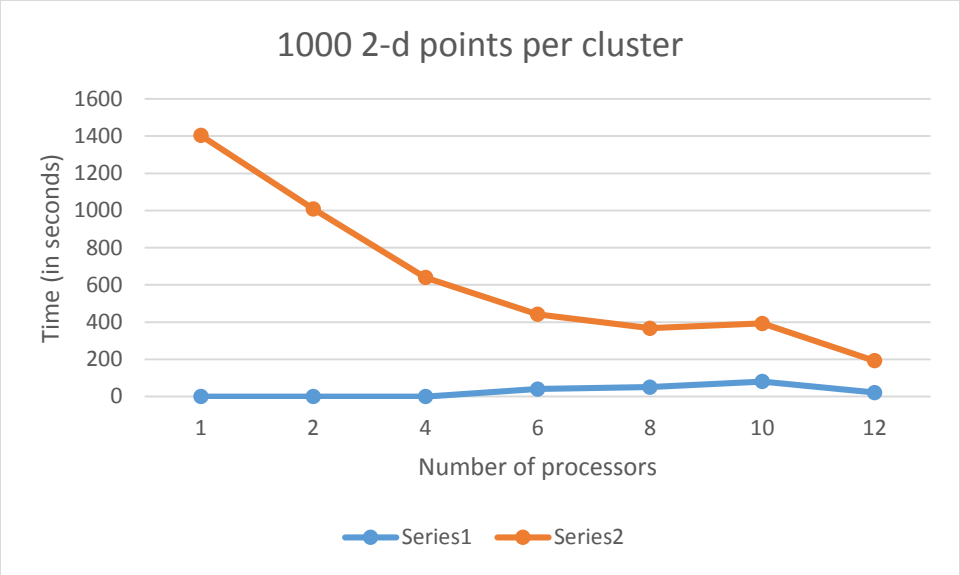


Figure 4
Series 1 – System time
Series 2 – User + System time

1. The length of the DNA strands were taken to be 30.
2. The number of clusters in all the plotted experiments was 1000.
3. We can see a pattern in the results. The “system time” or time taken by kernel processes increases with the number of processors. This makes sense, considering we are transferring data between more and more machines.
4. We also see that our code is able to achieve a reasonable speedup as the number of processors increase, AND as the number of data points (number of clusters X points per cluster) increase.
5. We see better performance with increase in the number of cluster centroids. This is understandable, as we are parallelizing that part of the code which calculates the distances. So naturally, when there are more distances to be calculated, parallelizing it gives better performances. In cases where there aren’t as many distances to be calculated, the cost of communication is not compensated for by the gain in time due to parallelism.
6. Finally, we have not observed linear speedups, which is to be expected, considering the overhead due to file I/O and network communication.