**15-640 Project 4 Report**

Name 1: Rui Zhang

Andrew ID 1: ruiz1

Name 2: Jing Gao

Andrew ID 2: jinggao

**Part I: Design and Features**

The project implements both the sequential and MPI-based K-Means algorithm.

**Introduction of Sequential version of K-Means:**

The sequential K-Means algorithm has three phase: First we need to select k random points from the dataset to be the centroids. Then for each d-dimensional data point, we compute the distance from it to every centroid and label it as the cluster with smallest distance from the point to that centroid. Then we compute k new centroids based on the labels we have computed.

The pseudocode is as follows:

A description...

**Strategy to parallelize the K-Means algorithm:**

Since the computation of distance for every data point is independent to each other, we divide the data points of datasets into **P** parts and each of which is taken care of one process. The strategy can be explained in the following phases:

1. The Master node divides the dataset according to the number of processes. We use MPI\_Scatterv to distribute the data to different processes
2. Use MPI\_BCast to broadcast the random selected centroids.
3. Based on minimum distance strategy (2D points) or most common occurrence strategy (DNA strands), each process will then determine which cluster each point it has belongs to. Meanwhile it will sum them up (2D points) or compute the occurrence of each character in every dimension (DNA strands) for each cluster.
4. When all the processes finishes the computing, we use MPI\_Gatherv method to let the Master to collect results from each process. The Master will then be responsible for using all the results to compute the new centroids.
5. If the new centroids differ from the old centroids under a pre-defined threshold, we will terminate the whole process. Otherwise we continue from step 2 until the termination condition meets.

The pseudocode is as follows (Take the 2D points as an example):

**On Master Process [1]:**

Initialize k different random centroids ci (i = 1….k) selected from the datasets.

**On Master Process [2]:**

BroadCast ci d(i = 1….k) to other P processes.

Divide the xid (i = 1….n) into P parts (the first P – 1 parts have equivalent size and the last one has the remaining part) and scatter them to corresponding process

**On each Process [3]:**

Initialize cntj = 0 for every j = 1….k

Initialize sumjd = 0 for every j=1…..k

For all xid in set X received from Master process

labeli = minj||xid – cjd||

cntj++

sumjd = sumjd + xjd

End for

Send all the cntj and sumjd to the Master Process

**On Master Process [4]:**

Collect the cntj[i] and sumjd [i] (i = 1….P which represents which process it comes from) from every process.

Initialize the new centroids NCjd (j=1….k)

Sum all the cntj[i] and sumjd[i[ to be cntj and sumjd (Use MPI\_Reduce here)

For j=1….k

Update the centroid NCjd = sumjd / cntj

End for

If sum(||NCjd - cjd||) < threshold then

Terminate all processes

Else

Set cjd as NCjd

Continue from step [2]

End if

**Part II: Experimentation and Analysis**

**Running time:**

DNA(5,000,000 strands, 5 cluster):

sequencial: 25.69s

1 processes:15.19s

2 processes:9.80s

3 processes: 9.05s

4 processes: 8.11s

5 processes: 8.14s

6 processes: 7.68s

7 processes: 7.50s

8 processes: 13.67s

9 processes: 13.50s

10 processes:13.51s

11 processes:13.72s

12 processes:13.37s

2D(5,000,000 points, 5 cluster):

sequencial: 26.80s

1 processes: 8.46s

2 processes: 8.32s

3 processes: 7.58s

4 processes: 6.79s

5 processes: 8.03s

6 processes: 12.88s

7 processes: 12.73s

8 processes: 12.44s

9 processes: 12.30s

10 processes: 12.64s

11 processes: 12.41s

12 processes: 12.09s

**Graph:**

A description...

A description...A description...

**Analysis:**

Comparing the running times of the sequential and MPI-based k-means clustering programs, we can find the following trends.

A. Sequential programs run much slower than MPI-based programs, even when MPI-based programs use only one process. This is a specific for our implementation. Sequential programs are implemented in Java and utilizes mature data structures like multiple levels of ArrayList, while MPI-based programs are implemented in C and utilizes pointers, arrays and other highly efficient data structures. Ideally, if sequential and MPI-based programs are implemented in same language and with same data structures, the performance of sequential programs should be slightly better than using MPI one process, considering the overhead MPI is adding to the original program.

B. Initially, MPI-based programs runs faster with more processes, indicating the power of MPI and parallelism. Scaling up of processes does not lead to boosting of the performances, only enhances the performance to a certain degree. Above trends can be seen in both 2D points and DNA clustering. Using processes from 1 to 4 for MPI-based programs lowers the running time substantially, though with using more processes, the enhance in performances become less obvious.

C. This advantage of scalability does not last for very long, when more and more processes are running in parallel with MPI mechanism, the communication overhead becomes obvious and then overwhelms the performance at some point immediately after a optimal number of processes. After that the already poor performance becomes stable with even more processes, indicating a parallel scaling up of computational capability versus communication overhead in MPI. The above trends can be seen in both 2D points and DNA clustering.

D. The sweet point, i. e., the optimum number of processes, is about 4 to 7, and it can be different in different MPI programs. For 2D points clustering, there is not much space for scalability, 4 processes appears to lead to best performance but this actually is comparable to using only one process; when processes add to 6, the turning point is obvious and MPI communication overhead is already dominant. For DNA clustering, the initial scaling up of performance is obvious, and the performance enhancement slows down and approaches optimum performance with 7 processes. After that, the performance is overwhelmed by MPI communication overhead.