# 15640 Project 4 MPI -- Clustering and DNA Report

Name: Yue Zhuo Andrew ID: yuezhuo Name: Yusi Zhang Andrew ID:yusiz

## Part1. Parallel K-Means algorithm

This project implements the K-means algorithm handle the 2D-points and DNA data in both sequential and parallel ways.

Since the computation of the distance of every data point is independent from each other, we can divide the data points into p parts,( p is the number of processors), so every processor can compute one part of the data points.

n = total amount of the data points

p = total amount of processors

c = total amount of centroids

1. The Master selects c data points as the initial centroids randomly from the dataset
2. The Master send the list of centroids to the processors via MPI
3. Every processor is responsible for computing n/p data points.

   The data points are assigned to different processors based on the processor's rank:

   For the processors with the rank of i, it is responsible for the data points with the indexes from n / p * (i - 1) to n / p * i
4. The processor calculates the distances between the data point and each centroids, to figure out which centroid is nearest to the data point, and label the data point as the cluster surrounding the centroid
5. Using the AllReduce function of MPI, we combine the values of every data point in the same cluster, and calculate the new centroid of the cluster
6. The Master send the list of new centroids to the processors
7. Repeat step 4,5,6, until:
   1) either a maximum number of iterations has been performed
   2) or the total distances between the new centroids and the old centroids is less than a threshold.
8. Get the final result of the c centroids of the dataset.

Pseudocode (Using the 2D points as an example):

Consider a set of n data points.

```
1.  Master:
    Initialize an array of cluster centroids $c_1$, $c_2$, …, $c_c$ selected randomly from the dataset
    Boardcast the centroids to the p processors
```

```
2. Each processor:
   Receive the centroids $c_i$(i = 1, 2, …, c)from the Master
   Set pointToCentroids as a list with the length of n/p
   Set subset as a list of the processor's part of the dataset
   for each point in subset
       var minDistance = MAX_VALUE
       var index = 0
       for each centroidIndex from 0 to c
           var tmpDistance = calculateDistance(point, centroid[centroidIndex])
           if tmpDistance < minDistance
               minDistance = tmpDistance
               index = centroidIndex
       pointToCentroids[index] = centroidIndex
   var xSum
   var ySum
   var clusterSize
   for each point's index in the subset
       cluster = pointToCentroid[index]
       xSum[cluster] += point.x
       ySum[cluster] += point.y
       clusterSize++
3. AllReduce:
   Combine the xSum and ySum of each cluster
   for each cluster
       $x_i = \sum xSum[i] / clusterSize_i$
       $y_i = \sum ySum_i / clusterSize_i$
       new centroid = new Point($x_i$, $y_i$)
4. Master sends the new centroids to the processors
5. Each processor:
   if iterations >= threshold
       terminate
   else
```

Update the centroids

Repeat step2, 3, 4

## Part2. Experimentation and analysis

### Experiment results:

2D Points:

Sequential: 9758 ms

Parallel：

Process number is the total number of processors, including the Master and the Slaves.

2 processes: 10117 ms

3 processes: 5738 ms

4 processes: 5722 ms

8 processes: 5963 ms

12 processes: 6845 ms

DNA strands:

Sequential: 10501 ms

Parallel：

2 processes: 7386 ms

4 processes: 4756 ms

8 processes: 5322 ms

12 processes: 8382 ms

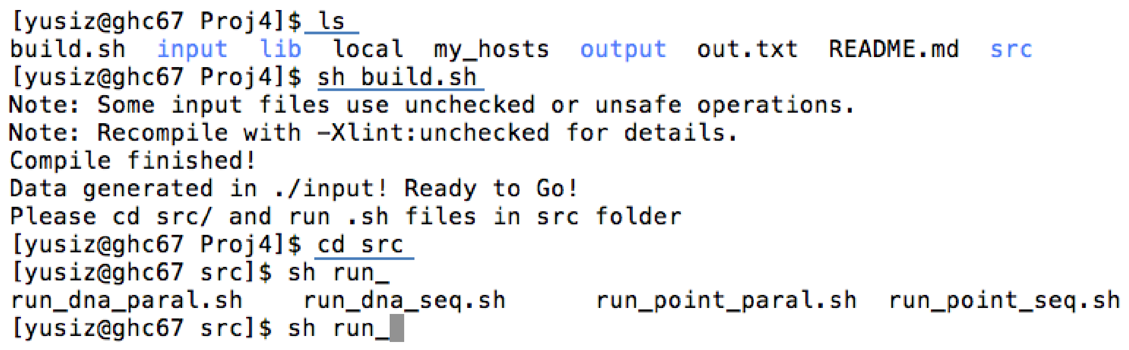
### Analysis:

Comparing the running times of the sequential and parallel K-means clustering programs, we can come to the following conclusions.

1. The sequential programs run much slower than the parallel programs, both with the dataset of 2D points and DNA strands. Both the Sequential programs and the MPI-based programs are implemented in Java. However, the Sequential programs use some complex data structures such as HashMaps and ArrayLists, while the Parallel programs use the elementary data structures such as Array with higher efficiency.

2. Initially, the more processors the MPI-based program has, the faster it will run, which shows the efficiency of parallelism. With from 1 to 4 computing processors(slaves), the more processors we have, the faster the running time is.
3. The advantage of parallelism turns to an disadvantage when we have four or more processors. above 4 processors, the more processors we have, the slower the running time is. This is because the MPI communication among different processors overwhelms the performance of parallel computing.
4. The sweet spot is between 4 and 8 processors, and it can be different with different MPI programs.

## Part3. Test and run



```
[yusiz@ghc67 Proj4]$ ls
build.sh  input  lib  local  my_hosts  output  out.txt  README.md  src
[yusiz@ghc67 Proj4]$ sh build.sh
Note: Some input files use unchecked or unsafe operations.
Note: Recompile with -Xlint:unchecked for details.
Compile finished!
Data generated in ./input! Ready to Go!
Please cd src/ and run .sh files in src folder
[yusiz@ghc67 Proj4]$ cd src
[yusiz@ghc67 src]$ sh run_
run_dna_paral.sh    run_dna_seq.sh      run_point_paral.sh  run_point_seq.sh
[yusiz@ghc67 src]$ sh run_
```

Run Sequential DNA program:

sh run_dna_seq.sh

Run MPI-based DNA program:

sh run_dna_paral.sh

to run with different processes, you need to modify the run_dna_paral.sh file:

```
mpirun -np process_Number java MainDNA cluster_Number
```

Run Sequential 2D points program:

sh run_point_seq.sh

Run MPI-based 2D points program:

sh run_point_paral.sh

to run with different processes, you need to modify the run_dna_paral.sh file:

```
mpirun -np process_Number java MainDNA cluster_Number
```