15640 Project 4: Clustering Data Points and DNA Strands Using MPI

Kuang-Huei Lee, kuanghul
Yangdong Liao, yangdonl

This report contains 3 sections:

1. The first section describes our implementation of sequential and parallel computation
2. The second section evaluation of sequential and parallel computation
3. The last section will give the TA's a guide of how to build, deploy, and run the program

Section 1 - Implementation Overview

Cluster data generation

Generate 2-D point clusters: We directly use the provided Python code to generate points with x and y values which are float number from 0 to 10

Generate DNA strand clusters: we generate strings composed of 100 characters which are all ACGT. the pseudo-code is described as follows

```
Parameters:
k = number of clusters, b = number of data in each clusters, length

function generate_DNA_strands(k, b)
      memory DNA_strand_list;
      define bases = ('A', 'C', 'G', 'T')
      foreach cluster
            foreach position in strands
                  randomly pick a character from bases to form
                  cluster centroid
            for i = 1:b
                  randomly choose how many place will be changed
                  randomly choose the position to change
                  copy centroid
                  foreach position_to_change in centroid strand
                        pick a character from bases expect the original one
                        push generated DNA strands into DNA_strand_list
```

How do we evaluate distance and centroid

For 2-D points:

Distance: evaluated by euclidean distance

Centroid: evaluated by average value of x and y values of each points in a set

For DNA strands (lengths are equal):

Distance: evaluated by number of character

Centroid: evaluated by majority character on each position in all strings in a set

Sequential K-Means implementation

For sequential K-Means, we followed the pseudo-code described in project instruction and implemented the program in C++

Parallel K-Means implementation

Parallel K-Means is also implemented the program in C++. The program structure is described as follows

Parameters:

n: number of points / DNA strands to cluster
k: number of desired clusters
m: number of participants excluding the master

Pseudo-code:

function mpi_kmeans(input_data, n, k, m)

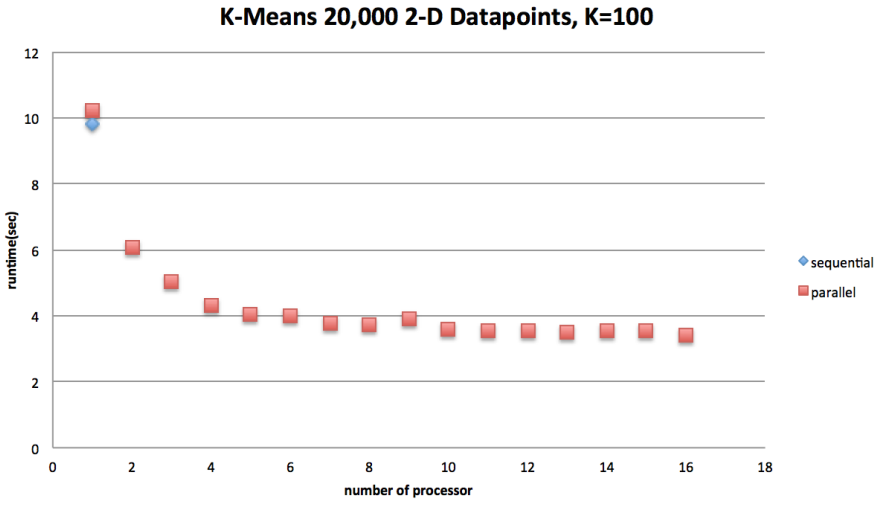
1. Master read data and randomly pick training data points from data set as initial centers
2. MPI_Bcast to broadcast the initial centers to each participant
3. MPI_Scatterv to scatter n data into m sections of size n/m, denote each section S_i, and send to all participants
4. Each participant P_i, divide data in S_i according to the closest centroid for each data and compute the partial means of each cluster.
5. MPI_reduce to sum up the partial means from participant P_i to get the new global means of each cluster
6. Recursively do 4-6 and update global means until the means converge

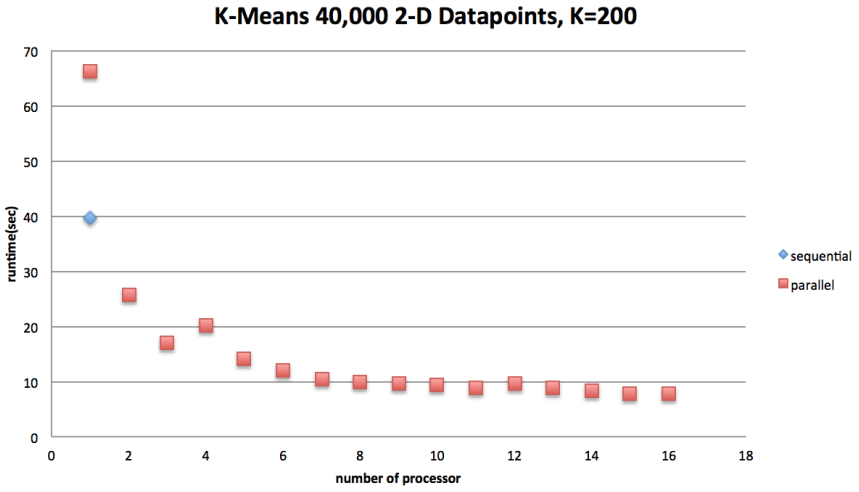
Section 2 - Result Evaluations

Result of 2-D points:

Our test for 2-D points is based on two cases: 300 clusters and 200 2-D points in each cluster; 100 clusters and 200 2-D points in each cluster. Runtime of MPI version of K-Means depends on the number of processors used. Given the graph below, we can find that the sequential version only run faster than the MPI version running 1 processors. We assume the additional cost came from communication overhead. Note that number of data points has significant effect on processing time - 2 times of points yield larger factor on processing time when computation load is large on few nodes.

The effect of processor number increase is dramatical initially. The bottleneck was reached around 8 processors from which the boosting effect became insignificant.

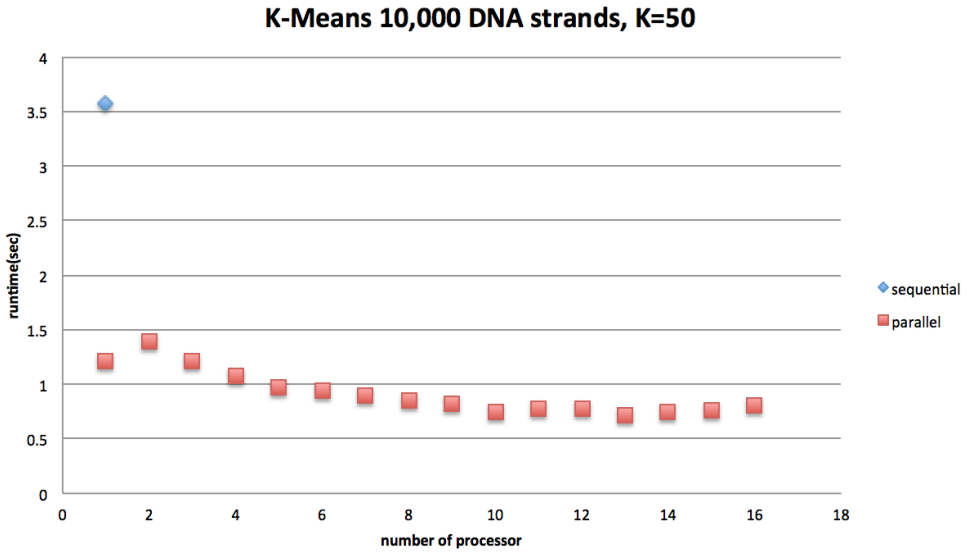


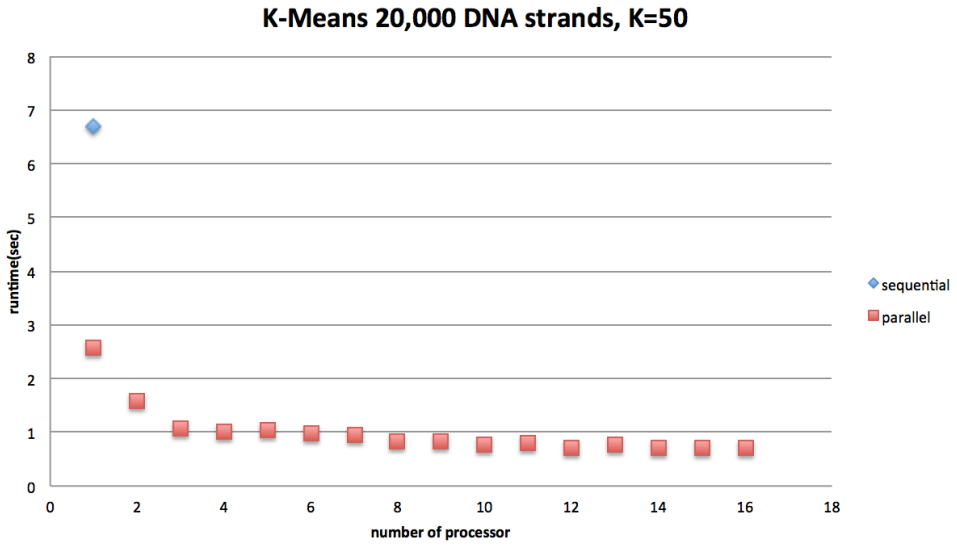


Result of DNA strands:
Similar to 2-D points, our test for DNA strands is based on two cases: 50 clusters and 200 strands in each cluster; 50 clusters and 400 strands in each cluster. DNA strand has fixed length of 100 characters. The sequential version is slower than parallel version of 1 processor. The reason we assume is because in sequential version we use more C++ library to process string, whereas in parallel version the program is more C-style.

A interesting fact which is more significant in DNA strands clustering test is we actually encountered a sweet spot in the trend. We assume this happens because communication overhead overwhelm the benefit of distributed computation.

Another interesting fact is that we can see a bump happen where process number goes from 1 to 2 in 10000 strands case. We assume the reason of this bump is because the communication overhead kicked in.





Section 3 - Compile and Running

Data generation

You can **cd** into */kmeans_sq/input* or */kmeans_pl/input* and type "**sh kmeans.sh**" in bash. This will generate 100 clusters and 200 2D-points in each cluster in *data/points.csv*; 50 clusters and 400 DNA strands in each cluster in data/DNA_strand.txt. You can edit kmeans.sh script file to generate different data

...or you can **cd** into /randomclustergen to generate using python scripts

Generate 2D points
python ./randomclustergen/generatepoint.py -c $k_point -p $b_point -o data/points.csv -v 10
$b_dna = Number of Points
$k_dna = Number of Cluster

How to generate DNA strands
python ./randomclustergen/generatednastrand.py -c $k_dna -p $b_dna -o data/DNA_strand.txt -v 100
$b_dna = Number of Points
$k_dna = Number of Cluster

How to build and run sequential K-means

1. type "**cd into /kmeans_sq"** in bash
2. type **"make"**
3. follow the data generation guideline if needed
4. type **"./main -k [cluster number] -f [filename] [-p|-d]"** option **-p** runs points K-means, option **-d** runs DNA strands K-means
   e.g. **"./main -k 50 -n 20000 -f ../input/data/DNA_strand.txt -d"**
5. Program shows the final result and runtime of each node

How to build and run parallel K-means

1. type "**cd into /kmeans_pl"** in bash
2. type **"make"**
3. follow the data generation guideline if needed

4. type **“mpirun -np 4 -machinefile hosts main -k [cluster number] -n [line counts of data file] -f [filename] [-p|-d]”** option **-p** runs points K-means, option **-d** runs DNA strands K-means
   e.g. **“mpirun -np 4 -machinefile hosts main -k 100 -n 20000 -f ../input/data/points.csv -p”**
   (If the host in machinefile does not work, feel free to edit **hosts** file)
5. Program shows the final result and runtime of each node