CPSC 479

Dr. Bein

Project Assignment 2

Group members:

Huy Ho hho114@csu.fullerton.edu

Chandler Ebrahimi csebra@csu.fullerton.edu

Darren Vu vuchampion@csu.fullerton.edu

I. Problem statement:

Sometimes we just want to see how the data is organized, and that's where clustering comes into play. The word 'clustering' means grouping similar things together, and the most commonly used clustering method is K-Means because of its simplicity. So in this Project, we implement k mean clustering with mpi to show how to apply a high performing computers application in data science.

II. Implement

This program is written in C and using Mpi

Step 1: Create random data point numbers from 0 to 1, and assign partial number points to each processor. (Use MPI_Scatter)

Step 2: Choose the first few K points as centroids and assign them to cluster.

Step 3: In each processor for each data point find its cluster by calculating its distance with centroids.(User MPI_bcast for centroid list and mean distance tracking)

Step 4: Calculate the mean distance each cluster, and update centroids for each cluster.(Use MPI_Reduce to get points distance from each process)

Step 5: Go to step (3) and repeat until the number of iterations > 10000 or mean distance has changed is less than 0.

Step 6: Label all points with its cluster, print results.

III. Pseudocode

```
BEGIN Main()

K = amount of cluster
Dimension = dimensions of data
totalPoint = total points inputted
MPI_Init
Int rank, size
MPI_Init(NULL, NULL);
int rank, sizeRank;
Set random seed
MPI_Comm_rank(MPI_COMM_WORLD, &rank);
MPI_Comm_size(MPI_COMM_WORLD, &sizeRank);
MPI_Barrier(MPI_COMM_WORLD);
Set pointsPerProcess = totalPoints / sizeRank
Start = MPI_Wtime()
Set counter = 0
Declare float pointers =  recvPoints, points, centroids
Declare int pointers = count, labels

Set recvPoints to the size of pointsPerProcess * dimension
Set points to the size of k * dimension
Set counts to the size of k
Set centroids to the size of  k * dimension
Set labels to the size of pointsPerProcess

float *allPoints = NULL;
float *pointSums = NULL;
int *clusterCounts = NULL;
int *allLabels;

If (rank == 0)
        allPoints = createRandomNums(dimension * totalPoint)
        For i = 0 to k * dimensions
                Centroids[i]  = allPoints[i]
        initialCentroids(centroids,k, dimension)
        Counter++
        Set pointSums to size of k * dimensions
        Set clusterCounts to size of k
        Set allLabels to size of sizeRank * pointsPerProcess
MPI_Scatter(allPoints,dimensions * pointsPerProcess,MPI_FLOAT, recvPoints,
                dimension * pointsPerProcess, MPI_FLOAT, 0, MPI_COMM_WOLRD)
```

```
distance = 1;

While (distance > 0 and counter < MAX_ITERATIONS)

        MPI_Bcast(centroids, k * dimension, MPI_FLOAT, 0, MPI_COMM_WORLD);
        for  i = 0 to k * dimension; i++
                points[i] = 0.0;
         for i = 0 to k
                counts[i] = 0;
        float *pointsAssign = recvPoints;
        for i = 0 to pointsPerProcess and pointsAssign += dimension
                int clusterNum = assignLabel(pointsAssign, centroids, k, dimension);
                counts[clusterNum]++;
                addPoint(pointsAssign, &points[clusterNum * dimension], dimension)

        MPI_Reduce(points, pointSums, k * dimension, MPI_FLOAT, MPI_SUM, 0,
        MPI_COMM_WORLD);
        MPI_Reduce(counts, clusterCounts, k, MPI_INT, MPI_SUM, 0, MPI_COMM_WORLD);

         if (rank == 0)
                For i = 0 to k
                   For j = 0 to dimensions
                      pointSums[dimension * i + j] /= clusterCounts[i];

                distance = distanceBetween(pointSums, centroids, dimension * k);
                for i = 0 to k* dimension
                   centroids[i] = pointSums[i];

                notifyUpdateCentroids(centroids, k, dimension, &counter);
        MPI_Bcast(&distance, 1, MPI_FLOAT, 0, MPI_COMM_WORLD);
        counter++;

pointsAssign = recvPoints;
For i = 0 to pointsPerProcess and pointsAssign += dimension
        Labels[i] = assignLabel(pointsAssign, centroids, k, dimension)
```

```
MPI_Gather(labels, pointsPerProcess, MPI_INT,
            allLabels, pointsPerProcess, MPI_INT, 0, MPI_COMM_WORLD);

MPI_Barrier(MPI_COMM_WORLD)

End = MPI_Wtime()
MPI_Finalize
Print execution time

END MAIN
```

IV. How to run the program

Go to the project directory, and use mpicc to compile the main.c file (mpicc main.c), then run the compile file with four input arguments.

mpirun -n "number process" a.out "k number or number of cluster" "number dimension" "number points"

For example: mpicc main.c && mpirun -n 6 a.out 2 2 100
//will use 6 process with 2 kmean and 2 dimension for 100 data points

Optional:
For better understanding how k-means work, we develop a program to output image files which show how the centroids change. However this will only work for 2 dimensions and the k number limit to 3.

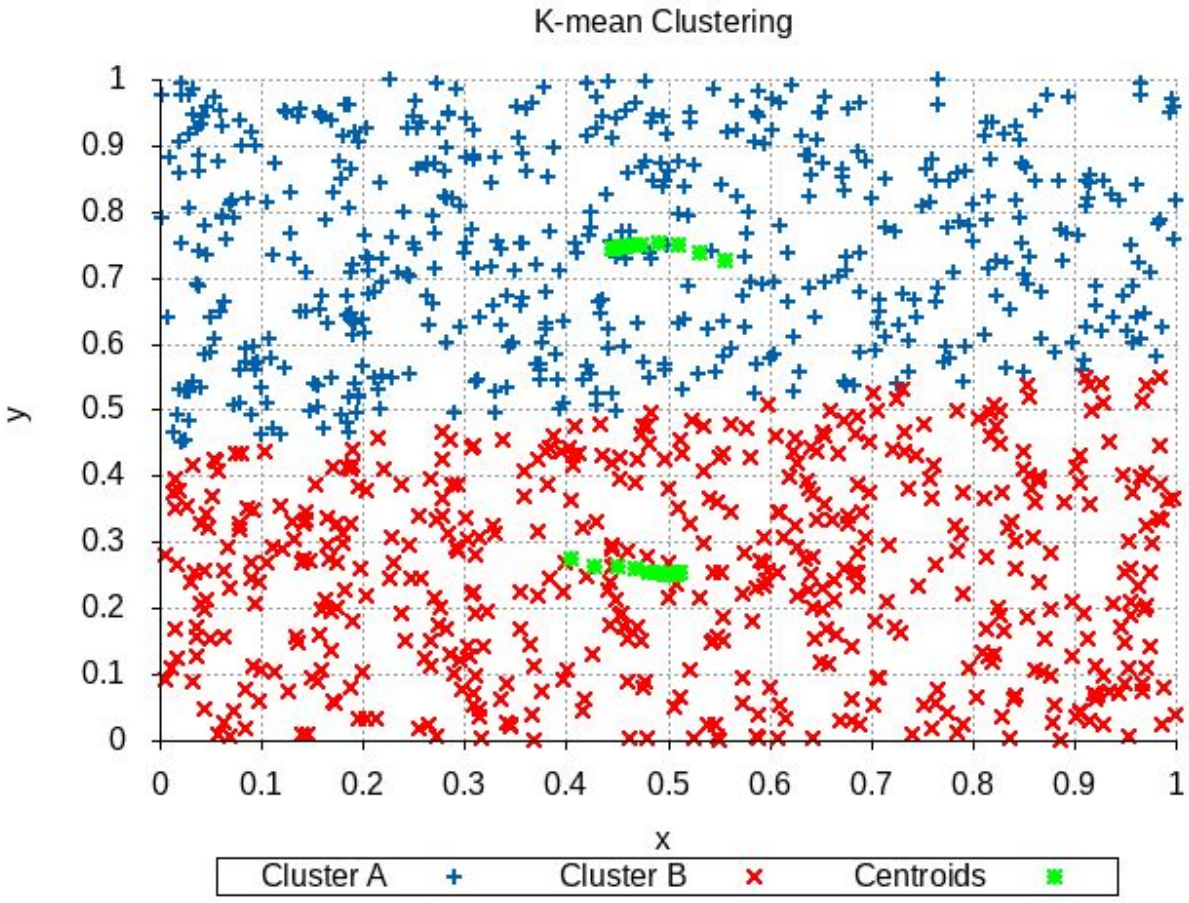
Note: This method require gnuplot program

- 2 Kmean Clustering 2 dimension with graph

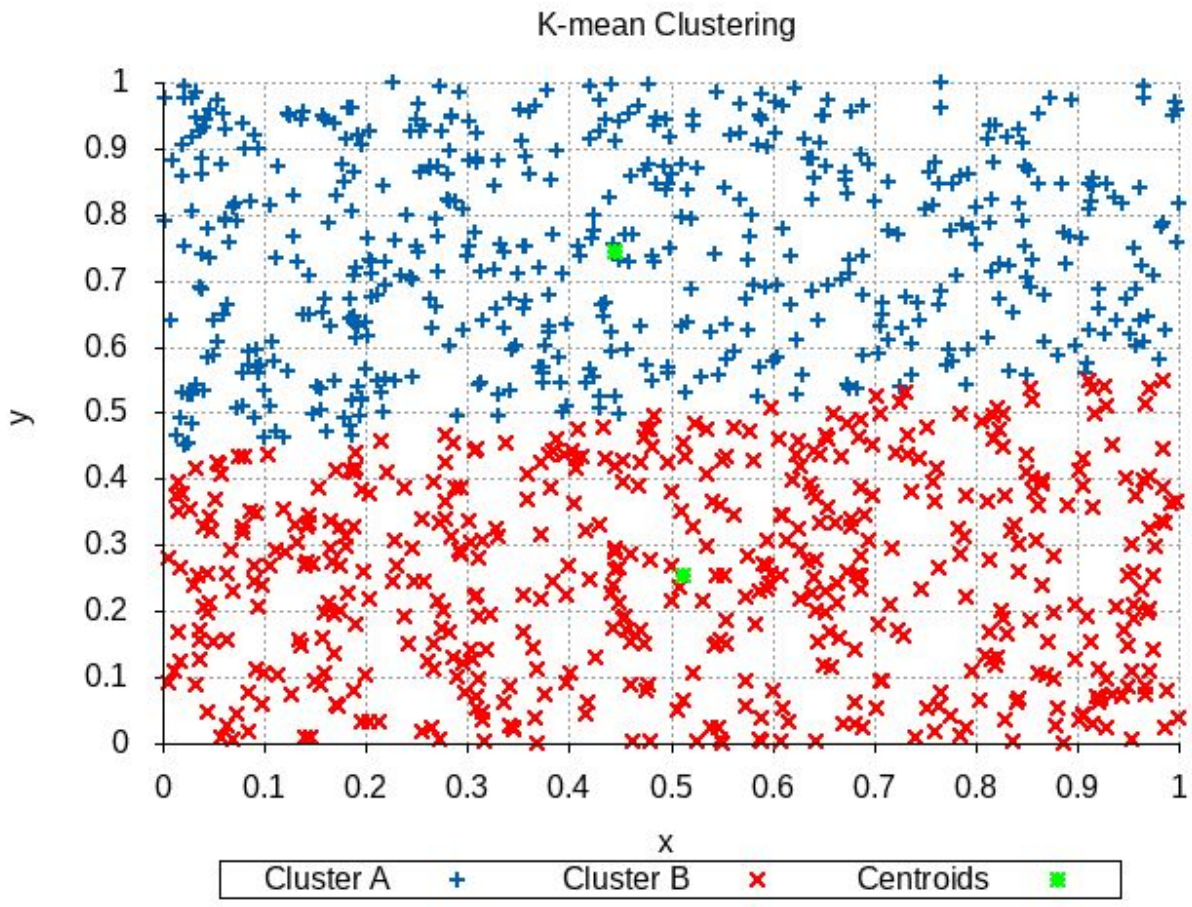
Use this command to create kmean cluster with 2 clusters and 100 data points per process graph:
Standard: mpicc main.c && mpirun -n "number process" a.out 2 2 "number points" && gnuplot graphs/2_kmean_graph.gp
Example: mpicc main.c && mpirun -n 10 a.out 2 2 1000 && gnuplot graphs/2_kmean_graph.gp

The output will look like this:

Changing centroids Process



Final centroid when finish



- 3 Kmean Clustering 2 dimension with graph

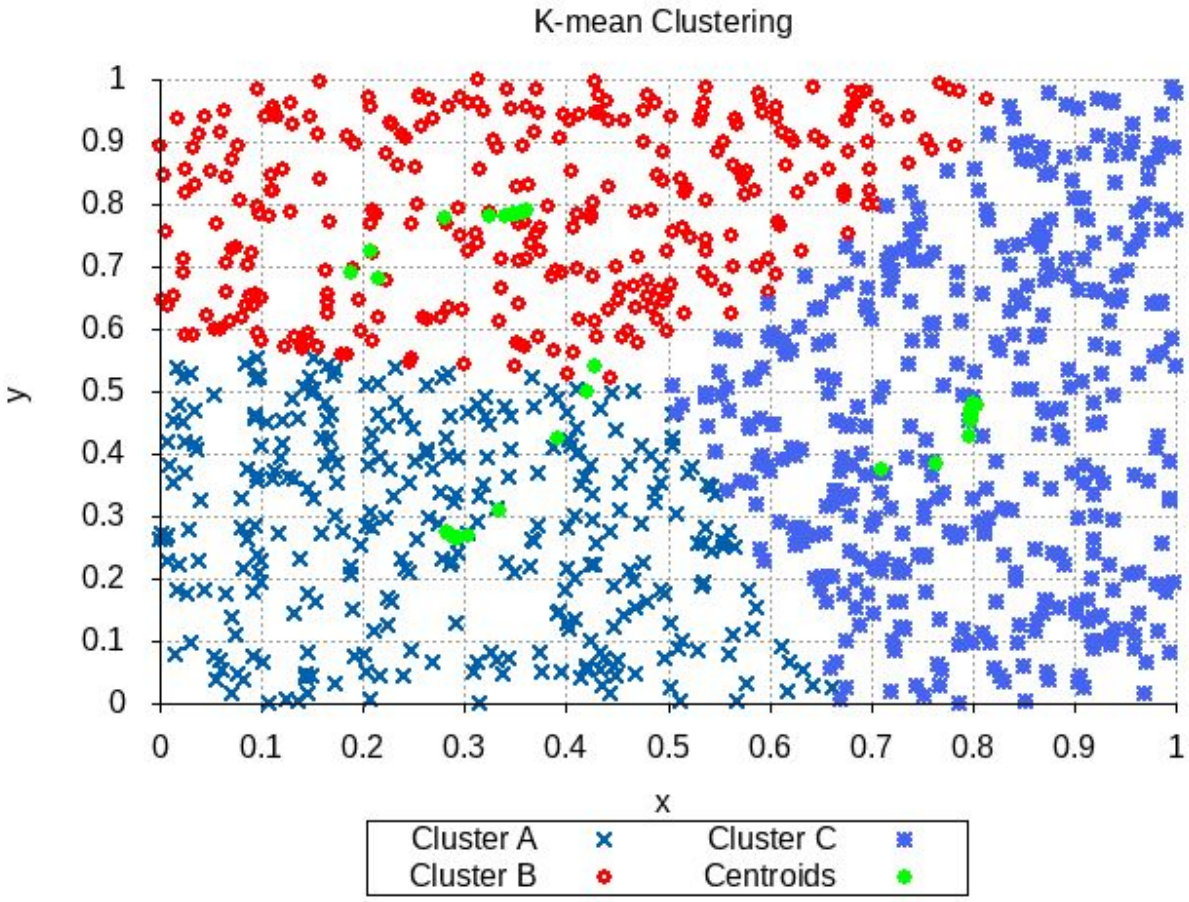
Use this command to create kmean cluster with 3 clusters and 100 data points per process graph:

Standard: mpicc main.c && mpirun -n "number process" a.out 3 2 "number point" && gnuplot graphs/3_kmean_graph.gp

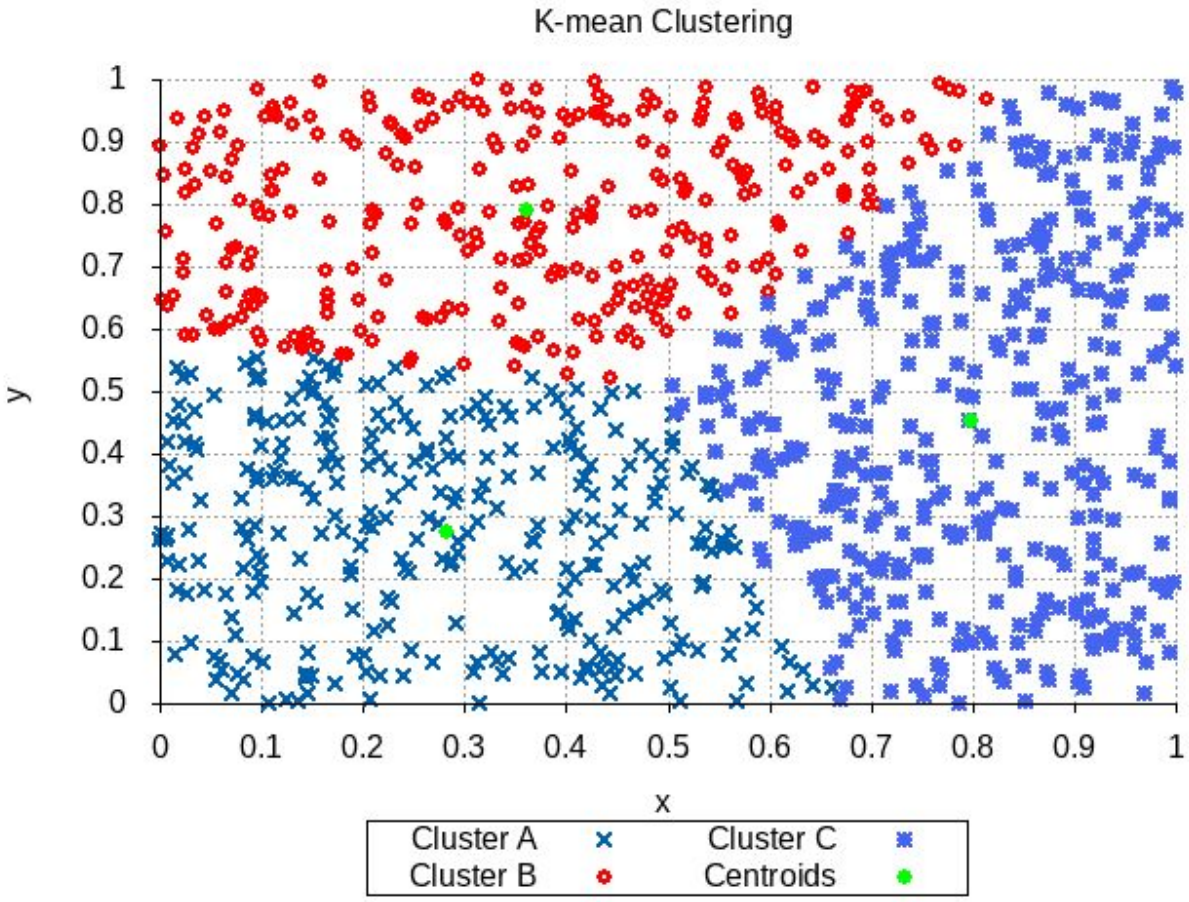
Example: mpicc main.c && mpirun -n 10 a.out 3 2 1000 && gnuplot graphs/3_kmean_graph.gp

The output will look like this:

Changing centroids Process



Final centroid when finish

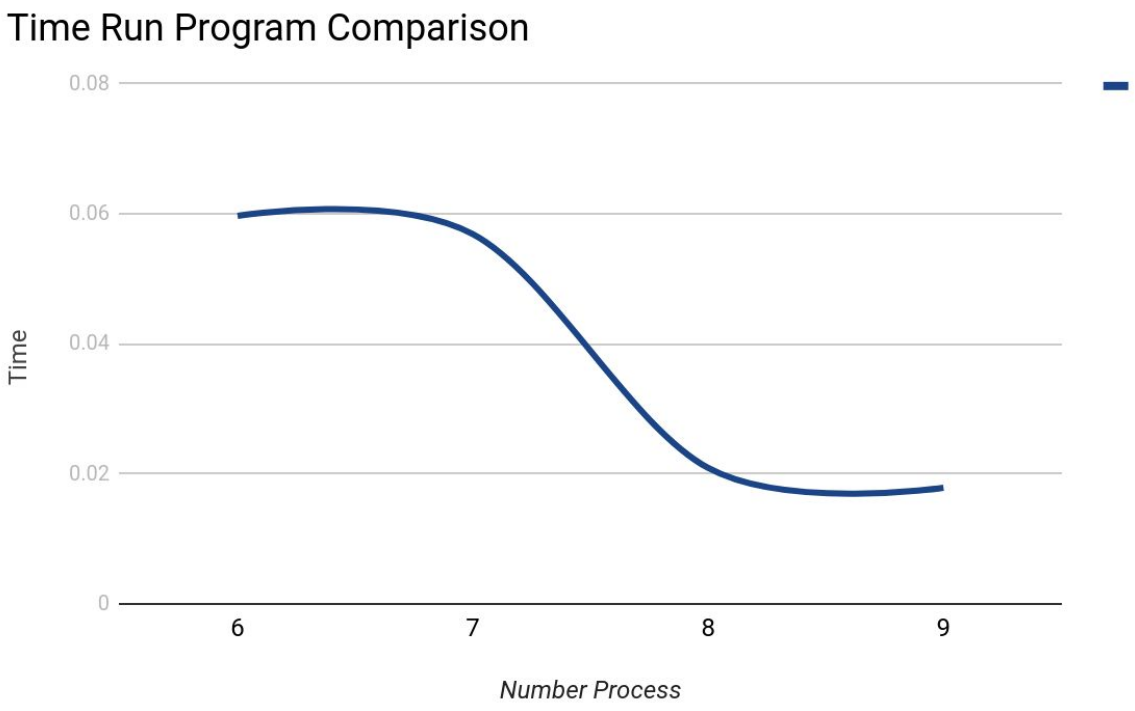


Command to run both method above “make”

V. Result From Run Program

Table time run program with k number is 2, dimension is 2 and total point 1000 but N number process increase

N	6	7	8	9
Time	0.059706	0.056879	0.020904	0.017869



Conclusion: The more process use to clustering, the faster program will be

Run sample screenshots: