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

1. Create random data point numbers from 0 to 1, and assign partial number points to each processor. (MPI_Scatter)
2. Choose the first few data points as *centroids* and assign them to clusters.
3. In each processor for each data point choose the next one in the array tol make up one point based on dimension, for example 2 dimensions will be x and y. Then find it's cluster by calculating it's distance with centroids using the Euclidean distance formula. (MPI_bcast for centroid list and mean distance tracking if mean reach equally among centroids)
4. Calculate the mean distance of each cluster which is also a new centroid, and update centroids for each cluster. (MPI_Reduce to get points distance from each process)
5. Repeat Step 3 until the number of iterations is greater than 10,000 or mean distance has changed less than 0, it means all centroids have equal distance to each other.
6. Label all points with it's cluster and print the 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.

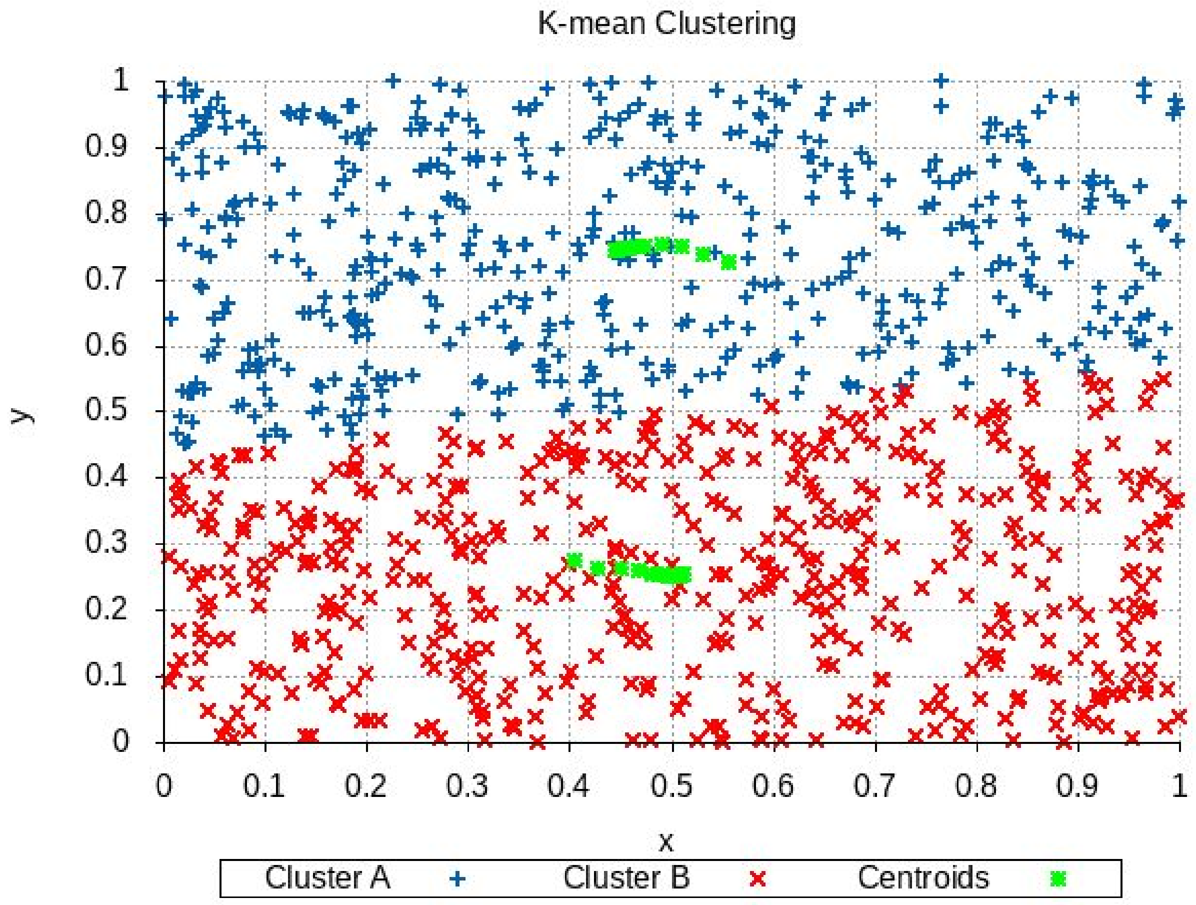
Note: This method require gnuplot program

- 2 Kmean Clustering 2 dimension with graph

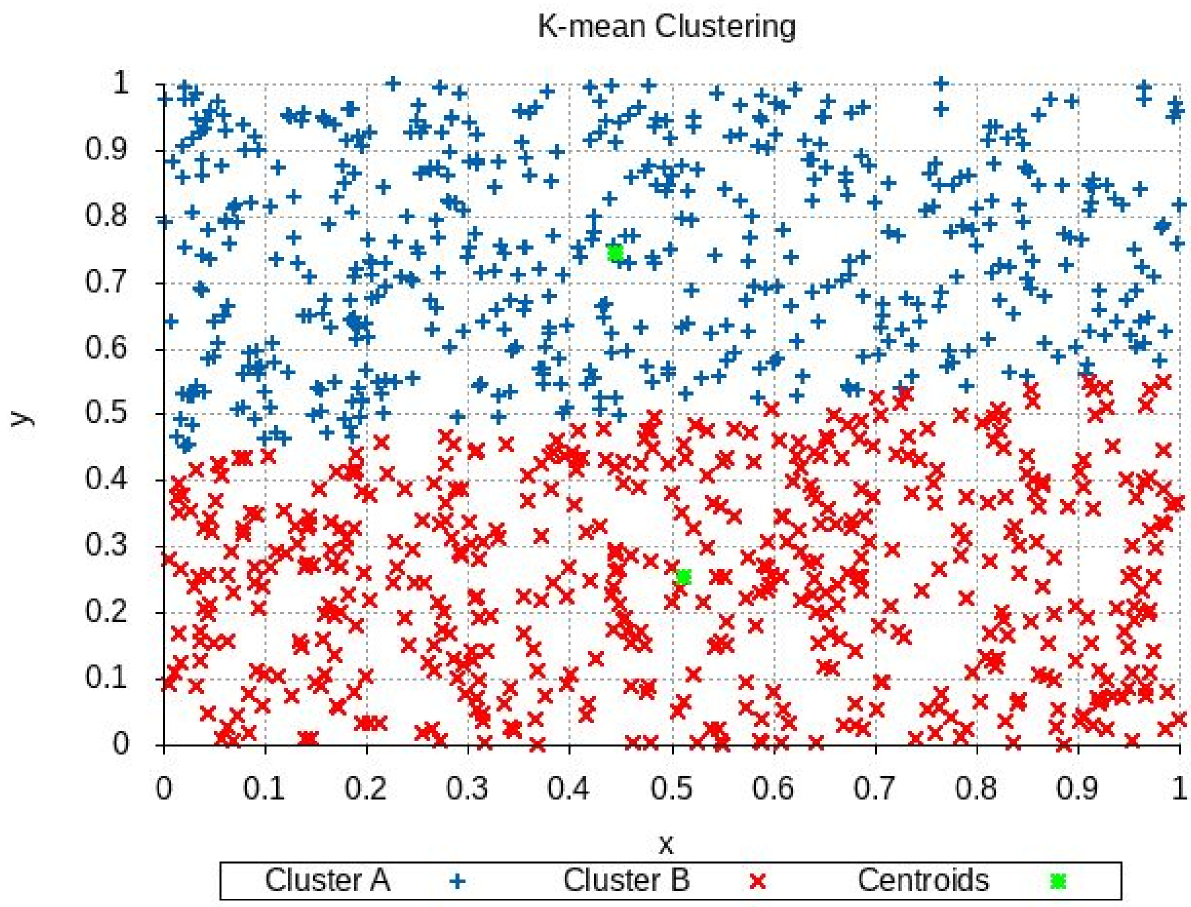
Use this command to create kmean cluster with 2 clusters and 100 data points per process graph:
Standard: `mpicc main.c && mpirun -n "number process" a.out 2 2 "number points" && gnuplot graphs/2_kmean_graph.gp`
Example: `mpicc main.c && mpirun -n 10 a.out 2 2 1000 && gnuplot graphs/2_kmean_graph.gp`

The output will look like this:

Changing centroids Process



Final centroid when finish



- 3 Kmean Clustering 2 dimension with graph

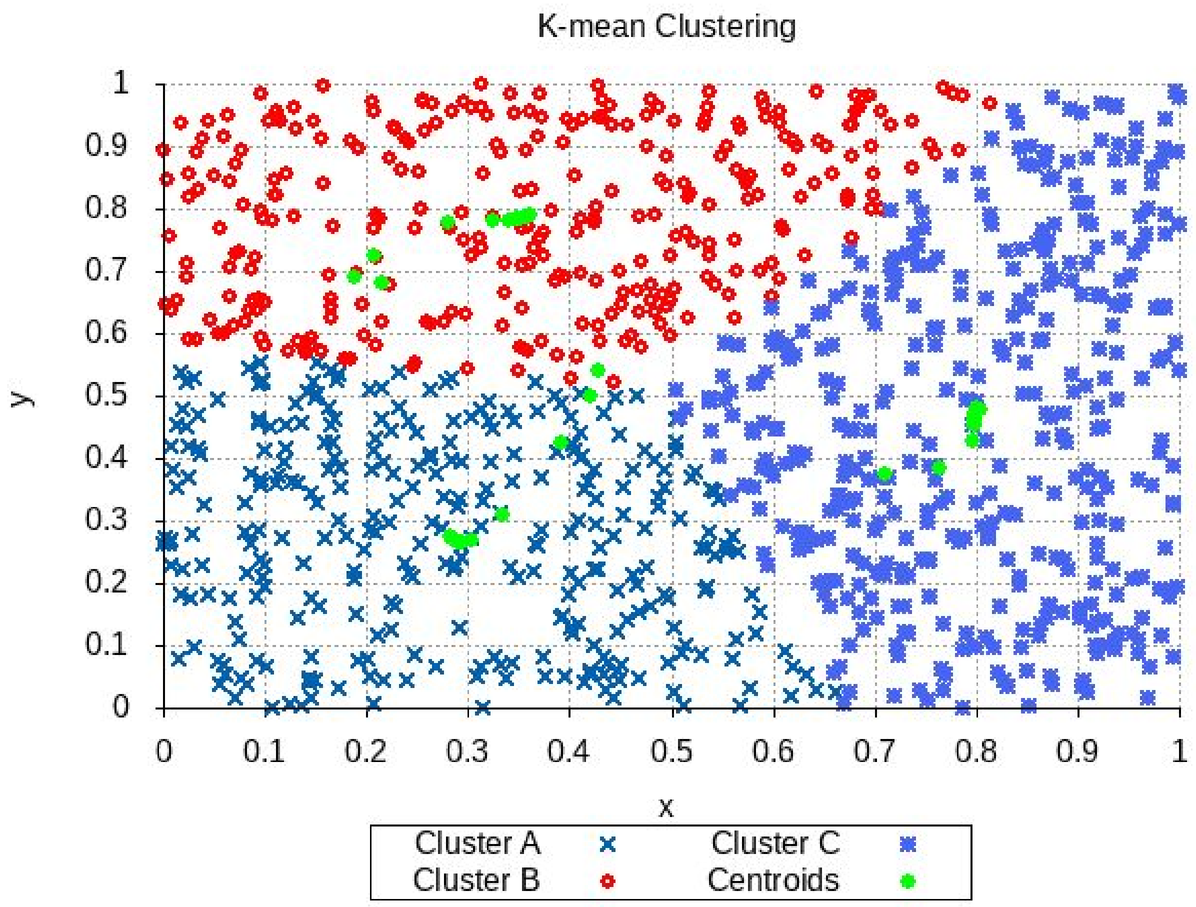
Use this command to create kmean cluster with 3 clusters and 100 data points per process graph:

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

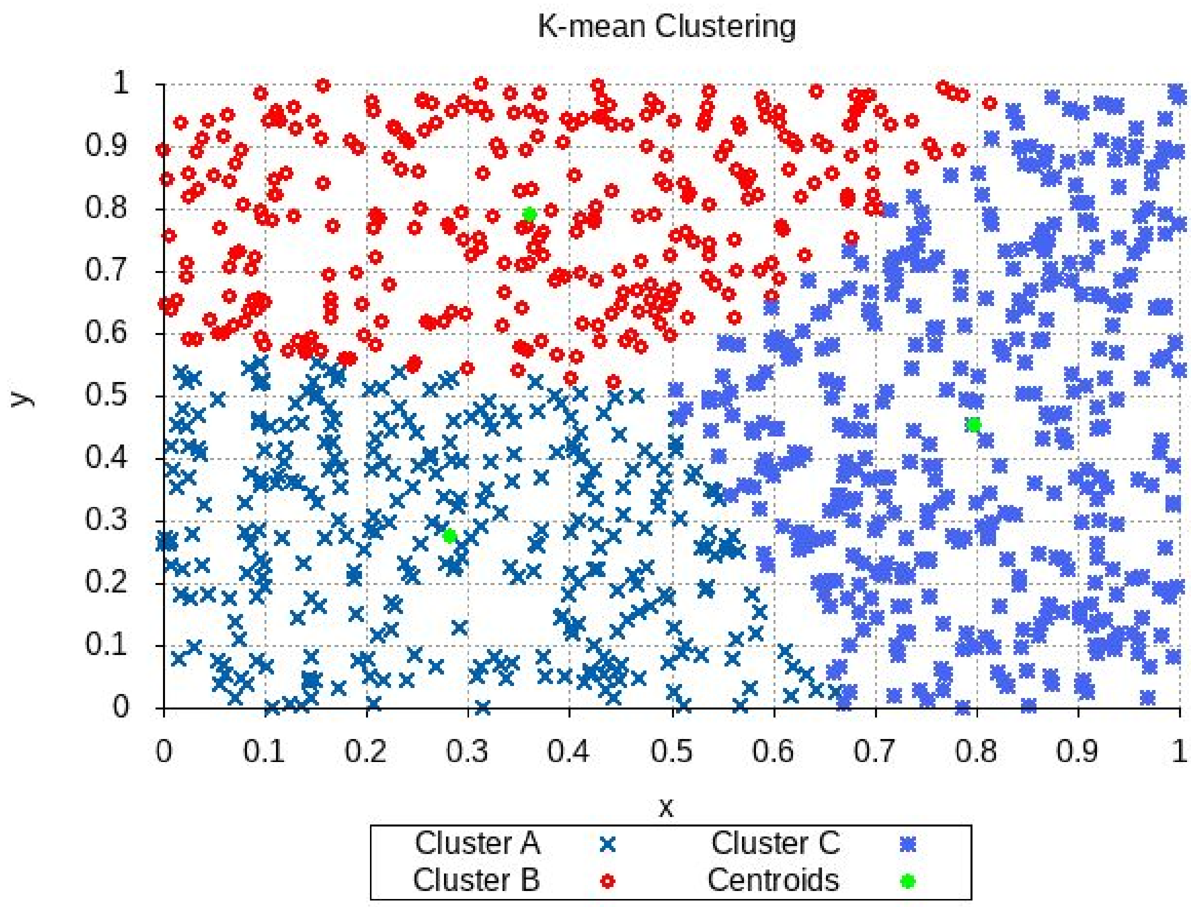
Example: mpicc main.c && mpirun -n 10 a.out 3 2 1000 && gnuplot graphs/3_kmean_graph.gp

The output will look like this:

Changing centroids Process



Final centroid when finish

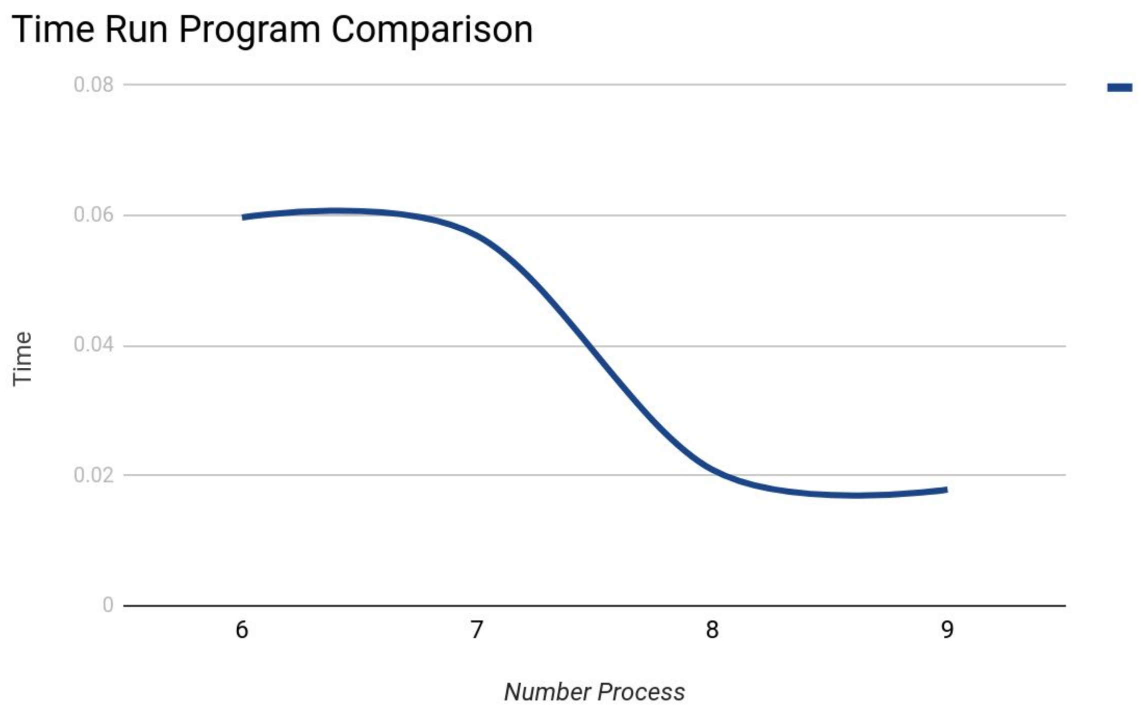


Command to run both method above “make”

V. Result From Run Program

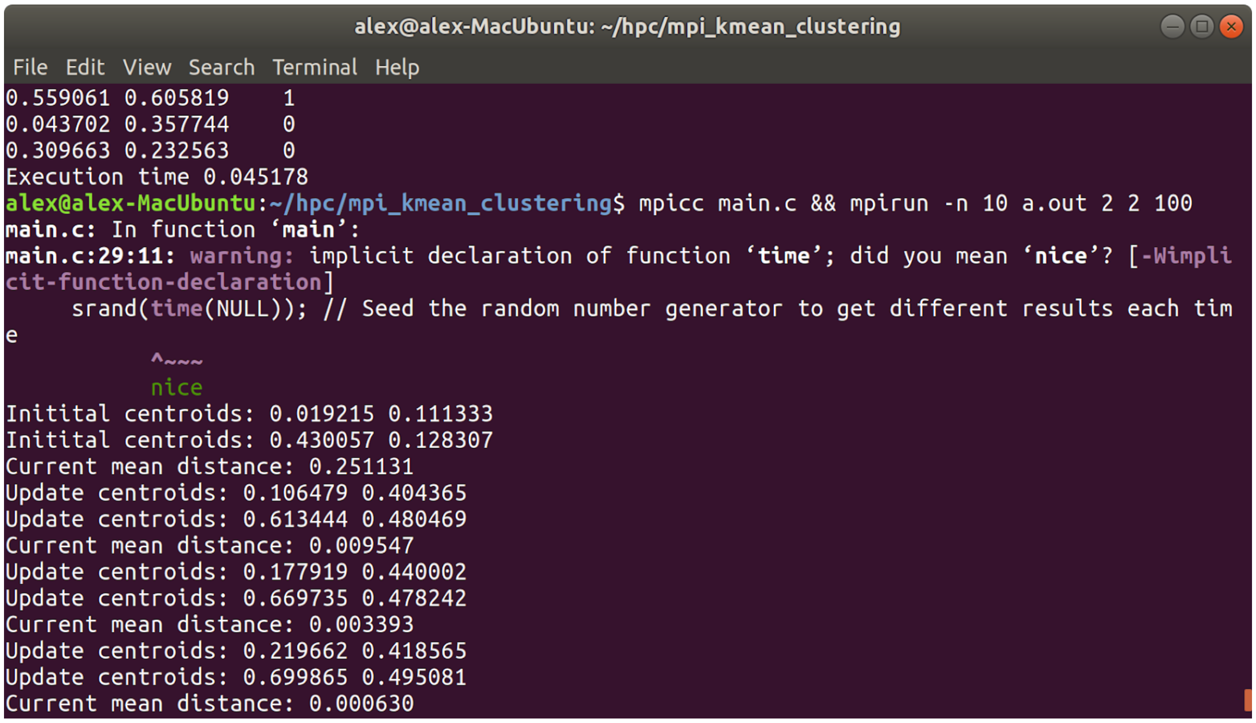
Table time run program with k number is 2, dimension is 2 and total point 1000 but N number process increase

N	6	7	8	9
Time	0.059706	0.056879	0.020904	0.017869

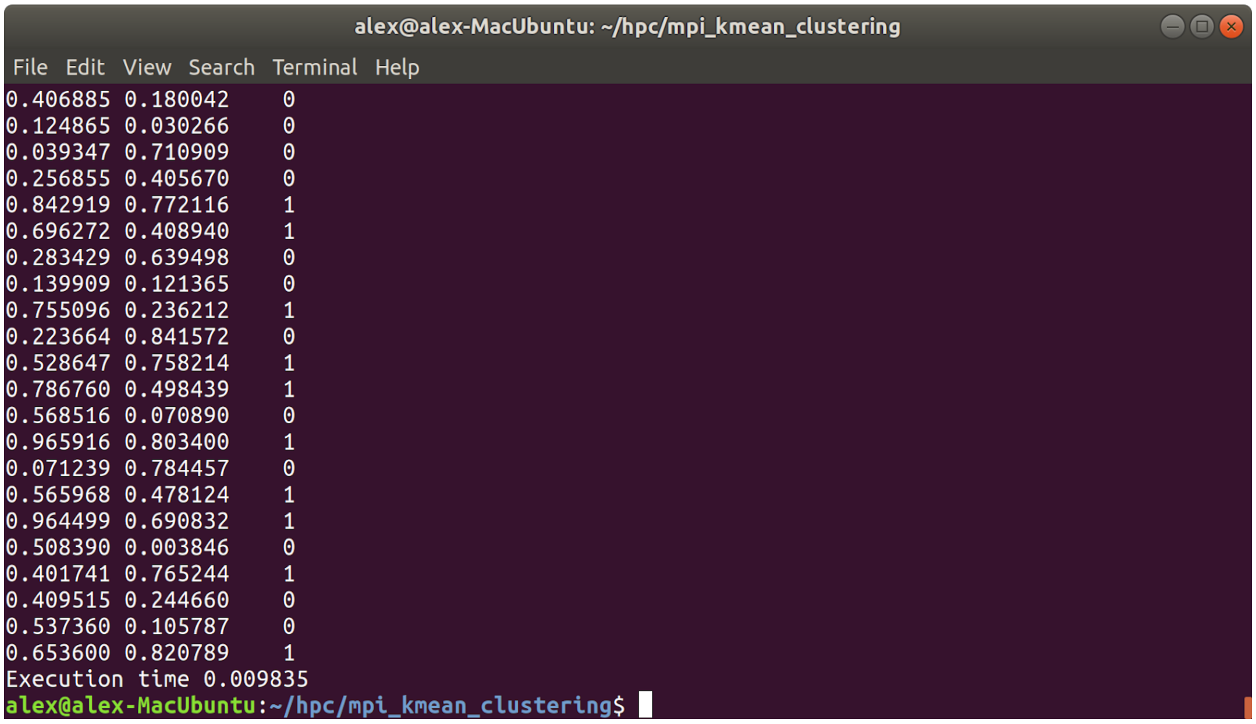


Conclusion: The more process use to clustering, the faster program will be

Run sample screenshots:



```
0.559061 0.605819     1
0.043702 0.357744     0
0.309663 0.232563     0
Execution time 0.045178
alex@alex-MacUbuntu:~/hpc/mpi_kmean_clustering$ mpicc main.c && mpirun -n 10 a.out 2 2 100
main.c: In function ‘main’:
main.c:29:11: warning: implicit declaration of function ‘time’; did you mean ‘nice’? [-Wimpli
cit-function-declaration]
     srand(time(NULL)); // Seed the random number generator to get different results each tim
e
           ^~~~
           nice
Initital centroids: 0.019215 0.111333
Initital centroids: 0.430057 0.128307
Current mean distance: 0.251131
Update centroids: 0.106479 0.404365
Update centroids: 0.613444 0.480469
Current mean distance: 0.009547
Update centroids: 0.177919 0.440002
Update centroids: 0.669735 0.478242
Current mean distance: 0.003393
Update centroids: 0.219662 0.418565
Update centroids: 0.699865 0.495081
Current mean distance: 0.000630
```



```
0.406885 0.180042     0
0.124865 0.030266     0
0.039347 0.710909     0
0.256855 0.405670     0
0.842919 0.772116     1
0.696272 0.408940     1
0.283429 0.639498     0
0.139909 0.121365     0
0.755096 0.236212     1
0.223664 0.841572     0
0.528647 0.758214     1
0.786760 0.498439     1
0.568516 0.070890     0
0.965916 0.803400     1
0.071239 0.784457     0
0.565968 0.478124     1
0.964499 0.690832     1
0.508390 0.003846     0
0.401741 0.765244     1
0.409515 0.244660     0
0.537360 0.105787     0
0.653600 0.820789     1
Execution time 0.009835
alex@alex-MacUbuntu:~/hpc/mpi_kmean_clustering$
```