#include<stdio.h>

#include<stdlib.h>

#include<time.h>

#include<unistd.h>

#include<math.h>

#include<errno.h>

#include<mpi.h>

#define MAX\_ITERATIONS 100

int numOfClusters = 0;

int numOfElements = 0;

int num\_of\_processes = 0;

/\* This function goes through that data points and assigns them to a cluster \*/

void assign2Cluster(double k\_x[], double k\_y[], double recv\_x[], double recv\_y[], int assign[])

{

double min\_dist = 10000000;

double x=0, y=0, temp\_dist=0;

int k\_min\_index = 0;

for(int i = 0; i < (numOfElements/num\_of\_processes) + 1; i++)

{

for(int j = 0; j < numOfClusters; j++)

{

x = abs(recv\_x[i] - k\_x[j]);

y = abs(recv\_y[i] - k\_y[j]);

temp\_dist = sqrt((x\*x) + (y\*y));

// new minimum distance found

if(temp\_dist < min\_dist)

{

min\_dist = temp\_dist;

k\_min\_index = j;

}

}

// update the cluster assignment of this data points

assign[i] = k\_min\_index;

}

}

/\* Recalcuate k-means of each cluster because each data point may have

been reassigned to a new cluster for each iteration of the algorithm \*/

void calcKmeans(double k\_means\_x[], double k\_means\_y[], double data\_x\_points[], double data\_y\_points[], int k\_assignment[])

{

double total\_x = 0;

double total\_y = 0;

int numOfpoints = 0;

for(int i = 0; i < numOfClusters; i++)

{

total\_x = 0;

total\_y = 0;

numOfpoints = 0;

for(int j = 0; j < numOfElements; j++)

{

if(k\_assignment[j] == i)

{

total\_x += data\_x\_points[j];

total\_y += data\_y\_points[j];

numOfpoints++;

}

}

if(numOfpoints != 0)

{

k\_means\_x[i] = total\_x / numOfpoints;

k\_means\_y[i] = total\_y / numOfpoints;

}

}

}

int main(int argc, char \*argv[])

{

// initialize the MPI environment

MPI\_Init(NULL, NULL);

// get number of processes

int world\_size;

MPI\_Comm\_size(MPI\_COMM\_WORLD, &world\_size);

// get rank

int world\_rank;

MPI\_Comm\_rank(MPI\_COMM\_WORLD, &world\_rank);

// send buffers

double \*k\_means\_x = NULL; // k means corresponding x values

double \*k\_means\_y = NULL; // k means corresponding y values

int \*k\_assignment = NULL; // each data point is assigned to a cluster

double \*data\_x\_points = NULL;

double \*data\_y\_points = NULL;

// receive buffer

double \*recv\_x = NULL;

double \*recv\_y = NULL;

int \*recv\_assign = NULL;

if(world\_rank == 0)

{

if(argc != 2)

{

printf("Please include an argument after the program name to list how many processes.\n");

printf("e.g. To indicate 4 processes, run: mpirun -n 4 ./kmeans 4\n");

exit(-1);

}

num\_of\_processes = atoi(argv[1]);

char buffer[2];

printf("\nIMPLEMENTING PARALLEL K-MEANS CLUSTERING ALGORITHM\n\nHow many clusters would you like to analyze for? ");

scanf("%s", buffer);

printf("\n");

numOfClusters = atoi(buffer);

printf("Ok %d clusters it is.\n", numOfClusters);

// broadcast the number of clusters to all nodes

MPI\_Bcast(&numOfClusters, 1, MPI\_INT, 0, MPI\_COMM\_WORLD);

// allocate memory for arrays

k\_means\_x = (double \*)malloc(sizeof(double) \* numOfClusters);

k\_means\_y = (double \*)malloc(sizeof(double) \* numOfClusters);

if(k\_means\_x == NULL || k\_means\_y == NULL)

{

perror("malloc");

exit(-1);

}

printf("Reading input data from file...\n\n");

FILE\* fp = fopen("input.txt", "r");

if(!fp)

{

perror("fopen");

exit(-1);

}

// count number of lines to find out how many elements

int c = 0;

numOfElements = 0;

while(!feof(fp))

{

c = fgetc(fp);

if(c == '\n')

{

numOfElements++;

}

}

printf("There are a total number of %d elements in the file.\n", numOfElements);

// broadcast the number of elements to all nodes

MPI\_Bcast(&numOfElements, 1, MPI\_INT, 0, MPI\_COMM\_WORLD);

// allocate memory for an array of data points

data\_x\_points = (double \*)malloc(sizeof(double) \* numOfElements);

data\_y\_points = (double \*)malloc(sizeof(double) \* numOfElements);

k\_assignment = (int \*)malloc(sizeof(int) \* numOfElements);

if(data\_x\_points == NULL || data\_y\_points == NULL || k\_assignment == NULL)

{

perror("malloc");

exit(-1);

}

// reset file pointer to origin of file

fseek(fp, 0, SEEK\_SET);

// now read in points and fill the arrays

int i = 0;

double point\_x=0, point\_y=0;

while(fscanf(fp, "%lf %lf", &point\_x, &point\_y) != EOF)

{

data\_x\_points[i] = point\_x;

data\_y\_points[i] = point\_y;

// assign the initial k means to zero

k\_assignment[i] = 0;

i++;

}

// close file pointer

fclose(fp);

// randomly select initial k-means

time\_t t;

srand((unsigned) time(&t));

int random;

for(int i = 0; i < numOfClusters; i++) {

random = rand() % numOfElements;

k\_means\_x[i] = data\_x\_points[random];

k\_means\_y[i] = data\_y\_points[random];

}

printf("Running k-means algorithm for %d iterations...\n\n", MAX\_ITERATIONS);

for(int i = 0; i < numOfClusters; i++)

{

printf("Initial K-means: (%f, %f)\n", k\_means\_x[i], k\_means\_y[i]);

}

// allocate memory for receive buffers

recv\_x = (double \*)malloc(sizeof(double) \* ((numOfElements/num\_of\_processes) + 1));

recv\_y = (double \*)malloc(sizeof(double) \* ((numOfElements/num\_of\_processes) + 1));

recv\_assign = (int \*)malloc(sizeof(int) \* ((numOfElements/num\_of\_processes) + 1));

if(recv\_x == NULL || recv\_y == NULL || recv\_assign == NULL)

{

perror("malloc");

exit(-1);

}

}

else

{ // I am a worker node

num\_of\_processes = atoi(argv[1]);

// receive broadcast of number of clusters

MPI\_Bcast(&numOfClusters, 1, MPI\_INT, 0, MPI\_COMM\_WORLD);

// receive broadcast of number of elements

MPI\_Bcast(&numOfElements, 1, MPI\_INT, 0, MPI\_COMM\_WORLD);

// allocate memory for arrays

k\_means\_x = (double \*)malloc(sizeof(double) \* numOfClusters);

k\_means\_y = (double \*)malloc(sizeof(double) \* numOfClusters);

if(k\_means\_x == NULL || k\_means\_y == NULL)

{

perror("malloc");

exit(-1);

}

// allocate memory for receive buffers

recv\_x = (double \*)malloc(sizeof(double) \* ((numOfElements/num\_of\_processes) + 1));

recv\_y = (double \*)malloc(sizeof(double) \* ((numOfElements/num\_of\_processes) + 1));

recv\_assign = (int \*)malloc(sizeof(int) \* ((numOfElements/num\_of\_processes) + 1));

if(recv\_x == NULL || recv\_y == NULL || recv\_assign == NULL)

{

perror("malloc");

exit(-1);

}

}

/\* Distribute the work among all nodes. The data points itself will stay constant and

not change for the duration of the algorithm. \*/

MPI\_Scatter(data\_x\_points, (numOfElements/num\_of\_processes) + 1, MPI\_DOUBLE,

recv\_x, (numOfElements/num\_of\_processes) + 1, MPI\_DOUBLE, 0, MPI\_COMM\_WORLD);

MPI\_Scatter(data\_y\_points, (numOfElements/num\_of\_processes) + 1, MPI\_DOUBLE,

recv\_y, (numOfElements/num\_of\_processes) + 1, MPI\_DOUBLE, 0, MPI\_COMM\_WORLD);

int count = 0;

while(count < MAX\_ITERATIONS)

{

// broadcast k-means arrays

MPI\_Bcast(k\_means\_x, numOfClusters, MPI\_DOUBLE, 0, MPI\_COMM\_WORLD);

MPI\_Bcast(k\_means\_y, numOfClusters, MPI\_DOUBLE, 0, MPI\_COMM\_WORLD);

// scatter k-cluster assignments array

MPI\_Scatter(k\_assignment, (numOfElements/num\_of\_processes) + 1, MPI\_INT,

recv\_assign, (numOfElements/num\_of\_processes) + 1, MPI\_INT, 0, MPI\_COMM\_WORLD);

// assign the data points to a cluster

assign2Cluster(k\_means\_x, k\_means\_y, recv\_x, recv\_y, recv\_assign);

// gather back k-cluster assignments

MPI\_Gather(recv\_assign, (numOfElements/num\_of\_processes)+1, MPI\_INT,

k\_assignment, (numOfElements/num\_of\_processes)+1, MPI\_INT, 0, MPI\_COMM\_WORLD);

// let the root process recalculate k means

if(world\_rank == 0)

{

calcKmeans(k\_means\_x, k\_means\_y, data\_x\_points, data\_y\_points, k\_assignment);

//printf("Finished iteration %d\n",count);

}

count++;

}

Endtime = MPI\_Wtime();

cpu\_time\_used = Endtime – StartTime;

printf("\nTotal Time consumed by processor %d : %f seconds..\n",world\_rank,cpu\_time\_used);

if(world\_rank == 0)

{

printf("--------------------------------------------------\n");

printf("FINAL RESULTS:\n");

for(int i = 0; i < numOfClusters; i++)

{

printf("Cluster #%d: (%f, %f)\n", i, k\_means\_x[i], k\_means\_y[i]);

}

printf("--------------------------------------------------\n");

}

// deallocate memory and clean up

free(k\_means\_x);

free(k\_means\_y);

free(data\_x\_points);

free(data\_y\_points);

free(k\_assignment);

free(recv\_x);

free(recv\_y);

free(recv\_assign);

//MPI\_Barrier(MPI\_COMM\_WORLD);

MPI\_Finalize();

}

--------------------------------Execution--------------------------------

Complilation – mpicc mpi\_kmeans.c –o mpi\_kmeans –lm

Execution – mpirun –n 3 mpi\_kmeans 3

--------------------------------------------------------------------------

-----------------------------------Output---------------------------------

Output for k = 2

IMPLEMENTING PARALLEL K-MEANS CLUSTERING ALGORITHM

2

How many clusters would you like to analyze for?

Ok 2 clusters it is.

Reading input data from file...

There are a total number of 258 elements in the file.

Running k-means algorithm for 100 iterations...

Initial K-means: (24.300000, 27.000000)

Initial K-means: (0.000000, 0.000000)

Total Time consumed by processor 1 : 1.232257 seconds..

Total Time consumed by processor 2 : 1.232275 seconds..

Total Time consumed by processor 0 : 1.232797 seconds..

--------------------------------------------------

FINAL RESULTS:

Cluster #0: (31.275177, 30.846099)

Cluster #1: (3.820513, 3.585470)

--------------------------------------------------

Output for k = 3

IMPLEMENTING PARALLEL K-MEANS CLUSTERING ALGORITHM

3

How many clusters would you like to analyze for?

Ok 3 clusters it is.

Reading input data from file...

There are a total number of 258 elements in the file.

Running k-means algorithm for 100 iterations...

Initial K-means: (0.000000, 0.000000)

Initial K-means: (0.000000, 0.000000)

Initial K-means: (44.000000, 36.500000)

Total Time consumed by processor 1 : 1.385946 seconds..

Total Time consumed by processor 2 : 1.385940 seconds..

Total Time consumed by processor 0 : 1.385979 seconds..

--------------------------------------------------

FINAL RESULTS:

Cluster #0: (26.608108, 25.598649)

Cluster #1: (4.252101, 3.987395)

Cluster #2: (36.643077, 36.923077)

--------------------------------------------------

Output for k = 4

IMPLEMENTING PARALLEL K-MEANS CLUSTERING ALGORITHM

4

How many clusters would you like to analyze for?

Ok 4 clusters it is.

Reading input data from file...

There are a total number of 258 elements in the file.

Running k-means algorithm for 100 iterations...

Initial K-means: (0.000000, 0.000000)

Initial K-means: (25.800000, 29.000000)

Initial K-means: (0.000000, 0.000000)

Initial K-means: (0.000000, 0.000000)

Total Time consumed by processor 1 : 1.414641 seconds..

Total Time consumed by processor 2 : 1.414590 seconds..

Total Time consumed by processor 0 : 1.414597 seconds..

--------------------------------------------------

FINAL RESULTS:

Cluster #0: (36.261765, 36.573529)

Cluster #1: (12.833333, 17.333333)

Cluster #2: (67.896552, 65.320690)

Cluster #3: (2.225806, 1.829032)

--------------------------------------------------

Output for k = 5

IMPLEMENTING PARALLEL K-MEANS CLUSTERING ALGORITHM

5

How many clusters would you like to analyze for?

Ok 5 clusters it is.

Reading input data from file...

There are a total number of 258 elements in the file.

Running k-means algorithm for 100 iterations...

Initial K-means: (8.000000, 6.000000)

Initial K-means: (0.000000, 0.000000)

Initial K-means: (54.000000, 50.000000)

Initial K-means: (0.000000, 0.000000)

Initial K-means: (38.000000, 33.000000)

Total Time consumed by processor 1 : 1.037083 seconds..

Total Time consumed by processor 2 : 1.037089 seconds..

Total Time consumed by processor 0 : 1.037144 seconds..

--------------------------------------------------

FINAL RESULTS:

Cluster #0: (67.896552, 65.320690)

Cluster #1: (67.896552, 65.320690)

Cluster #2: (29.758621, 29.106897)

Cluster #3: (0.310345, 0.275862)

Cluster #4: (26.676190, 26.339286)

--------------------------------------------------