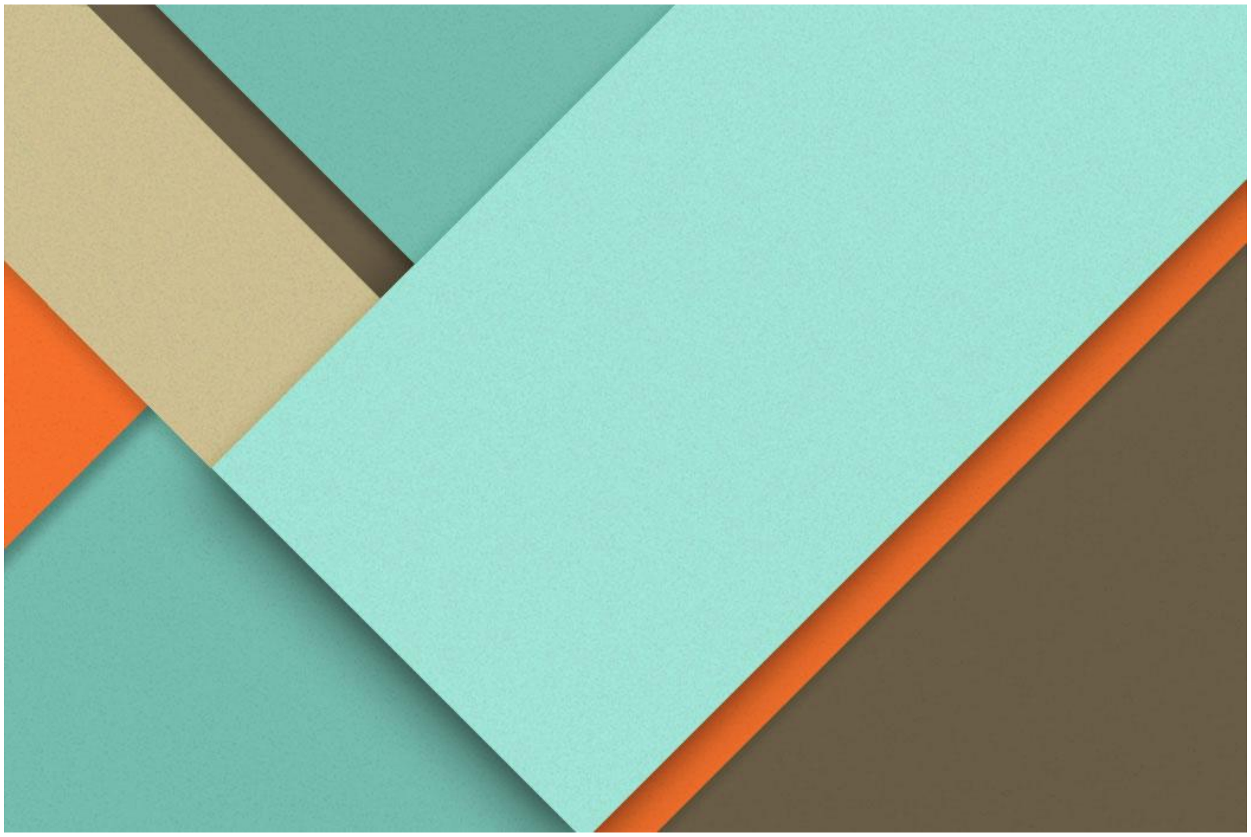


Assignment Report

CSE355 (UG2018) - Parallel and Distributed Algorithms (27224)

Mohamed Mokhtar
18P7232

Overview

K-means clustering is a widely used algorithm for clustering data into groups of similar items. It works by iteratively assigning each data point to the closest group (called a cluster) and then re-computing the centroid of each cluster as the mean of all the data points assigned to it.

One way to parallelize the k-means algorithm is to divide the data points among the available processors and have each processor compute the centroids for its assigned data points. The centroids can then be combined and the final centroids can be computed. This can be done using the Message Passing Interface (MPI) library, which provides a set of functions for passing messages between processors.

Here is a high-level outline of the steps involved in parallelizing k-means using MPI:

1. Initialize MPI and get the number of processors and the rank of the current processor.
2. Divide the data points among the processors using interleaved assignment
3. Broadcast the centroids to all processors using MPI_Bcast.
4. On each processor, compute the centroids for the assigned data points.
5. Gather the partial centroids (and the number of assigned points for each centroid) computed by each processor using MPI_Reduce.
6. On the master processor, compute the final centroids by weight-averaging the partial centroids.
7. Repeat steps 4-6 for the given number of iterations.

Code

```
#include <stdio.h>
#include <stdlib.h>
#include <mpi.h>

#define CLUSTER_COUNT   2
#define ITERATION_COUNT 10000

#define PRINT_ITERATIONS 0

int size, rank;

struct point {
 double x, y;
};

void usage(const char *err)
{
   if(rank == 0)
   {
       printf("%s\nUsage: ./kmeans 1,3 4,5 -2,4.43 42.3,2\n", err);
   }

   MPI_Finalize();

   exit(0);
}

int kmeans(struct point *points, size_t point_count)
{
   // Pick cluster points
   double l_kx[CLUSTER_COUNT], kx[CLUSTER_COUNT];          // X Pos for each
cluster
   double l_ky[CLUSTER_COUNT], ky[CLUSTER_COUNT];          // Y Pos for each
cluster

   double l_kcount[CLUSTER_COUNT], kcount[CLUSTER_COUNT];  // Number of points
contained in each cluster

   // Pick the cluster points according to the given points (at first)
   // NOTE: It would be better to make sure no points start at the same
location
```

```
    for(int i = 0; i < CLUSTER_COUNT; i++) {
        kx[i] = points[i].x;
        ky[i] = points[i].y;
    }

    // Print the point info (rank only)
    if(rank == 0)
    {
        printf("\n");

        for (size_t i = 0; i < point_count; i++)
        {
            printf("Point %ld: %lf,%lf\n", i+1, points[i].x, points[i].y);
        }
    }

    for (size_t i = 0; i < ITERATION_COUNT; i++)
    {
#if PRINT_ITERATIONS
        if(rank == 0)
        {
            printf("\n");
            for (size_t i = 0; i < CLUSTER_COUNT; i++)
                printf("%lf,%lf    ", i+1, kx[i], ky[i]);
        }
#endif

        // Send the current cluster positions
        MPI_Bcast(&kx, CLUSTER_COUNT, MPI_DOUBLE, 0, MPI_COMM_WORLD);
        MPI_Bcast(&ky, CLUSTER_COUNT, MPI_DOUBLE, 0, MPI_COMM_WORLD);

        // Using interleaved assignment because it's easier to get your head
around
        for (size_t j = rank; j < point_count; j += size)
        {
            // Check which cluster this point belongs to
            size_t min_index = 0;
            size_t min_dist = -1;
            for (size_t k = 0; k < CLUSTER_COUNT; k++)
            {
                size_t current_dist = (kx[k] - points[j].x)*(kx[k] -
points[j].x) + (ky[k] - points[j].y)*(ky[k] - points[j].y);
                if(current_dist < min_dist)
                {
                    min_index = k;
                    min_dist = current_dist;
                }
            }

            // Update cluster values
            l_kx[min_index] += points[j].x;
            l_ky[min_index] += points[j].y;
            l_kcount[min_index]++;
        }

        // Send the sums to root
        MPI_Reduce(&l_kx, &kx, CLUSTER_COUNT, MPI_DOUBLE, MPI_SUM, 0,
 MPI_COMM_WORLD);
        MPI_Reduce(&l_ky, &ky, CLUSTER_COUNT, MPI_DOUBLE, MPI_SUM, 0,
 MPI_COMM_WORLD);
        MPI_Reduce(&l_kcount, &kcount, CLUSTER_COUNT, MPI_DOUBLE, MPI_SUM, 0,
 MPI_COMM_WORLD);

        // Calculate averages from the sums
        for (size_t j = 0; j < CLUSTER_COUNT; j++)
        {
            kx[j] /= kcount[j];
            ky[j] /= kcount[j];
        }
    }

    if(rank == 0)
    {
        printf("\n\n");
        for (size_t i = 0; i < CLUSTER_COUNT; i++)
            printf("Cluster %ld: %lf,%lf\n", i+1, kx[i], ky[i]);
    }
 }

 int main(int argc, char** argv)
 {
    MPI_Init(&argc, &argv);

    // Initialisation
    MPI_Comm_size(MPI_COMM_WORLD, &size);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);

    if(argc == 1)
```

```
        usage("\nPlease provide the points as arguments...\n");

    // Parse Args
    size_t point_count = argc-1;
    struct point *points = malloc(sizeof(struct point) * point_count);

    for (size_t i = 0; i < point_count; i++)
        if(sscanf(argv[1+i], "%lf,%lf", &points[i].x, &points[i].y) != 2)
            usage("\nAn argument is invalid...\n");

    kmeans(points, point_count);

    // Fin...
    MPI_Finalize();

    return 0;
}
```

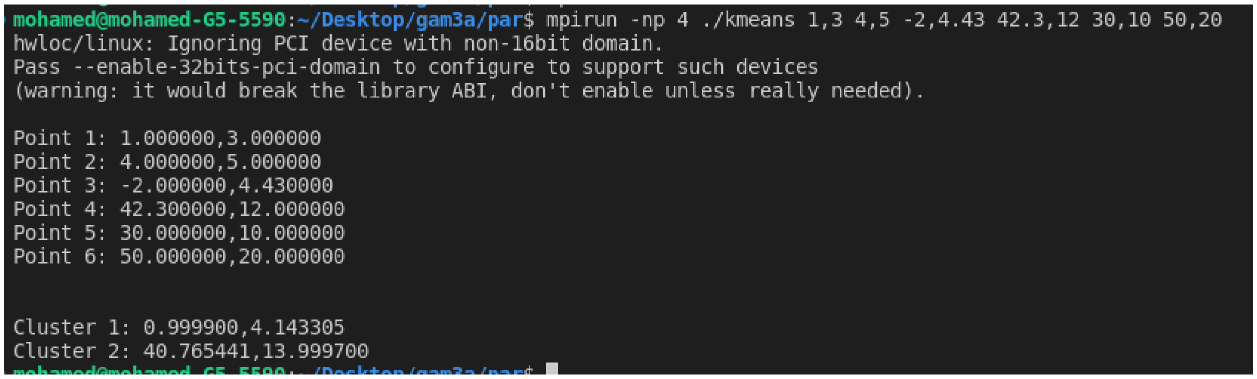
Example Run

Two clusters, five points, four processes

```
$ mpirun -np 4 ./kmeans 1,3 4,5 -2,4.43 42.3,12 30,10 50,20
hwloc/linux: Ignoring PCI device with non-16bit domain.
Pass --enable-32bits-pci-domain to configure to support such devices
(warning: it would break the library ABI, don't enable unless really
needed).

Point 1: 1.000000,3.000000
Point 2: 4.000000,5.000000
Point 3: -2.000000,4.430000
Point 4: 42.300000,12.000000
Point 5: 30.000000,10.000000
Point 6: 50.000000,20.000000


Cluster 1: 0.999900,4.143305
Cluster 2: 40.765441,13.999700
```



```
mohamed@mohamed-G5-5590:~/Desktop/gam3a/par$ mpirun -np 4 ./kmeans 1,3 4,5 -2,4.43 42.3,12 30,10 50,20
hwloc/linux: Ignoring PCI device with non-16bit domain.
Pass --enable-32bits-pci-domain to configure to support such devices
(warning: it would break the library ABI, don't enable unless really needed).

Point 1: 1.000000,3.000000
Point 2: 4.000000,5.000000
Point 3: -2.000000,4.430000
Point 4: 42.300000,12.000000
Point 5: 30.000000,10.000000
Point 6: 50.000000,20.000000


Cluster 1: 0.999900,4.143305
Cluster 2: 40.765441,13.999700
mohamed@mohamed-G5-5590:~/Desktop/gam3a/par$
```