Parallel implementation of K-Means

K-Means Background:

*k*-means clustering is a method of [vector quantization](https://en.wikipedia.org/wiki/Vector_quantization), originally from [signal processing](https://en.wikipedia.org/wiki/Signal_processing), that is popular for [cluster analysis](https://en.wikipedia.org/wiki/Cluster_analysis) in [data mining](https://en.wikipedia.org/wiki/Data_mining).

K-Means is one of the most popular "clustering" algorithms. K-means stores k centroids that it uses to define clusters. A point is considered to be in a particular cluster if it is closer to that cluster's centroid than any other centroid.

The Problem:

Large datasets of points (two dimensions), each includes velocity that change their location every pulse. the goal is to find the best distribution of n points in k clusters by measure the quality (an average of diameters of the cluster divided by distance to other clusters).

Input data:

* **N** - number of points
* **K** - number of clusters to find
* **LIMIT** – the maximum number of iterations for K-MEAN algorithm.
* **QM** – quality measure to stop
* **T** – defines the end of time interval [0, T]
* **dT** – defines moments t = n\*dT, n = { 0, 1, 2, … , T/dT} for which calculate the clusters and the quality
* Coordinates and Velocities of all points

Solution:

The basic of the algorithm is double loop – the outer loop pulse the dT (move the points by their velocity), and the inner loop clustering “LIMIT” times the points (LIMITS times OR if there is no change in the points per cluster).

After we get out from the inner loop, we check are quality – if the quality is under or equal to given QM, we finish the algorithm and we need to write to file the quality and the clusters centers. If the quality is above given QM, we continue to the outer loop, move the points by their velocity and get the inner loop again.

Make the K-Means algorithm parallel instead of serial, thanks to calculates that not dependent each other’s.

The technologies to do so are:

1. MPI - Process Message Passing Interface.
2. Open MP - Open Multi-Processing - multi-threaded, shared memory parallelism.
3. CUDA - GPU Programming API created by Nvidia.

In my solution, I decided that every available process make calculates on his using Open MP and Cuda methods. The master proc handle the reminder.

All the data managed by collecting (gather) and broadcasting by one process with MPI.

Pseudo Code:

If ( Myid == 0 )

readFromFile();

MPI\_Bcast(info from file);

for ( double i = 0; i < T; i += dT ) //the outer loop

{

If ( its not the first iteration )

{

cudaRefreshPoints (every proc send his part of points);

master handle the reminder(); // iterative function

MPI\_Gather(gather the points from proc);

MPI\_Bcast(broadcast the points to proc);

}

newCicleClusterArr();//init the clusters

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

{

cudaOrganizePoints (every proc send his part of points and flag if need to stop);

master handle the reminder(); // iterative function

MPI\_Gather(gather the points from proc);

refreshClusterByPoints();//master refresh the cluster by points

MPI\_Gather(gather the flags from proc);

Master check the flags();

MPI\_Bcast(clusters);

MPI\_Bcast(final flag if need to stop);

MPI\_Bcast(points);

recalculateClusterCenter();

check flag(break if true);

}

DiameterOMP(every proc send his part of points);

MPI\_Gather(clusters);

MPI\_Bcast(clusters);

Q = calculateQuality();

If ( Q <= QM )

{

writeToFile();

break();

}

}

Implementation:

Phase 1 – read from file

All the data store in text file. In this phase the master proc read the data from the file.

Phase 2 – prepare for k-means algorithm, send the points and data

Using MPI:

1. The master proc broadcasting the information (N, K, T, dT, LIMIT, QM) to the other proc.
2. The master proc broadcasting all the points to the other proc.

All the proc make cluster array and initialize each array with initClusterArr() Open MP method.

Create reminder variable (N % number of proc ) to know if reminder handle is needed.

Phase 3 – dynamic k-means algorithm

Phase 3.1 – refresh the points by their velocity (except the first iteration)

* Every proc send his part of points to cuda method cudaRefreshPoints() that refresh the points by their velocities.
* The master proc handle the reminder – if there is reminder the master proc send only the reminder last part of the points array.
* The master proc gather all the parts of the points array from other proc.
* The master proc broadcast the points

In this point every proc have updated array of points.

Phase 3.2 – organize points (clustering the points by the closest clusters)

* Every proc send his part of points to cuda method cudaOrganizePoints() that clustering every points to the closest cluster, and change the boolean flag if there is no movement at all (every point stay in the last cluster).
* The master proc handle the reminder – if there is reminder the master proc send only the reminder last part of the points array.
* The master proc gather all the parts of the points array from other proc.
* The master update the cluster array by the updated points array with Open MP method refreshClusterByPoints().
* The master proc gather all the booleans flags.
* The master proc check if there is no change of points in all proc (by the Boolean flags).
* Master proc broadcasting the updated cluster array.
* Master proc broadcasting the flag if there was points movement or not (if break from the inner loop or continue).
* Master proc broadcasting the updated points array.
* All proc calculate the new center of each cluster with Open MP method recalculateClusterCenter().
* All proc check the flag to know if break or continue.

Phase 3.3 – calculate clusters diameter

* Every proc using Open MP method diameterOMP() to calculate his part. The method calculate the updated diameter of each cluster. The partition here for every proc is different then because here the partition is double loop that every outer loop variable is passing all the points from him and further on. So if the partition was the first proc was work more than the last proc. I get to this for the best partition:

int part = N / sumOfNumOfProc;

int\* startPointsForEachProcDiameter = (int\*)malloc(numprocs \* sizeof(int));

startPointsForEachProcDiameter[0] = 0;  
 for (int i = 1; i <= numprocs; i++)

startPointsForEachProcDiameter[i]=startPointsForEachProcDiameter[i-1]+i\*part;

* Master proc gather the clusters parts from the other proc.
* Master proc broadcast the updated cluster array.

Phase 3.4 – calculate quality

* Using Open MP method quality(), every proc calculate the quality and check if the quality is under the given QM.

Complexity:

(n- number of points, k- number of clusters, p- number of proc, tp- number of threads in each proc, tc- number of threads in each cuda, cuda penalty- the time to send to cuda, malloc and memcopy, mpi penalty- the time to send with mpi)

1. Phase 1 – reading from file N points iteratively with one proc. O(n)
2. Phase 2 – the heaviest operation in this phase is broadcast all the points. O(n+mpi penelty)
3. Phase 3.1 – the heaviest operation is cudaRefreshPoints(). O(+cuda penelty)
4. Phase 3.2- the heaviest operation is refreshClusterByPoints() O(+K\*tp)
5. Phase 3.3- the heaviest operation is diameterOMP(). in average O()
6. Phase 3.4- quality() complexity. O()

Phase 3.1 is in the outer loop – T times.

Phases 3.2,3.3 is in the inner loop- LIMIT times.

* O{[T\*O(+cuda penelty)]\*[LIMIT\*O(+K\*tp)+O()]} O(T\*\*LIMIT\*)