**Parallel K-Means Algorithm Documentation**

Danielle Kahana, 204572846

**Steps of the algorithm:**

1. Process 0 reads the file input and initiate the first K clusters with the first K points using OMP.

2. Process 0 broadcasting using MPI\_Bcast to other processes the file input information -> N, K, T, dT, LIMIT, QM.

3. Process 0 broadcasting using MPI\_Bcast to other processes the K-clusters array with the initiate clusters.

4. The N points are divided by the number of processes, in case there is a remainder process 0 will get its share plus the remainder.

Process 0 sends to other processes using MPI\_Send their share. The other processes are receiving using MPI\_Recv.

**From this point time is running from 0 to T. each process is responsible for its points array.**

5. The position of every point is calculated by time using CUDA.

No point depends on another one so a thread is assign to every point.

6. The following steps are done in case the limit of iterations is not reached and there is at least one point that moved from one cluster to another.

6.1. Using CUDA, the distance between every point and every cluster is calculated and the minimum distance found is the "winner". a flag is updated in case one point has moved to another cluster.

here again, no point depends on another one so a thread is assign to every point.

6.2 Updating the clusters number of points and the sum of all the points x,y,z.

6.3 Using MPI\_Allreduce each process is calculating the sum of points for each cluster, so as the number of points. That way, there is no need for sending messages between processes and every process is now able to calculate the new clusters.

7. Each process is sending to process 0 using MPI\_Send its points after each one was classified to a cluster.

Process 0 receives ,using MPI\_Recv , from other process and gather them all to one array of points.

8. Process 0 is evaluating the quality of the results

8.1. The diameter is set for each cluster.

8.2. The quality is calculated by the distance between every clusters using OMP.

9. If the quality is achieved, process 0 broadcasting to other processes the algorithm is finished, otherwise the algorithm continues.

**Complexity:**

This implementation's complexity depends on its heaviest function which is **evaluating the quality**.

first, we calculate the diameter for every cluster:

* for sorting the points array I used quick sort =
* finding for each cluster the diameter requires 3 loops =
* total complexity =

calculating the distance between every 2 clusters and adding to the quality using OMP requires 2 loops = (**Threads** = number of threads the computer can provide)

total complexity =

Evaluating the quality is called in for loop [0,T] until the quality is reached or the loop is over =