**Parallel implementation of K-Means**

**Ran Weiner**

**203701289**

Final project

Course 10324, Parallel and Distributed Computation

2018 SUMMER Semester

# Problem Definition

Given a set of points of **N** points where each point is a **3**-dimensional vector , with initial position (x0,y0,z0) and velocity (Vx,Vy,Vz). The objective is to partition the N points into **K** clusters in which each point belongs to the nearest cluster.

The **K** clusters should be found during give time interval **[0,T]** when the system of **K** clusters has Quality measure **q** that is less than the given value **QM**.

The data set contains at least **10000** but not more than **3000000** points.

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

In my parallel k means implementation I used MPI , OPEN MP and CUDA in order to get maximum speed up that can be achieved.

* The input data set is stored in a text file in one machine, so only one process (the Master process) handles the reading.
* The master chooses the first K points as the clusters centers.
* The master divide the dataset and send each process **N/num of processes** points
* The master handles himself **N/num of processes** points + the reminder N **% num of processes** points**.**
* The master sends the input data, the K clusters and the data points to all other processes.
* All processes have their set of points and know all the necessary information to start the algorithm.
* All processes begin one iteration of K-Means algorithm.
  + the k means iteration continues until there is no point that moved to a different cluster (at start no point assigned to cluster) or until number of iteration as reached to maximum – LIMIT
  + each process assign points to the nearest cluster based on the Euclidean distance formula, if a point changed cluster flag is up.
  + in order to know whether a point is moved in some process I used MPIAllReduce to sum up all the flags from each process in order to detect point movement (flag >0).
  + all processes calculates the new clusters centers based on the average of all points in the clusters with MPIAllReduce
* after the K means iteration the master collect all the points in order to calculate the quality of the clusters
* The master calculates the quality, and let the other processes know if quality achieved or not (less then QM)
* In case quality achieved – we stop the k means and the master write the result to the output text file.
* In case quality is not achieved – we go back to the k means iteration but know with different point's position – updated by the time.

# Implementation

**Step 1 – Initialization:**

Creating the necessary MPI types we are going to use - MPI\_Input\_type, MPI\_Position\_type, MPI\_Velocity\_type, MPI\_Cluster\_type, MPI\_Point\_type.

-Master read all the data from the input file text and initializes array that store all the points and also all the necessary data – N,K,T,dT,LIMIT and QM.

-Master initializes the clusters array with the firs t k points as centers- using OpenMP.

**Step 2 – Share the data:**

-Master broadcast with MPI\_Bcast all the other processes the clusters and all the necessary data – N,K,T,dT,LIMIT and QM.

**Step 3- Divide data between all processes:**

-The master divides the points between all processes so that each process including himself will get the same amount of points – NumPoints/ NumProcs. if there is a reminder the Master will handle it. the points is send to each process with MPI\_Send

- All processes receive the points from Master with MPI\_Recv

**Step 4 – Update points position and start iterations**

-All process performing the same amount of iterations which is T/dT.

- update the points position using Cuda. each point change position by the time = n\*dT .

**Step 5 – K means iterations:**

-each process performs the same iterations which is until the number of iterations exceed to LIMIT or until no points moved in all processes.

**5.1. Assign points to clusters using Cuda:**

- Each process determine the closest cluster for each point using Cuda. This function also give us the information if a point changed cluster – using variable flag.

- With summing all the flags with MPIAllReduce we can check if there is a movement of some point in all the other process.

**5.2. Update local means data:**

- in this step all the points are assigned to a cluster. so with OpenMP we iterate through all points and update the right cluster data – num of points, sum\_x , sum\_y, sum\_z which are cluster attributes that helping us to calculate the new centers later. updating the data implemented with shared cluster array so each thread has its own cluster array – because of a race condition. and afterwards we merge the updated array into the original array.

**5.3. Calculate global means:**

- After all clusters data is updated, we calculate the means new centers. We iterate over the clusters and with Sum operation on MPIAllReduce we can get the number of points, the sum\_x , sum\_y,sum\_z positions of all points assigned to each cluster.

-calculate the centers by dividing the summing of all the positions with the num of points.

-each process knows the new centers.

**Step 6 – Gather all points**

-in order to evaluate the quality later on, the master gathers all the points to one array.

-Master copy his points to the array

-each process sends his points to the master

-Master receives the all the processes points and add to the array.

**Step 7- evaluate quality**

**-**only the Master process calculates the quality because he has the array of all points we needed. at the end we check if the quality achieved which means less then QM.

**7.1.** **Calculate clusters diameter**

-with openMP we calculate each cluster diameter. like the method described in **step 5.2** , every thread responsible for his array of clusters. We calculate the max distance between two points in the same cluster. each thread update the cluster in his cluster array (because the race condition).

- We merge the threads array into the original cluster array so that the original array will have the max diameter found.

**7.2 Calculate the quality**

-For each cluster we iterate over all the clusters to find the distance between the clusters centers- using OpenMP. Also we divide the cluster diameter with the distance. Summing the results will give us the clusters quality.

**Step 8- Finish**

-If the quality not achieved we continue iterating –back to Step 4.

-if the quality achieved we break the loop and the Master process write the result to output text file.

# Complexity

* **N** - number of points
* **K** - number of clusters to find
* **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
* **Th** - number of threads running in OpenMP parallel region

The heaviest operation in the program is evaluating the clusters quality – **Step 7**.

In this step I have the function calculate clusters diameters and two for loops iterating over the number of clusters.

* calculating diameters function is O( but with open MP we calculate max distances in the threads array - O( ) , and the merge – O(k\*Th)

total time = O( ) + O(k\*Th) ≈ O( )

* the two for loops afterwards is O(*),* but with open MP the complexity is O( ) .

total time = O ( ) .

Result of the quality evaluation = O( ) + O ( ) ≈ O( ) , k ≤ N

This function gets called inside another for loop that iterating over T/dT iterations.

**The total time is - O( ) .**