**Parallel implementation of K-Means – Project Overview Liran Zaltzberg 305631731**

# Problem Definition:

Given a set of points in **2**-dimensional space. Initial position **(xi, yi)** and velocity **(vxi, vyi)** are known for each point **Pi**. Its position at the given time **t** can be calculated as follows:

xi(t) = xi +t\*vxi

yi(t) = yi +t\*vyi

Implement simplified K-Means algorithm to find **K** clusters. Find a first occurrence during given time interval [0, T] when a system of **K** clusters hasa Quality Measure **q** that is less than given value **QM**.

# Initialize points and clusters:

* Process number 0 (master) read and the points and the algorithm info from the input file, then broadcasts the info to the other processes, and divides the points equally by using MPI\_SCATTER.
* Process number 0 (master) chooses first **K** points as a cluster centers.
* Every process allocates is points on the GPU – the points address on the GPU will be used throughout the entire algorithm, no multiple unnecessary memory allocations.

# Group points around the given cluster centers:

* Process number 0 (master) broadcasts the updated clusters in the beginning of each inner do while loop, then every process allocates his clusters on the GPU.
* Every process calls the function GroupPointsAroundClustersCentersHandler that that gets an updated points array that returns from the cuda kernel that allocates every cuda thread to a single points, and iterate over all clusters for a single point and assign it to the nearest one, so the O(n\*k) complexity is turned into O(k) because every thread handles one point.
* After the updated points array return from cuda kernel the process iterates over is CPU points **and clusters** and updates them according to the cuda result.
* In every GroupPointsAroundClustersCentersHandler call there is a reset of clusters sums and points number of each cluster that is parallelized by OMP.
* The breaking clause of the inner loop is if limit was reached or PointsMovedToOtherClusters was true, so after every GroupPointsAroundClustersCentersHandler call process updates the flags array and the master checks it and informs every slave if he need to be stopped.
* The slaves send their updated clusters to master, the master gets it and update his clusters array and then he recalculates the cluster centers.

# Evaluate the quality of clusters:

* After the inner loop conditions was reached the master gather all points from slaves.
* First, the master calculates the clusters diameters by an O(N^2) loop, that checks a distance between every points couple that are related to the same clusters, this function was parallelized by **OMP**, by creating a big clusters array as the number of OMP threads, every thread is assigned to certain amount of points, and updates his suitable place for the diameter calculation.
* After calculating the diameters the quality calculation is also parallelized by **OMP** in the same form as the diameters calculation.
* The master then broadcasts the quality to the slaves.

# Update Points:

* If quality measure was not reached every process updates his points by calling the UpdatePoints kernel, that updates every process GPU points and then copies it over his cpu Points – an O(N) loop is divided to each GPU thread.