# KMeans Cluster Algorithm Parallelization

## Problem Analysis and Design of Solution:

### Introduction:

Clustering is a well known machine learning technique which has been utilized in various applications including image segmentation, bioinformatic, weather forecasting and statistics. The most famous and least complex clustering algorithm is K-means on account of its simple execution, straightforwardness, proficiency and empirical achievement. However, its present application produces large volumes of data, in this way, how to effectively deal with these data is a significant mining task which has become a difficult and noteworthy issue. Likewise, MPI (Message Passing Interface) with distributed memory and OpenMP shared memory, a programming model of message passing presents superior performance, portability and scalability. Taking advantage of heterogeneous computing of NVIDIA CUDA Graphics Processing Units (GPUs) programming, I am spurred to explore an equal K-means clustering algorithm with MPI and CUDA called MPC-Kmeans. The algorithm is intended to empower the application of the clustering computation to adequately handle numeric weather forecast as well as abnormal climate event identification.

### Proposed Methodology:

For this problem, I will begin by converting the algorithm to a serial version of the code. The next step would be to (in any order) to work on a CUDA implementation for a single node, and work on an MPI version which decomposes the problems onto multiple processes, with each process doing their local part of the computation which contributes to the global solution. Once both of those parts are done, I will merge the two and have a distributed-memory (the MPI part) GPU (the CUDA part) version of a code to solve this problem.

### Standard Algorithm:

***k*-means clustering** is a method of vector quantization, originally from signal processing, that aims to partition *n* observations into *k* clusters in which each observation belongs to the cluster with the nearest mean (cluster centers or cluster centroid), serving as a prototype of the cluster. This results in a partitioning of the data space into Voronoi cells. The following is an animation exhibiting the K-mean algorithm, in view of a brilliant K-mean representation made by Naftali Harris blog: www.naftaliharris.com/blog/visualizing-k-means-clustering:

As presented in the above link; K-means algorithm also referred to as Lloyd's algorithm, particularly in the computer science community is sometimes also referred to as "naive *k*-means", because there exist much faster alternatives. Given an initial set of *k* means $m_1^{(1)},...,m_k^{(1)}$ (see below), the algorithm proceeds by alternating between two steps:

**Assignment step:** Assign each observation to the cluster with the nearest mean: that with the least squared Euclidean distance. (Mathematically, this means partitioning the observations according to the Voronoi diagram generated by the means.)

$$S_i^{(t)} = \{x_p : \|x_p - m_i^{(t)}\|^2 \leq \|x_p - m_j^{(t)}\|^2 \ \forall j, 1 \leq j \leq k\},$$

where each $X_P$ assigned to exactly one $S^{(t)}$ even if it could be assigned to two or more of them.

$$m_i^{(t+1)} = \frac{1}{|S_i^{(t)}|} \sum_{x_j \in S_i^{(t)}} x_j$$

**Update step**: Recalculate means (centroids) for observations assigned to each cluster.

The algorithm has converged when the assignments no longer change. The algorithm does not guarantee to find the optimum. Next, the derivation of the Pseudo C code, MPI and CUDA Codes.

### Parallel K-Means Algorithm

**MPI:**

With MPI, I would distribute data points to different processes using MPI_Allreduce and MPI_Bcast to exchange information whenever needed. Thus, both the M-step and the E-step can be parallelized.

### OpenMP

With OpenMP parallelization, only E-step would be directly parallelized. If M-step is directly parallelized with OpenMP pragmas, different data points might be added to one cluster at the same time, leading to Write-After-Write (WAW) hazard. Although it is possible to make drastic modifications to parallelize the M-step, it contradicts the basic idea of OpenMP that the serial code should be almost untouched. Therefore, we would only focus on the E-step.

### Hybrid MPI-OpenMP

I will also simply add OpenMP pragmas to the MPI code, to get the hybrid version. This time, I would have many combinations of OpenMP threads and MPI processes to test.

My main reference for the OpenMP & MPI Kmean algorithm is Bisgin 2008, along with their public code.

**The Main Feature: CUDA**

Given the massive potential of parallelism on GPUs, I would implement a parallel version of k-means algorithm using the Nvidia CUDA library. In my implementation, E-step would be parallize by distributing the computations of the nearest distance over blocks on "device".

**Computational Platforms:**

The MPI and CUDA programs are typically hosted on the Clusters of Gdansk Politechnika but for future advances, I would like to test it on Amazon EC2 Cloud Computing Environment (CUDA). For the data analysis, I would build a seamless interface between C and Pandas Python using NetCDF library along with other relevant libraries on my local Pc for necessary plots and visual representations.

**Application:**

Numerical Weather Prediction, Abnormal Climate event Identifier and Data & measure of Distance.

**Limitations and Future work:**

Time

Forest Covertype Classification