# K-Means Clustering

Enhancing performance and scalability through parallelism using MPI

Luís Neto (a77763), Gonçalo Raposo (a77211)

September 5, 2019

## 1 Case of Study

*K-means clustering* is an algorithm commonly used to partition data into $k$ groups based on their similarities. The algorithm consists in three step:

1. Selection of $k$ random centroids.
2. Assignment Step: each point is assigned to a cluster based on closest centroid determined with *Euclidean distance*.
3. Update Step: each centroid is updated to be the mean of his cluster.

The latter two phases are repeated until the centroids no longer change.

## 2 Algorithm Analysis

In this section, we describe the sequential and parallel algorithm's complexity, N will represent the data set size, K the number of clusters and P the number of processes.

For the sequential version, in the first phase we iterate the data set in order to obtain the maximum value, subsequentially, we generate all centroids, therefore, this phase complexity is $N + K$.

Phase 2 calculates the minimum distance between each point and centroids, consequentially, this phase complexity is $N * K$.

Finally, for phase 3, we iterate the clusters in order to calculate the first part of the error, update the clusters by iterating the data set and caculate the final error by iterating the clusters, hence, this phase complexity is $2 * K + N$. With this we can say that the sequential algorithm complexity is:

$$(N + K) + (N * (K + 1) + 2 * K) \tag{1}$$

In the parallel version, we equally divide the data set by the processes, so, we can rewrite the last complexity as:

$$(N + K) + (\frac{N}{P} * (K + 1) + 2 * K) \tag{2}$$

In this version, we also need to take in account the communication complexity, we will consider the following values for the primitives used.

1. Broadcast : $log_2(P)$
2. Reduce : $log_2(P)$
3. Scatter : $P$
4. Gather : $P$

After the first phase, we send the data set chunk to each process and clusters using the primitive Scatter and Broadcast, hence, the communication complexity is $2 * log_2(P) + P$.

At the end of second phase, we reduce the local counters of each cluster to one process and broadcast the result, therefore, the complexity is $2 * log_2(P)$.

After the update the local clusters, we reduce the local clusters to one process and broadcast the result, therefore, the complexity is $4 * log_2(P)$.

Finally, after the final error calculation, we broadcast a message to each process in order to stop or continue the algorithm. If the algorithm continues the complexity is $log_2(P)$, otherwise, the complexity is $P + log_2(P)$, due to the fact that we use a *Gather* primitive to collect the final result.

Having the communication complexity we can write the parallel algorithm complexity as:

$$(N + K) + (\frac{N}{P} * (K + 1) + 2 * K) + 9 * log_2 P + 2 * P \tag{3}$$

With this the speedup can be obtained by:

$$\frac{(N + K) + (N * (K + 1) + 2 * K)}{(N + K) + (\frac{N}{P} * (K + 1) + 2 * K) + 9 * log_2(P) + 2 * P} \tag{4}$$

# 3 Optimisation of previous versions

One of the most significant changes that were made was the memory alignment in the *OpenMP* version. This small change made a significant impact on the overall elements of the speedup chart.

# 4 Hybrid approach

After analysing the results from the previous approaches, we concluded that a possible implementation of a hybrid version of the algorithm should be, since the algorithm requires a lot of computation, a *hyperthreading* version of the *MPI* solution. With this type of solution, we could explore the maximum amount of the memory hierarchy since L1 and L2 are private to each core available and since every core will be executing a different process with different data in these.