Parallel Implementation of K-Means Clustering using MPI

Abstract

Clustering algorithms are used in various fields such as visualization, pattern recognition, learning theory, computer graphics, neural networks, AI, and statistics. Clustering large data sets can be time consuming and processor intensive.

In this project, I implement a serial and a parallel version of a popular clustering algorithm, the K-means algorithm, to provide faster clustering for large amounts of datasets. Both algorithms were implemented in C++. For the parallel version, I used the MPI library in C++.

Background

Clustering is the process of partitioning or grouping a given set of patterns into disjoint clusters. is the grouping of similar objects and a clustering of a set is a partition of its elements that is chosen to minimize some measure of dissimilarity.

Data clustering is frequently used in Data Mining. Data clustering algorithms are generally classified into 2 major categories: Hierarchical algorithms, and, Partition algorithms.

A hierarchical clustering algorithm builds a hierarchy of clusters that are organized as a tree in an iterative manner. Strategies of hierarchical clustering generally fall into two types: Agglomerative (bottom up approach) and Divisive (top down approach).

A partitional clustering algorithm divides the dataset into non-overlapping subsets (i.e. clusters) such that each data object is in exactly one cluster. Then it iteratively improves the clusters until no cluster can be further improved.

Goal of clustering is to maximize inter-cluster similarity and minimize inter-cluster similarity between data objects.

What is K-Means Clustering?

Given a dataset, D, of n objects, and k, the number of clusters to form, the k-means algorithm organizes the objects into k partitions, where each partition represents a cluster. The objective is to have high intra-cluster similarity b/w data objects and low inter-cluster similarity. Data objects have 1 or more attributes. Attributes must be of type integer. Similarity is measured by calculating the Euclidean distance between data objects.

K-Means: Serial Algorithm

Input:

- k: the number of clusters,
- D: a data set containing n objects.

Output: A set of k clusters.

Method:

(1) arbitrarily choose k objects from D as the initial cluster centers;
(2) **repeat**
(3) (re)assign each object to the cluster to which the object is the most similar, based on the mean value of the objects in the cluster;
(4) update the cluster means, that is, calculate the mean value of the objects for each cluster;
(5) **until** no change;

Figure 1: Serial Algorithm Pseudo Code

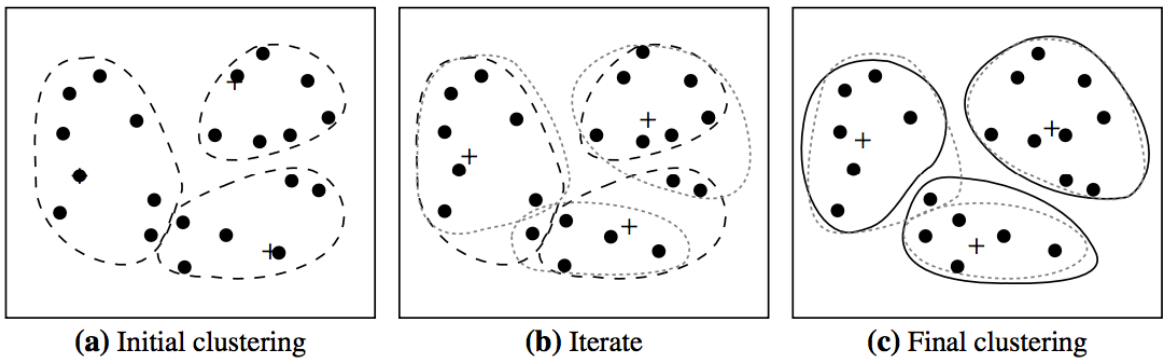

Figure 2: Graphical Illustration of the Algorithm. '+' represent cluster means. Black dots are the data elements.

Time Complexity for the serial version: **O(K*N*T)**, where,

- K = Number of Clusters to form
- N = Number of Elements in the Dataset
- T = Number of iterations that the algorithm takes

K-Means: Parallel Algorithm

- At the beginning of the algorithm, each processor holds a portion of the Dataset "D"

- If the total number of data objects in “D” is “N”, and the total number of processors is “P”, then each processor receives “N/P” data objects.
- That means the computation time is reduced by a factor of “P”.

Time Complexity for the Parallel version = *computation_time* + *communication_time*.

computation_time = **O(K*T*N/P)**
communication_time = **O(K*P + T*(P*P))**, where,

- **K*P** = Initial broadcast of K vectors from the root to the P processors.
- **T*(P*P)** = All-to-all reduce for all the ‘P’ processors for ‘T’ iterations.

K-Means: Parallel Algorithm Pseudo Code

Root (process with rank=0) calculates the initial means of the “k” centroid vectors.
While TRUE {

- Root Broadcasts these “k” vectors to all the processors.
- Each process computes distance of each local document vector to “k” centroid vectors. (P.S. Note that size of 1 centroid vector is equal to the number of attributes in a data object.)
- Each process then re-computes the local “k” centroid vectors based on the re-assigned objects.
- Each process then does an All-to-All reduction of the “k” centroid vectors. Thus, after each iteration, an all-to-all reduction synchronizes the “k” centroid vectors across all processors.
- BREAK the loop when there is no change in the values of all the “k” centroid vectors in 2 consecutive iterations of the loop.

}

Testing: Data Sets

- Data Sets taken from UCI Machine Learning Data Repository.
- http://mlr.cs.umass.edu/ml/machine-learning-databases/
- I chose 3 data sets: Iris, Wine, Letter Recognition.
- For each of these data sets, number of processes in MPI is varied 1 – 40, and time taken for the program to run is calculated.

Testing: Results/Graphs

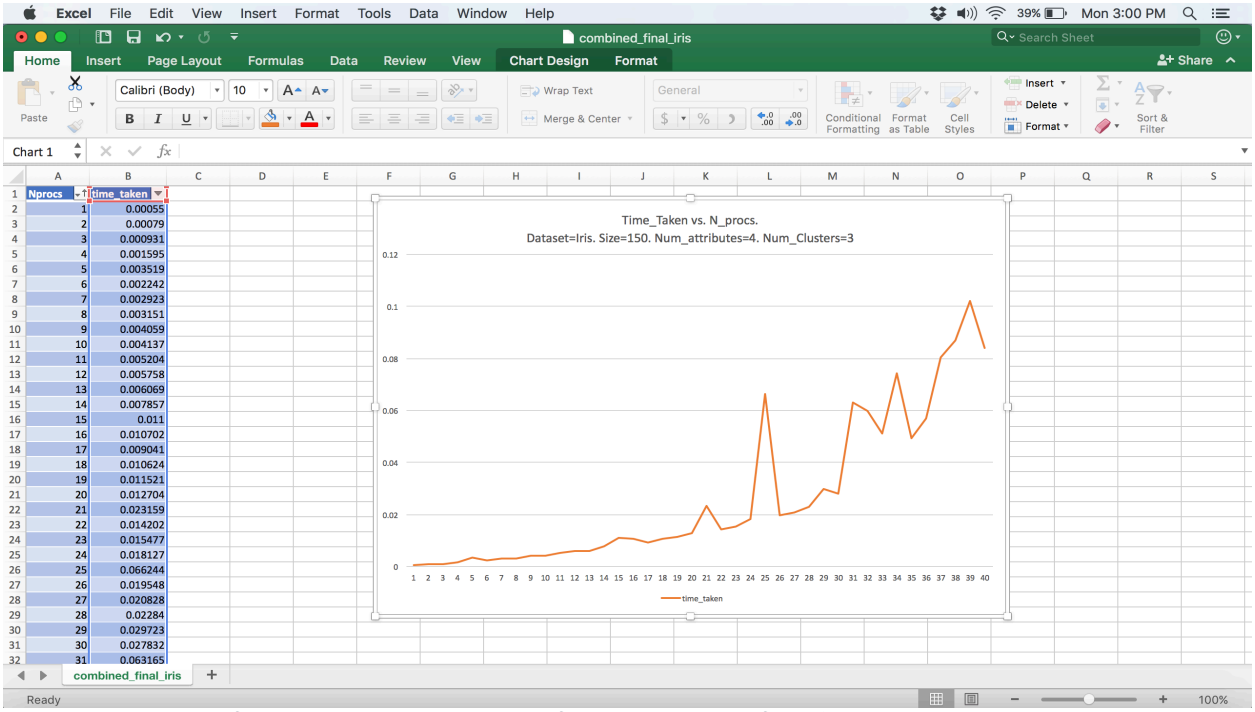

Figure 3: Time taken for Iris Dataset. MPI Number of Processes varied from 1-40

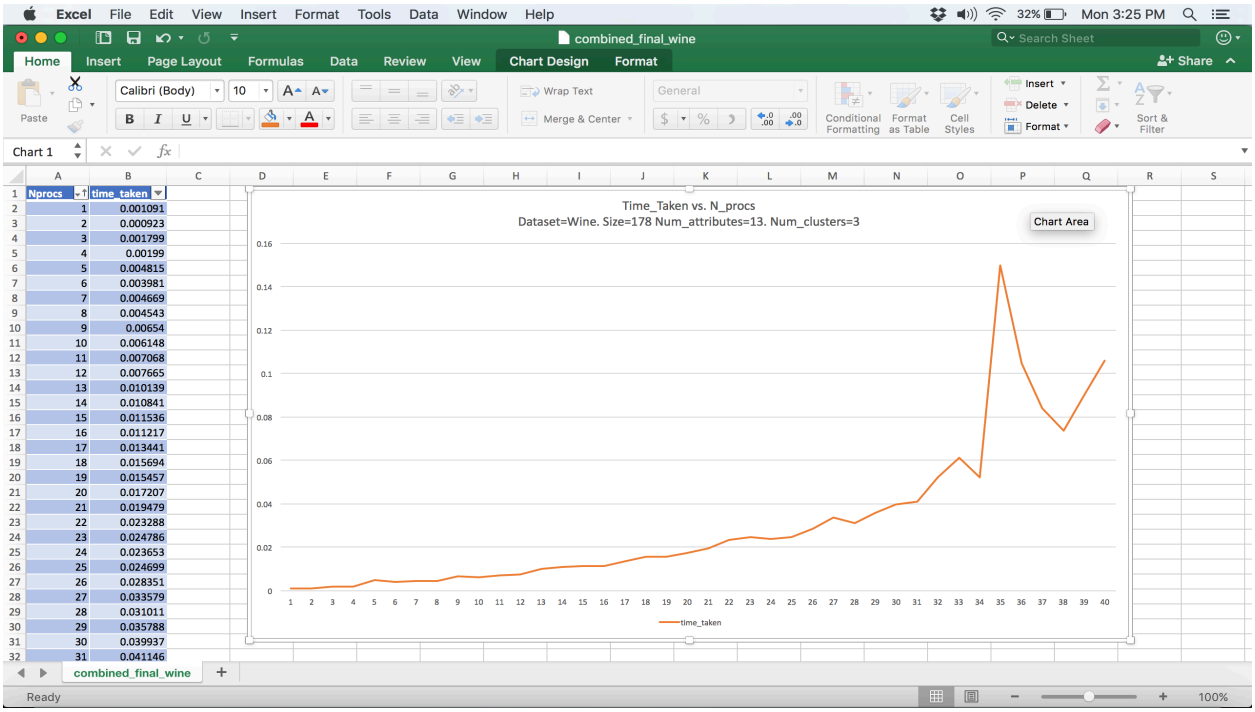

Figure 4: Time taken for Wine Dataset. MPI Number of Processes varied from 1-40

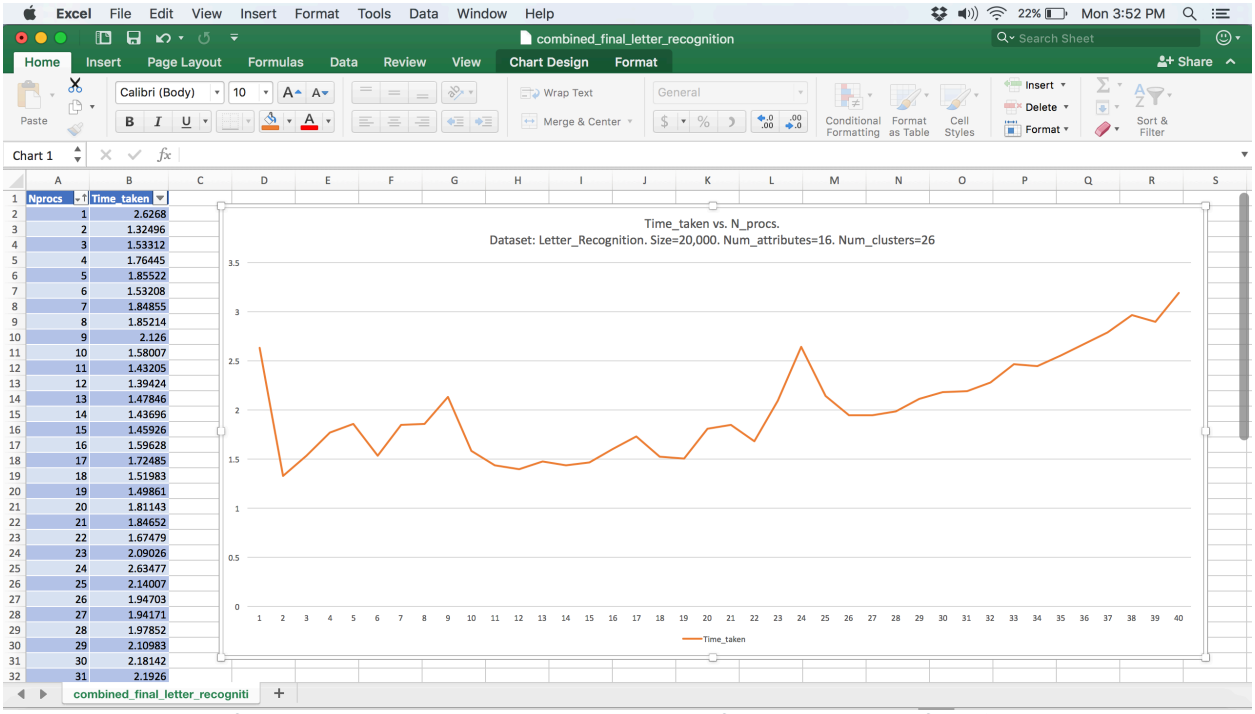

Figure 5: Time taken for Letter Recognition Dataset. MPI Number of Processes varied from 1-40

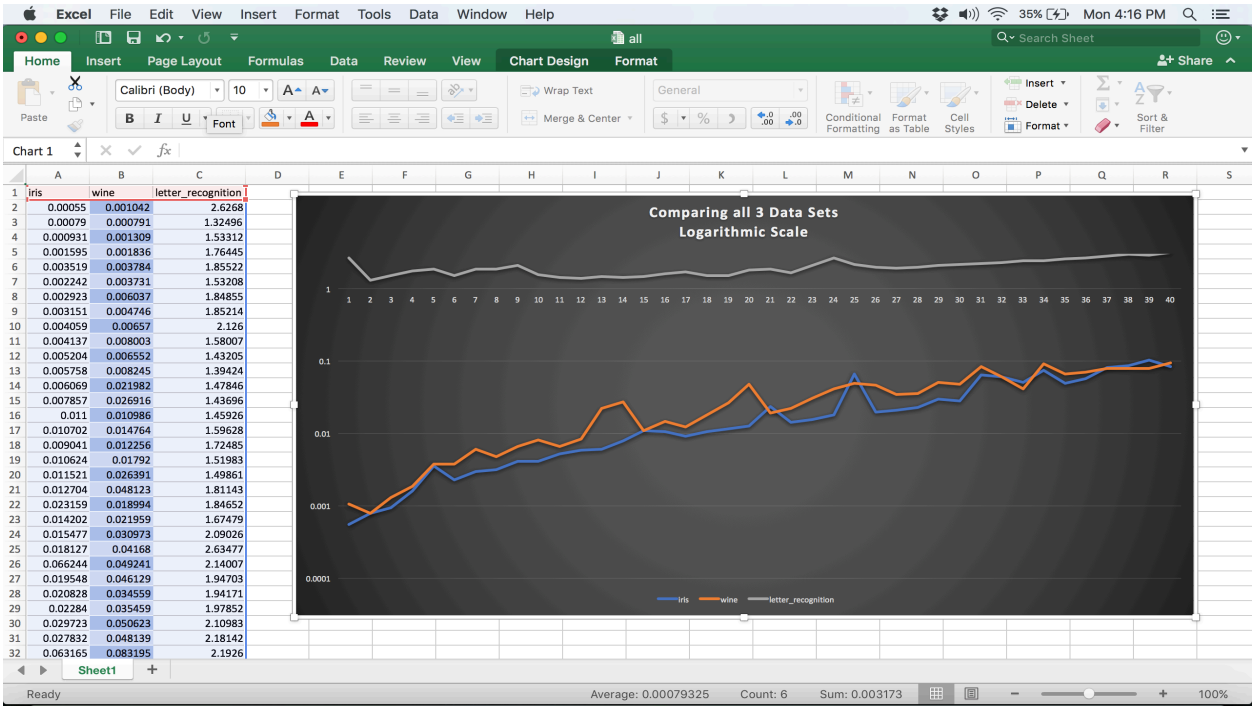

Figure 6: Time taken for all Datasets. MPI Number of Processes varied from 1-40

Sources

- http://www.csee.umbc.edu/~hillol/PUBS/review.pdf
- http://lansainformatics.com/wp-content/plugins/project-mgt/file/upload/pdf/2440Data-mining-with-big-data-pdf.pdf
- https://pdfs.semanticscholar.org/34ad/09cda075101dc4ce3c04006ff804aca3ebf8.pdf
- https://pdfs.semanticscholar.org/5a97/ad653a7213ee4abad846eddf86f4e858e999.pdf
- http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.101.1882&rep=rep1&type=pdf