Parallel K Means Clustering Algorithm.

1.Introduction

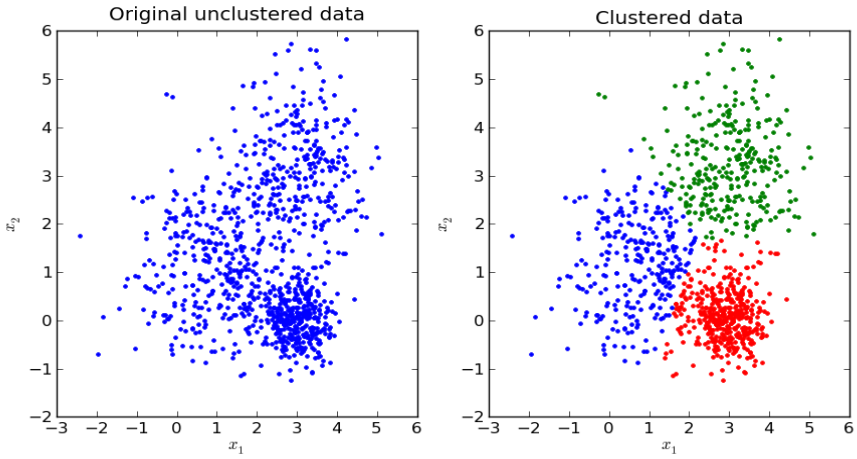
Clustering is a technique used in data analysis and machine learning to group data into clusters based on their similarity. Clustering algorithms aim to organize data into groups such that data within each group is more similar to each other than to data in other groups. This can help to uncover hidden patterns and relationships in the data, and can be used for a variety of applications such as data compression, outlier detection, and classification. There are many different types of clustering algorithms, each with its own strengths and weaknesses. Although there are numerous distinct clustering methods, K-means clustering is most well-used.

K- means clustering technique is known for its simplicity. The approach uses random centroids as a starting point and calculates squared Euclidean distance to assign data points to the closest cluster. After that, the centroids are adjusted, and the procedure is repeated until convergence. The linear computational complexity of the K-means clustering technique is O(nKt), where n is the number of data points, K is the desired number of clusters, and t is the number of iterations required for the algorithm to converge. The problem with this technique is that it cannot be scaled up because it spends a lot of time allocating points to the closest cluster.

Using Python's multiprocessing package, we suggest a parallelization of the K-means clustering technique in this project. The runtime and speedup of our parallel version and the fundamental K- means clustering approach are then compared using a series of simulations. We also investigate the effects of expanding our parallel approach to use more CPUs. Our parallel K-means clustering approach, however, greatly reduces overall time complexity as the number of data points rises.Added measures for speedup, scaleup and sizeup provide further evidence that our parallel K-means clustering algorithm is both robust and scalable..

2.Intuition behind K Means clustering

Setting the appropriate number of clusters, K, is the initial step in the K-means clustering process.Each data point is subsequently assigned to one of the K clusters by the K-means clustering method. The results of K-means clustering are shown in the following graphic for various K values.



The K-Means clustering algorithm applied to a simulated dataset with 300 observations in a two-dimensional space for K =3. The different colors represent separate clusters.

By using squared Euclidean distance as a measure of within-cluster variance, the K-means clustering method aims to find a solution to the issue. In this issue, we have a collection of Euclidean space points P R. Finding a set of K points with d C R such that the sum of the squared Euclidean distances between each point in P and its nearest centroid in C is reduced is the objective. Consequently, the goal function is:

$$\varphi(C) = \sum_{p \in P} min_{c \in C} \|p - c\|^2 ,$$

where |||| stands for Euclidean distance squared. By assigning each data point to its nearest centroid, the data points are divided into K clusters. The K-means clustering algorithm further needs the K-cluster count to be predetermined

2.Basic K- Means Clustering Algorithm

The basic K- means clustering technique is simple and easy. We begin by selecting K initial centroids at random. K stands for the desired number of clusters, which is normally given by the user. After that, we "assign stage" each data point to the closest centroid. A cluster is used to describe each set of points that is produced and allocated to a centroid. The centroids are then updated (referred to as the "update step") using the fresh data from each cluster. Up until our method converges or the maximum number of iterations is achieved, we repeatedly perform the assignment and update stages iteratively. The steps that we perform for the k-means clustering are:

- Randomly generate initial centroids
- Form K clusters by assigning each data point to its nearest centroid
- Update the centroids by calculating the cluster means
- Repeat steps 2 and 3 iteratively until convergence or maximum iterations is reached

Initially we have points for centroids, then we will perform iterative steps. First, we form clusters by calling the function assign_points_to_cluster(). The function distance_() to

calculate the squared Euclidean distance between each data point and the initial centroids and assigns each data point to the cluster for which this distance is minimized. The output of minimum_() is a vector of labels corresponding to each of the K clusters. The second part of assign_points_to_cluster() uses nested for-loops in order to map these cluster labels to the corresponding data points. The function then returns X_by_cluster, a list of K arrays corresponding to the data points within each cluster. Then, update our centroids by using the information in X_by_cluster to calculate the cluster means. We then check to see if our algorithm has converged.

3.Parallel K-Means Clustering Algorithm

While our basic K- means clustering algorithm is quite simple, it is also very computationally expensive for large datasets. Each iteration of the method until convergence calls for the calculation of squared Euclidean distances between each data point and the K centroids. We divide the task of allocating points to clusters among p processors to parallelize it and speed up the execution. Since there are n observations in the dataset, each processor only needs to finish this operation for n/p data points. Then, we compile the outcomes from each processor and utilize them to update our centroids. Steps that involved in parallel K-means clustering Algorithm are:

- Randomly generate initial centroids
- Randomly shuffle the data points and partition them into p subgroups
- For each subgroup p, assign each data point to its nearest centroid
- Aggregate the results from each subgroup p to form K clusters
- Update the centroids by calculating the cluster means
- Repeat steps 2-5 iteratively until convergence or maximum iterations is reached

In K-means parallel clustering we are considering the function fit()which is taken as input which is representing the complete set of data points. The fit(),similar to basic K -means clustering algorithm, serves to randomly generate the initial centroids. The next part, however, is where our parallel algorithm diverges. We begin our iterative steps by calling a function partition_() that randomly shuffles the data points in X and divides them into p subgroups according to the user specification for the parameter n_cores. Next, we make use of Python's multiprocessing library to create a pool . We use this pool to map a task to each of the p available processors. By doing this, we can perform the assignment stage for each subgroup of data points simultaneously. This results in p versions of X_by_cluster, a list of K arrays corresponding to the data points within each cluster. We use nested for loops to aggregate each of the p lists to get a complete account of all the data points assigned to each of the K clusters. We proceed from here in a similar manner to the basic K-means clustering algorithm by updating the centroids and checking for convergence.

4.Experimental Procedure

We evaluate the accuracy and performance of our K-means clustering algorithms by generating artificial datasets using the make_blobs() function from Python's sklearn.datasets module. We specify the following parameters:

- n_samples, the number of observations to be divided equally among the clusters.
- centers, the desired number of clusters.
- cluster_std, the standard deviation of the clusters; and
- random_state, which enables your output to be reproducible when an integer value is passed.

Our code is executed on a machine with the following characteristics detailed in Table

Feature	Description
CPU Model Name	11th Gen Intel(R) Core(TM) i7-11800H @ 2.30GHz 2.30 GHz
CPU Architecture	x86_64
CPU(s)	8
RAM	16GB

4.Evaluating Algorithm Accuracy and Performance

For the given Algorithm, accuracy of our basic and parallel K-means clustering algorithms, we cross-reference our results with the results from the KMeans() function housed within Python's scikit-learn library.To measure the runtime here we considered, timeit() function within Python's timeit library . The timeit() function allows us to record more accurate runtimes than do other similar functions, such as time(). Initially, by default turns off garbage collection during the timing. Next, iteration the function timeit() automatically picks the most accurate timer for our system. Last, timeit() allows us to repeat tests multiple times to eliminate the influence of other tasks that may be occurring on our machine. The closer the initial centroids are to the final centroids, the faster the algorithm converges

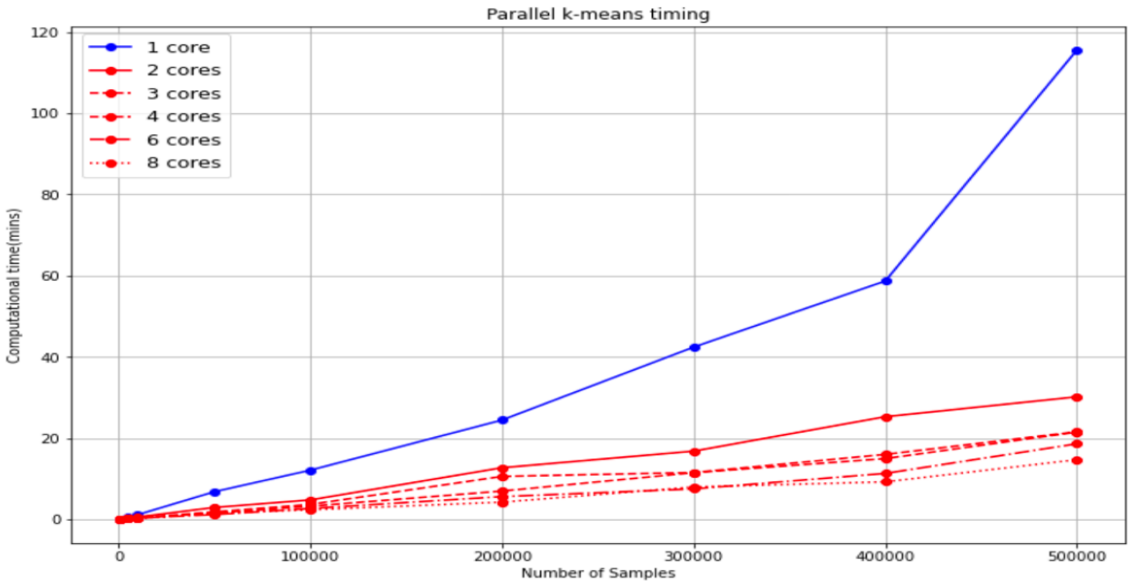
.

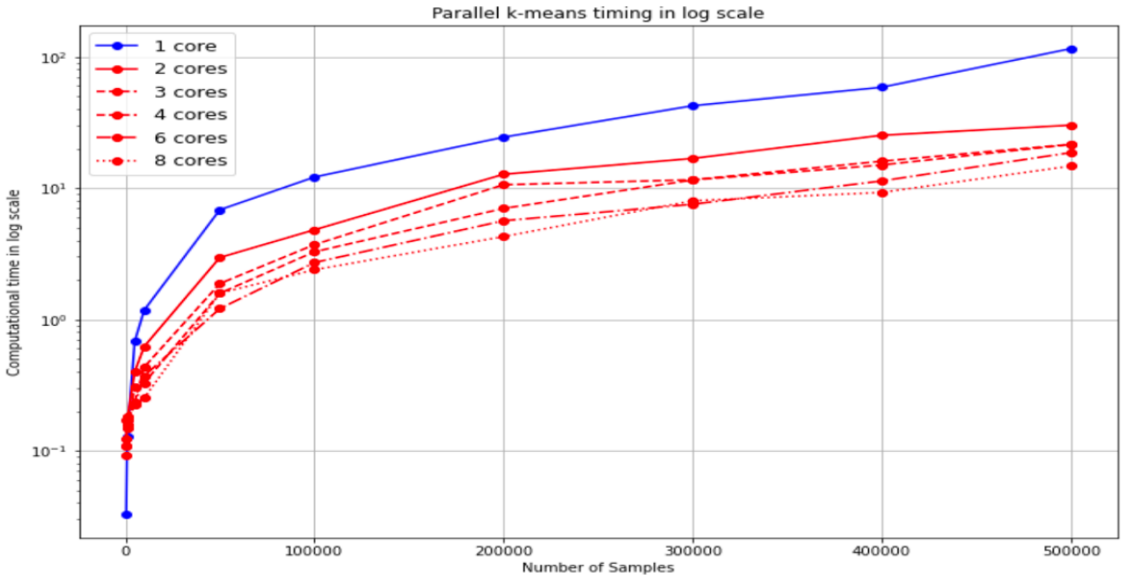
5.Experimental Results

Basic and parallel K-means clustering techniques are evaluated through various simulations using artificially generated datasets. In these simulations, we vary the number of data points while maintaining a consistent dataset with 2 dimensions and the necessary number of clusters. For our parallel K-means clustering method, we change

the number of processors. Table provides a detailed breakdown of the computational speeds for each of our simulations.

No.of Samples	Serial	2 Cores	3 Cores	4 Cores	6 Cores	8 Cores
500	0.03280352	0.16968365	0.12404876	0.10815304	0.1088206	0.09290097
1000	0.1276899	0.16980702	0.15808786	0.14781801	0.17765755	0.18362063
5000	0.67862014	0.39854986	0.23327839	0.3048646	0.22485415	0.22782555
10000	1.17679915	0.62129957	0.43309855	0.32797232	0.37106583	0.25198351
50000	6.83519494	2.96046755	1.87455277	1.58221045	1.20842524	1.58387674
100000	12.130214	4.81134537	3.70805233	3.27136087	2.70983727	2.38814976
200000	24.4660232	12.7425474	10.6044294	7.00283026	5.64016894	4.27247205
300000	42.4197249	16.8247024	11.5733578	11.5234972	7.53062098	8.01251969
400000	58.680946	25.2893798	16.0238947	14.9955345	11.3444062	9.26665285
500000	115.566191	30.1924458	21.5995608	21.5161029	18.6453909	14.7269645

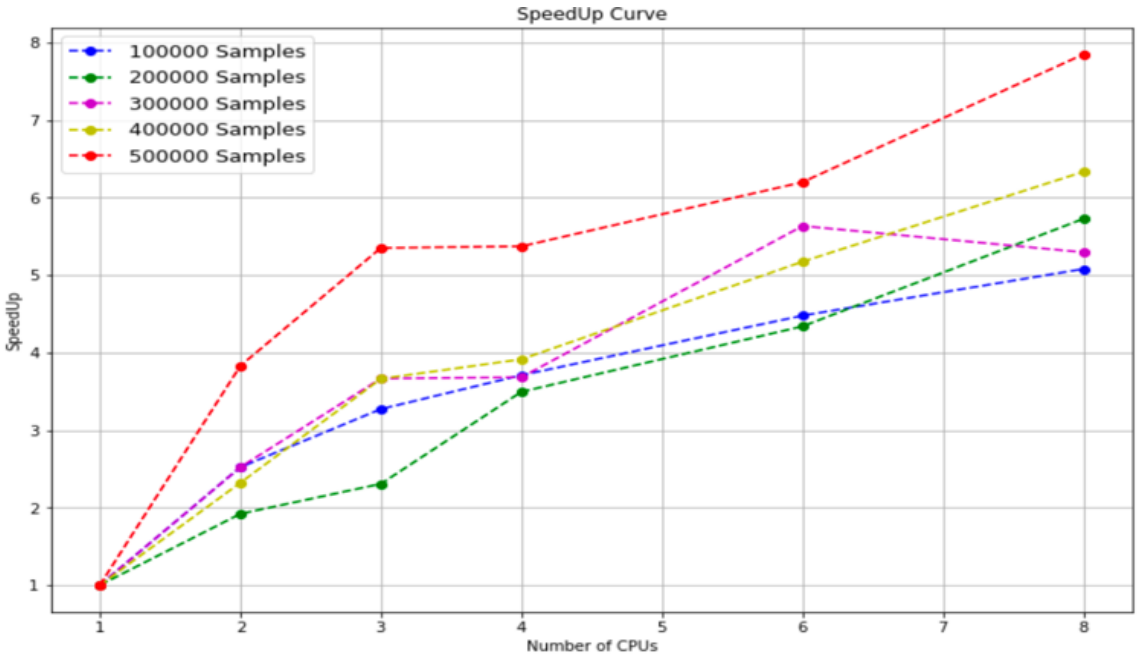




According to Table, the basic K-means clustering method performs better than the parallel K-means clustering algorithm for datasets with fewer than 5,000 observations. The overhead of processor communication is probably to blame for this.
However, the parallel K-means clustering algorithm works better when there are 5,000 or more observations. The difference in runtimes between the basic and parallel K-means clustering techniques is increasingly pronounced with subsequent increases in dataset size. Our findings point us that for datasets with fewer than 50,000 observations, the runtimes for various numbers of processors are relatively comparable when using the parallel K-means clustering technique. However, the disparities in runtimes become much more noticeable if the dataset gets even bigger.Runtime alone is not a sufficient indicator of scalability, despite the information in Table and Figure about the variations between our basic and parallel K-Means clustering algorithms.We therefore incorporate measures of speedup, scaleup, and sizeup to gain more insight into the advantage offered by our parallel K-Means clustering algorithm.

SpeedUp

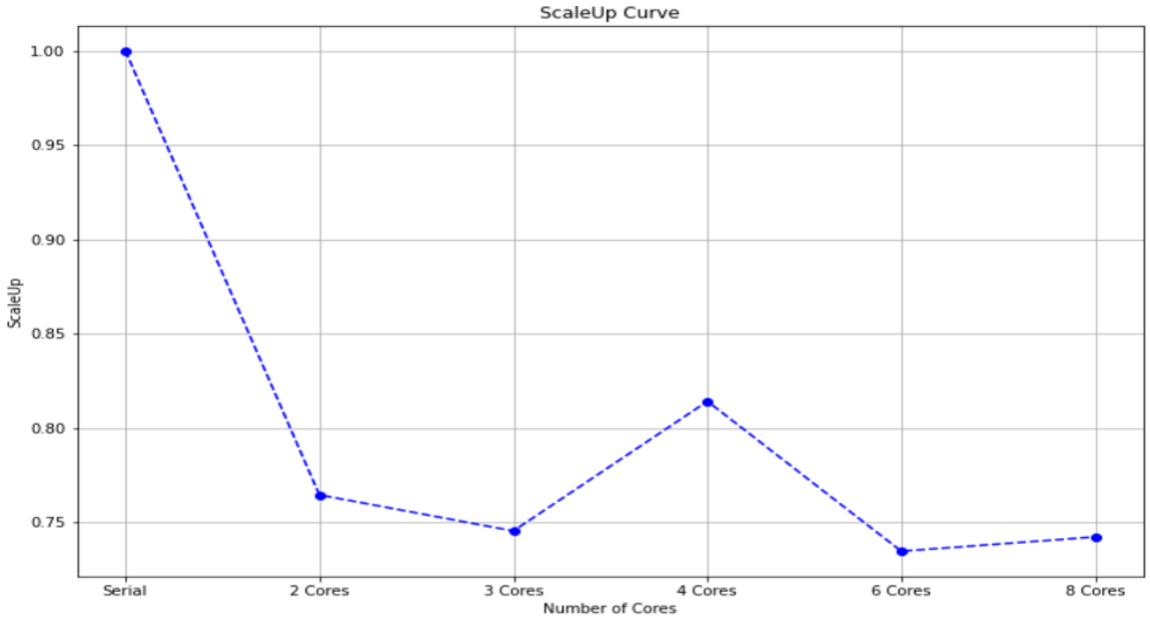
By altering the number of processors being used while maintaining a constant number of observations in a dataset, speedup can be determined. The time it takes for the algorithm to run on one processor (T1) and the time it takes for the method to run on n processors (Tn) must be recorded in order to determine speedup. If an algorithm speeds up linearly, it is said to be fully parallel. Our simulation findings for datasets with 100,000 to 500,000 observations and 1 to 8 CPUs are used to determine speedup.



The plot shows how the speedup of the parallel K-means clustering algorithm varies with the number of processors used and the size of the dataset. Speedup is a measure of how much faster the parallel algorithm is compared to the basic K-means algorithm. The plot shows that as the number of processors increases, the speedup also increases linearly. This indicates that the parallel K-means clustering algorithm is able to take advantage of multiple processors to improve its performance. This is a strong indication of the algorithm's good performance, as it is able to scale efficiently with the number of processors used.

ScaleUp

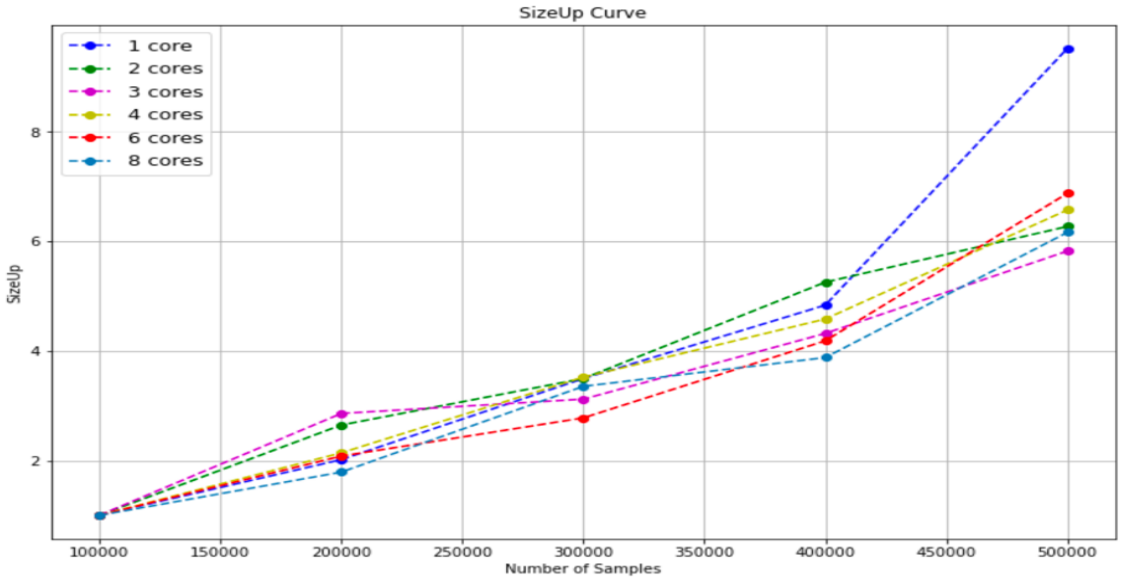
The ability of a n-times larger system to do a n-times larger job in the same amount of time is known as scaleup. It is determined by increasing the dataset's observation count and the number of active processors. Calculate the time it takes for the algorithm to run on one processor on a dataset of size s (T) and the time it takes on p processors on a dataset of size n x s (Tps) in order to measure scaleup.If the ratio is near to 1, an algorithm is said to have good scaleup. Our simulation results, which show that a dataset's number of observations grows in direct proportion to the number of processors being used, are used to measure scaleup. To be more precise, a dataset with 100,000 observations is processed by one processor, a dataset with 200,000 observations is processed by two processors, and so on up to 8 processors.



The scaleup outcomes for different processors are shown in the above plot. The general trend is downward. However, it is evident that our scaleup value for each processor count is still rather near to 1. Again, our simulation results for the dataset with 400,000 observations are to blame for the slight jump in the scaleup for 4 processors. Overall, the scalability of our parallel K-means is good.

SizeUp

Sizeup is a measure of how well an algorithm can handle larger datasets by keeping the number of processors used constant and increasing the number of observations in the dataset by a factor p. To measure sizeup, the time it takes for the algorithm to run on a dataset of size s (Ts) is compared to the time it takes for the algorithm to run on a dataset of size p x s (Tps). A good sizeup performance is indicated by a ratio that is close to p. In this study, sizeup is measured using simulation results with the number of processors fixed at 1, 2, 3, 4,6 and 8 cores. The dataset used has 100,000 observations and is increased by a factor p ranging from 1 to 5..



The results of our sizeup for various dataset sizes and processor counts are shown in the above image. The sizeup numbers are fairly close to the increase factor p, which runs from 1 to 5. This shows that the parallel K-means clustering technique can support huge datasets in an effective manner.

6.Conclusion:

K-means clustering is a straightforward yet effective method with several applications in a wide range of sectors. The K-means clustering algorithm must therefore be capable of processing datasets of various sizes and complexity. With datasets with fewer than 5,000 observations, the fundamental K-means clustering algorithm functions effectively, but anything greater causes it to become computationally expensive.We demonstrate that the computational time significantly decreases for datasets with more than 5,000 observations by taking use of the "embarrassingly parallel" feature of the K-means clustering algorithm. Our measurements of speedup, scaleup, and sizeup offer additional support for the robustness and scalability of our parallel K-means clustering approach.

7.References:

[1] Kerdprasop, K. & Kerdprasop, N. (2010) Parallelization of K-Means Clustering on Multi-Core Processors. Selected Topics in Applied Computer Science, pp. 472-477. ISSN: 1792-4863.

[2] Li, X. & Fang, Z. (1989) Parallel clustering algorithms. Parallel Computing 11(3): 275-290.

[3] Dhillon, I. & Modha, D. (2000) A Data-Clustering Algorithm on Distributed Memory Multiprocessors. Large-Scale Parallel Data Mining. Lecture Notes in Computer Science, vol 1759. https://doi.org/10.1007/3-540-46502-2_13

[4] Documentation for Python multiprocessing Library.
https://docs.python.org/3.7/library/multiprocessing.html

[5] Documentation for Python sklearn.datasets.make_blobs Function.
https://scikit-learn.org/stable/modules/generated/sklearn.datasets.make_blobs.html

[6] Documentation for Python sklearn.cluster.KMeans Function.
https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html

[7] Documentation for Python timeit Library.
https://docs.python.org/3/library/timeit.html