Project Report

On

Parallelizing K-Means Clustering with OpenMP

Submitted by

Arjit Arora (170001009)

Kalpit Kothari (170001025)

Computer Science and Engineering

3rd year

Under the Guidance of

Dr. Surya Prakash

Department of Computer Science and Engineering

Indian Institute of Technology Indore

Autumn 2019

Introduction

K-means clustering is an unsupervised learning algorithm that is used to divide an unlabelled dataset into a certain number of distinct subgroups (clusters) where each data point is mapped to some subgroup. K-means finds its use in hugely varied applications like customer segmentation, insurance fraud detection, pattern recognition, and image processing. Only around 0.1% of the data generated every day is being analyzed. Unsupervised learning algorithms like this help find structure in this hugely uncategorized data.

Methodology

Given an input dataset of m points, we have to find K clusters such that the variance of the data with respect to the K-clusters is the least.
The K-means algorithm comprises of three major steps:

1. **Initialization of centers** : There are multiple ways to do so. For example,
    - Randomly in the space of the domain
    - Randomly picked 'K' points of the input dataset
2. **Label assignment :** All points are assigned to the center they are closest to.
3. **Center reassignment** : The mean of the data points assigned to each cluster is calculated and the corresponding center is updated to the value of mean.

$$\mu_{c^k}' = \frac{1}{m_k} \sum_{i=1}^{m_k} (x^i - \mu_{c^k})$$

where μ_{c^k}' is the new value of the center.

Sequential Algorithm:

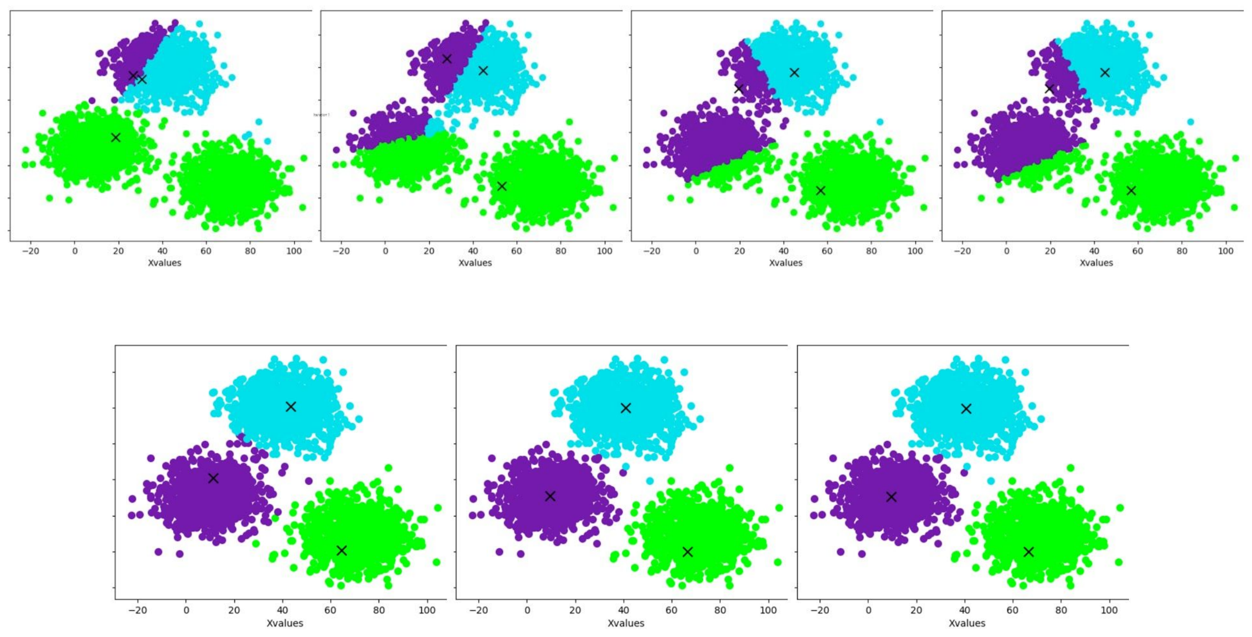
```
Step 1: Initialize the centers $\mu_1, \mu_2, \dots, \mu_k$
Step 2: Assign labels to each point of the dataset
Step 3: Compute the mean of data points assigned to each set.
Step 4: Initialize a flag $\leftarrow 0$ to check if any center changed
    Reassign the centers $\mu_1, \mu_2, \dots, \mu_k$
Step 5: If flag = 0
        Terminate
Step 6: Go to step 2
```

The algorithm can be parallelized by making certain changes to the skeleton of the above procedure.

A. The assignment of labels to each point of the dataset can be parallelized. The distance of every dataset point is from each cluster can be calculated in parallel, and the point is assigned a label corresponding to the center from which the distance is minimum.
B. Further, we need the mean of data points assigned to each cluster. This can be sped up by first separating the points of each set by **list packing** in O(log(n)) time.
C. The summation of the generated lists can be done in O(log(n)) time and O(nlog(n)) work. The work can be reduced to O(n) by first performing the summation sequentially over batches of log(n) points to reduce the size of the array to O(n/log(n)) and then computing the sum via the **balanced tree** method.

The algorithm can be further optimized by parallelizing the reassignment of the centers.

In this project, we've used C++ and OpenMP to implement the aforementioned sequential and parallel versions of the K-means algorithm. To aid visualization of successive iterations we've used **matplotlib** module to plot the labels and centers of data points after each iteration.



Graph 1. Visualization of K-means clustering. Size of dataset - 5000, Clusters - 3

Results

By parallelizing K-means clustering, we've achieved considerable improvements in the order of time complexity. The time complexities and execution times for the sequential and parallel versions of the algorithm can be seen in the table below:

Theoretical Results:

Algorithms	**Sequential**	**Parallel**
Time Complexity	O(n*k)	O(log(k)*log(n))
Work	O(n*k)	O(n*k*log(n)*log(k))
Cost	O(n*k)	O(n*k*log(n)* log(k))

Table 1. Performance analysis per iteration

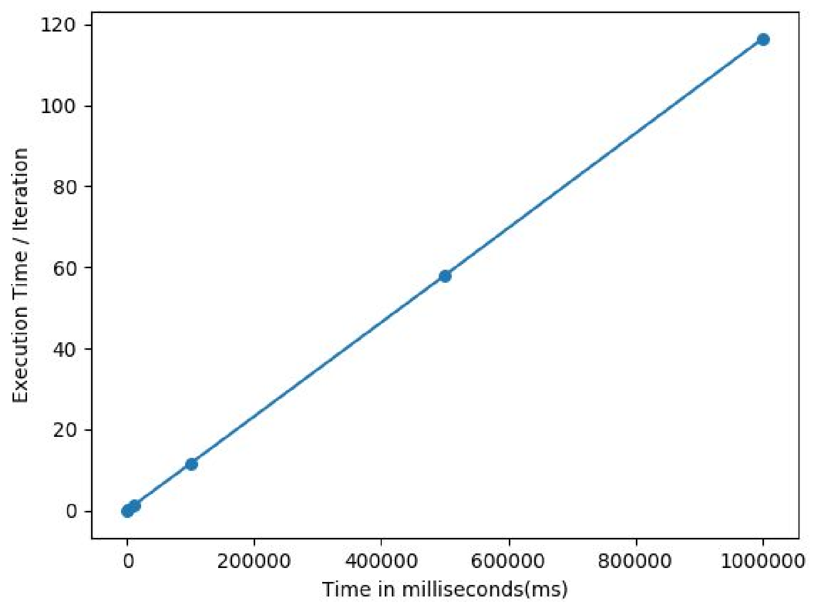
Observations

For K = 4 Clusters, we have the following execution times.

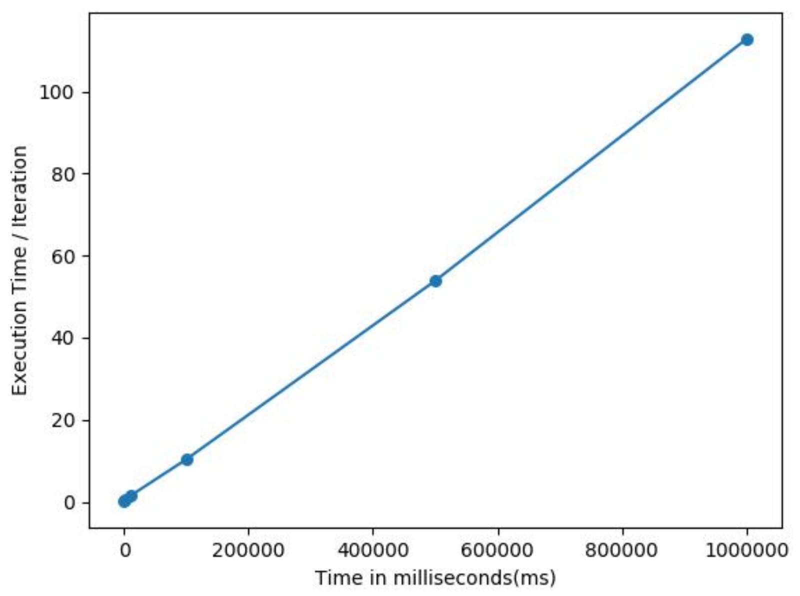
Size of dataset	**Time (Sequential) t_s (ms)**	**Time (Parallel) t_p (ms)**	**Speedup (t_p/t_s)**
100	0.0456046	0.29389	0.15551
1000	0.258456	0.682006	0.37896
10000	1.23774	1.42886	0.86624
100000	11.6062	10.3273	1.12383
500000	57.9781	53.8449	1.07676
1000000	116.328	112.729	1.03192

Intel(R) Xeon(R) Gold 6132 CPU @ 2.60GHz with 28 Cores

Table 2. Performance analysis on datasets of various sizes



Graph 2. Execution time per iteration for sequential algorithm with K=4 clusters



Graph 3. Execution time per iteration for parallel algorithm with K=4 clusters

Due to the various overheads, the execution time for the parallel algorithm is not as good as expected. Also, the algorithm expects processors to be of O(n), but the hardware used has only 28 processors. As we can see from the graph, as the size of the dataset increases, the performance of the parallel algorithm improves rapidly.

Conclusion

1. The above parallel algorithm has optimal time complexity, but it is not work optimal.
2. The list packing can be performed in O(n) complexity with which the algorithm can be made work optimal on a CREW machine.
3. Providing an adequate number of processors and a sufficiently large test data size will further improve the speedup of parallel algorithm relative to sequential.

References

1. *Qing Liao, Fan Yang and Jingming Zhao, "An improved parallel K-means clustering algorithm with MapReduce," 2013 15th IEEE International Conference on Communication Technology, Guilin, 2013, pp. 764-768.*
2. *D. S. B. Naik, S. D. Kumar and S. V. Ramakrishna, "Parallel processing of enhanced K-means using OpenMP," 2013 IEEE International Conference on Computational Intelligence and Computing Research, Enathi, 2013, pp. 1-4.*
3. *I. Borlea, R. Precup, F. Dragan and A. Borlea, "Parallel Implementation of K-Means Algorithm Using MapReduce Approach," 2018 IEEE 12th International Symposium on Applied Computational Intelligence and Informatics (SACI), Timisoara, 2018, pp. 000075-000080.*
4. *Datasets Used :*
    a. *https://www.kaggle.com/hdriss/xclara/download*
    b. *http://cs.joensuu.fi/sipu/datasets/s1.txt*