Parallel K-means Clustering

Ajay Padoor Chandramohan

Fall 2012

CSE 633

Outline

Problem description
Implementation – MPI
Implementation – OpenMP
Test Results
Conclusions
Future work

Problem Description

- **Clustering** is the task of assigning a set of objects into groups (called **clusters**) so that the objects in the same cluster are more similar (in some sense or another) to each other than to those in other clusters.

As a naïve example: animals can be clustered as land animals, water animals and amphibians.

- ***k*-means clustering** is a method of clustering which aims to partition *n* data points into *k* clusters (n >> k) in which each observation belongs to the cluster with the nearest mean.
- The nearness is calculated by distance function which is mostly Euclidian distance or Manhattan distance.
- One important assumption to be made is the data points are independent of each other. In other words there exists no dependency between any data points.

Problem Description – Sequential Algorithm

Given an initial set of *k* means $\mathbf{m}_1^{(1)},\ldots,\mathbf{m}_k^{(1)}$

(initially k random data points are assigned as K means) where m_i is the means of the cluster i.

Assignment step: Assign each observation to the cluster with the closest mean

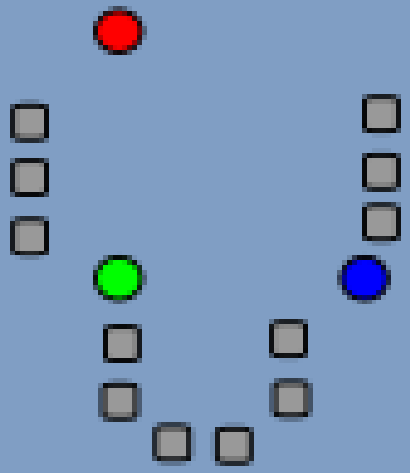
$$S_i^{(t)} = \{x_p : \|x_p - m_i^{(t)}\| \leq \|x_p - m_j^{(t)}\| \ \forall\ 1 \leq j \leq k\}$$

Update step: Calculate the new means to be the centroid of the observations in the cluster.

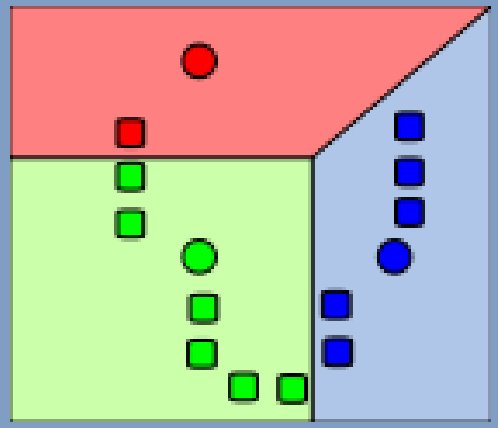
$$\mathbf{m}_i^{(t+1)} = \frac{1}{|S_i^{(t)}|} \sum_{\mathbf{x}_j \in S_i^{(t)}} \mathbf{x}_j$$

Repeat Assignment step and Update step until convergence

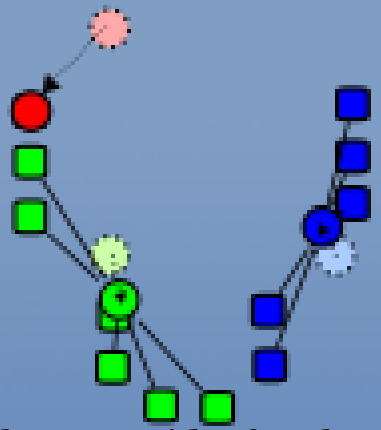
Problem Description - Example



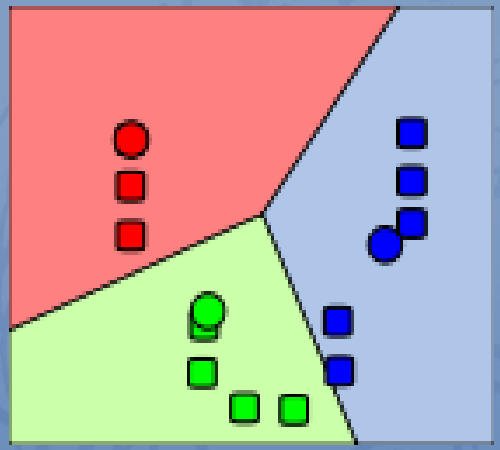
K points are randomly chosen and taken as means of K sets



K clusters are created by associating each point to the set with nearest mean



The centroid of each of the *k* clusters becomes the new mean



Repeat until convergence

Problem Description – K-means clustering convergence

When to stop ?

- A maximum number of iterations has been performed.
- Fewer than a threshold percentage of data points are reassigned during an iteration.
- All means migrate less than a threshold distance during an update cycle.

Implementation - MPI

Consider N data points each of it is vector and P processors.

(1) Assign N/P data points to each processor.

(2) Node 0 randomly choose K points and assigns them as cluster means and broadcast.

(3) In each processor for each data point find membership using the cluster mean.

(4) Recalculate local means for each cluster in each processor.

(5) Globally broadcast all local means for each processor find the global mean.

(6) Go to step (3) and repeat until the number of iterations > 10000 or number of points where membership has changed is less than 0.1 %.

Implementation-MPI-Example

K=2 ,P = 3 ,Dataset=[41,42,43,101,102,103]

{} = subset [] = membership () = cluster mean

Initialization

{41,42} [-1,-1] (41,42)	{43,101} [-1,-1] (41,42)	{102,103} [-1,-1] (41,42)

Implementation-MPI-Example

K=2 ,P = 3 ,Dataset=[41,42,43,101,102,103]

{} = subset [] = membership () = cluster mean

Initialization

{41,42} [-1,-1] (41,42)	{43,101} [-1,-1] (41,42)	{102,103} [-1,-1] (41,42)

Assignment Step

{41,42} [1,2] (41,42)	{43,101} [2,2] (41,42)	{102,103} [2,2] (41,42)

Implementation-MPI-Example

{} = subset [] = membership () = cluster mean

Update Centroid , Broadcast and find Global mean

{41,42} [1,2] (41,78.2)	{43,101} [2,2] (41, 78.2)	{102,103} [2,2] (41, 78.2)

Implementation-MPI-Example

{} = subset [] = membership () = cluster mean

Update Centroid , Broadcast and find Global mean

{41,42} [1,2] (41,78.2)	{43,101} [2,2] (41, 78.2)	{102,103} [2,2] (41, 78.2)

Assignment Step

{41,42} [1,1] (41,78.2)	{43,101} [1,2] (41, 78.2)	{102,103} [2,2] (41, 78.2)

Implementation-MPI-Example

{} = subset [] = membership () = cluster mean

Update Centroid , Broadcast and find Global mean

{41,42} [1,1] (42,102)	{43,101} [1,2] (41, 102)	{102,103} [2,2] (41, 102)

Implementation-MPI-Example

{} = subset [] = membership () = cluster mean

Update Centroid , Broadcast and find Global mean

{41,42} [1,1] (42,102)	{43,101} [1,2] (41, 102)	{102,103} [2,2] (41, 102)

Assignment Step

{41,42} [1,1] (42,102)	{43,101} [1,2] (41, 102)	{102,103} [2,2] (41, 102)

EXIT AFTER THIS POINT AS MEMBERSHIP DOESN'T CHANGE

Implementation - OpenMP

Consider N data points each of it vector and P threads.

(1) Node 0 randomly choose K points as assign them as cluster means.

(2) In each thread for each data point find membership using the cluster mean.

(3) Recalculate local means for each cluster in each thread.

(4) Globally broadcast all local means for each processor and find the global mean.

(5) Find the global mean in each thread

(6) Go to step (2) and repeat until the number of iterations >10000 or number of points where membership has is less than 0.1 %.

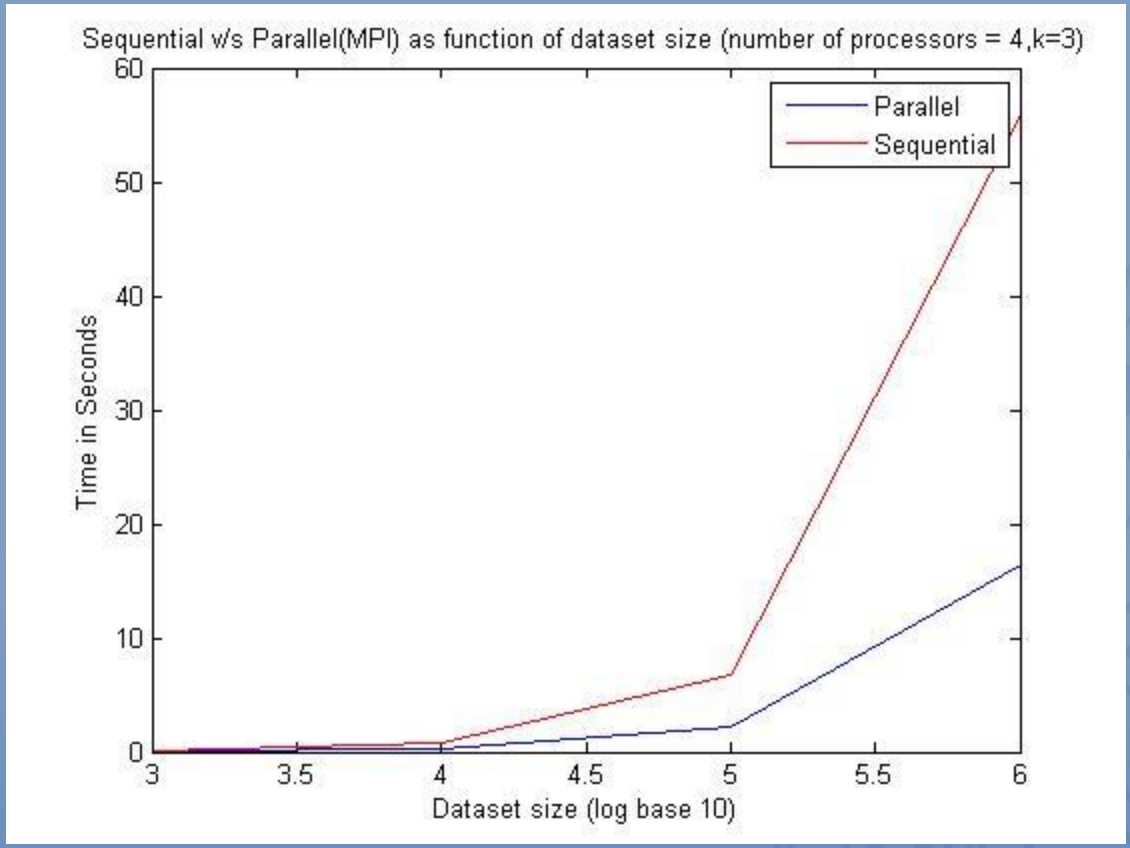
Test Results

- Each data point is 1X100 vector (which can be extended to any size 1XK without changing the code)
- Used 8 nodes of 8 cores for MPI and 1 node of 32 core for OpenMP
- Tried for cores 2,4,8,16,32 and 64 for MPI
- Tried for threads 2,4,8,16 and 32 for OpenMP

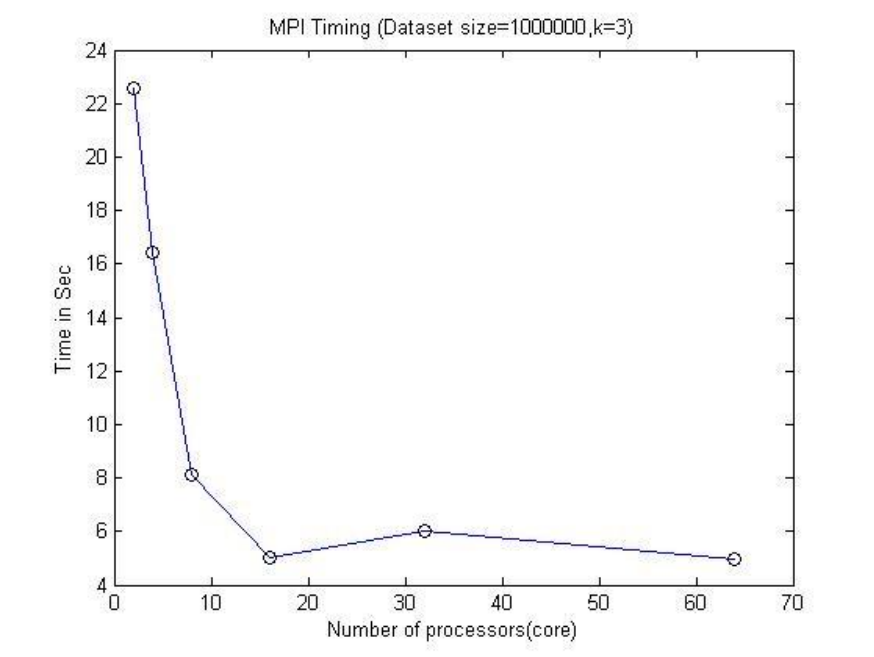
Test Results

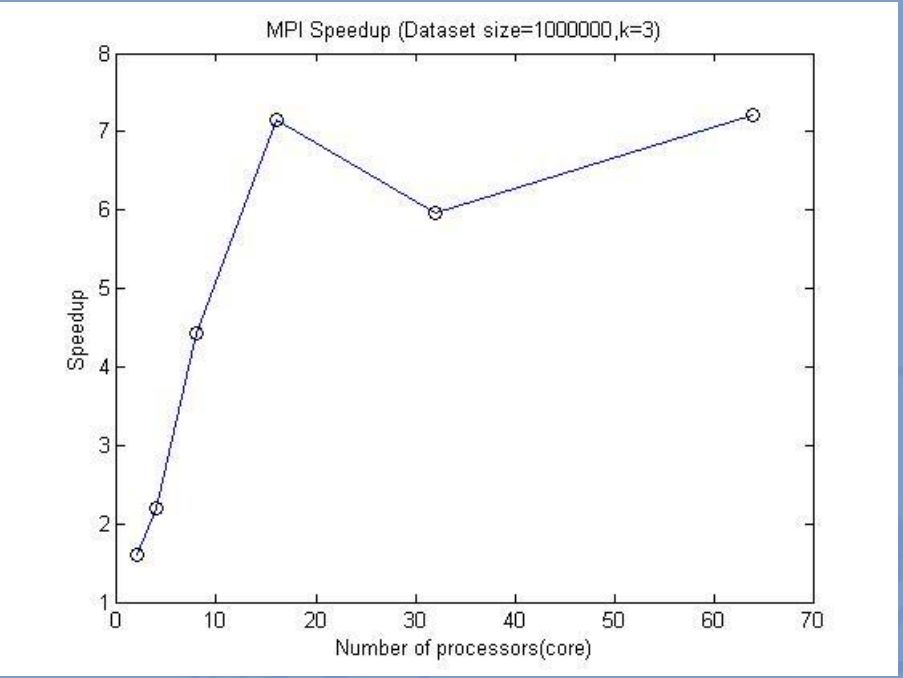
- Took an average of 10 runs
- The number of data points tested were 1000,10000,100000,1000000
- In MPI we start taking timings after the I/O operation of data placement into the nodes.
- I/O operation of data placement is much more than computational time.

Sequential V/s Parallel (MPI)

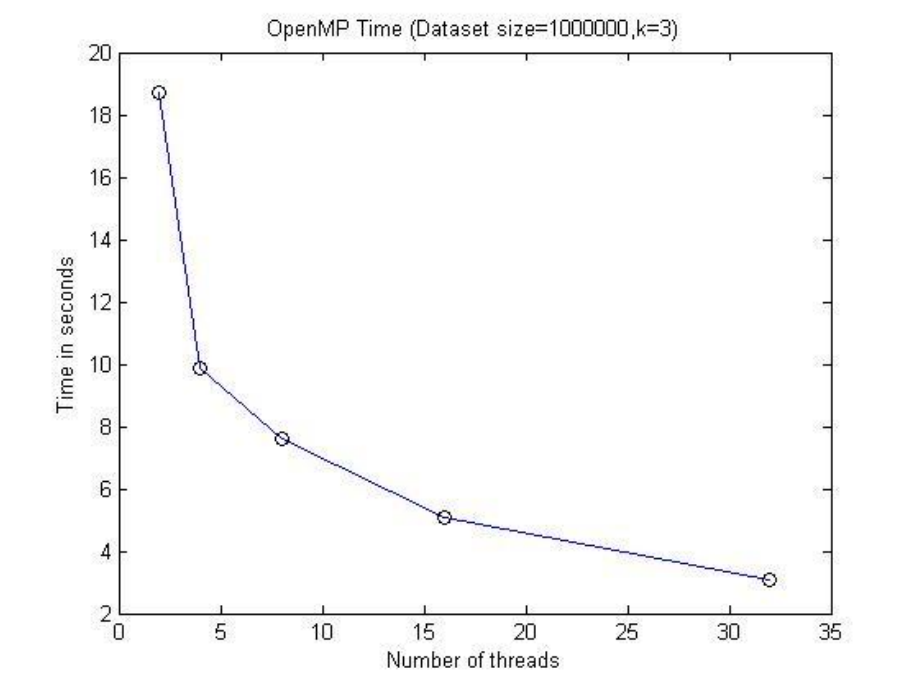


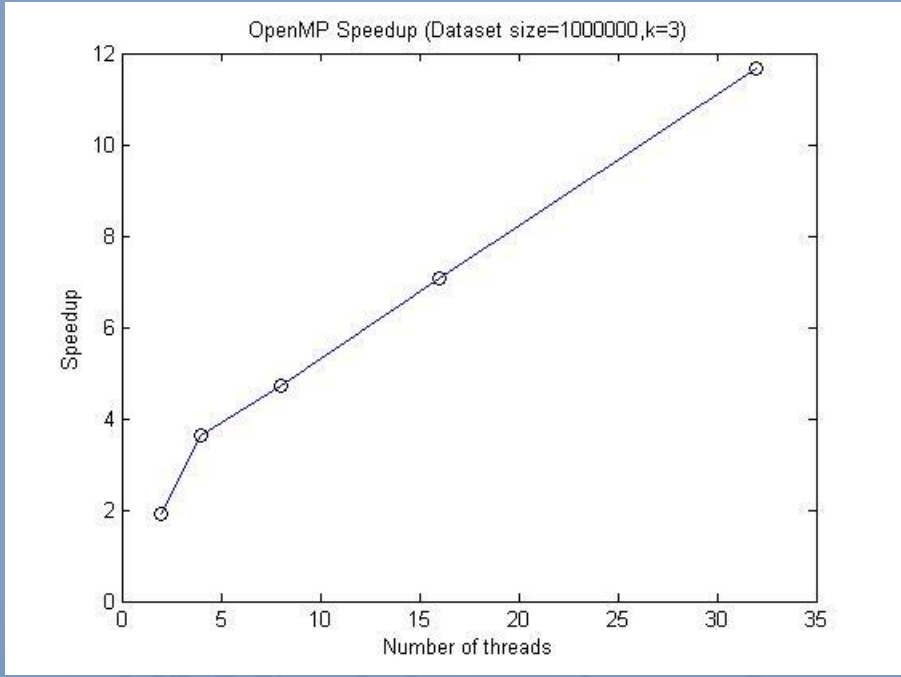
MPI Results


MPI Timing (Dataset size=1000000,k=3)
Time in Sec
Number of processors(core)
24
22
20
18
16
14
12
10
8
6
4
0
10
20
30
40
50
60
70

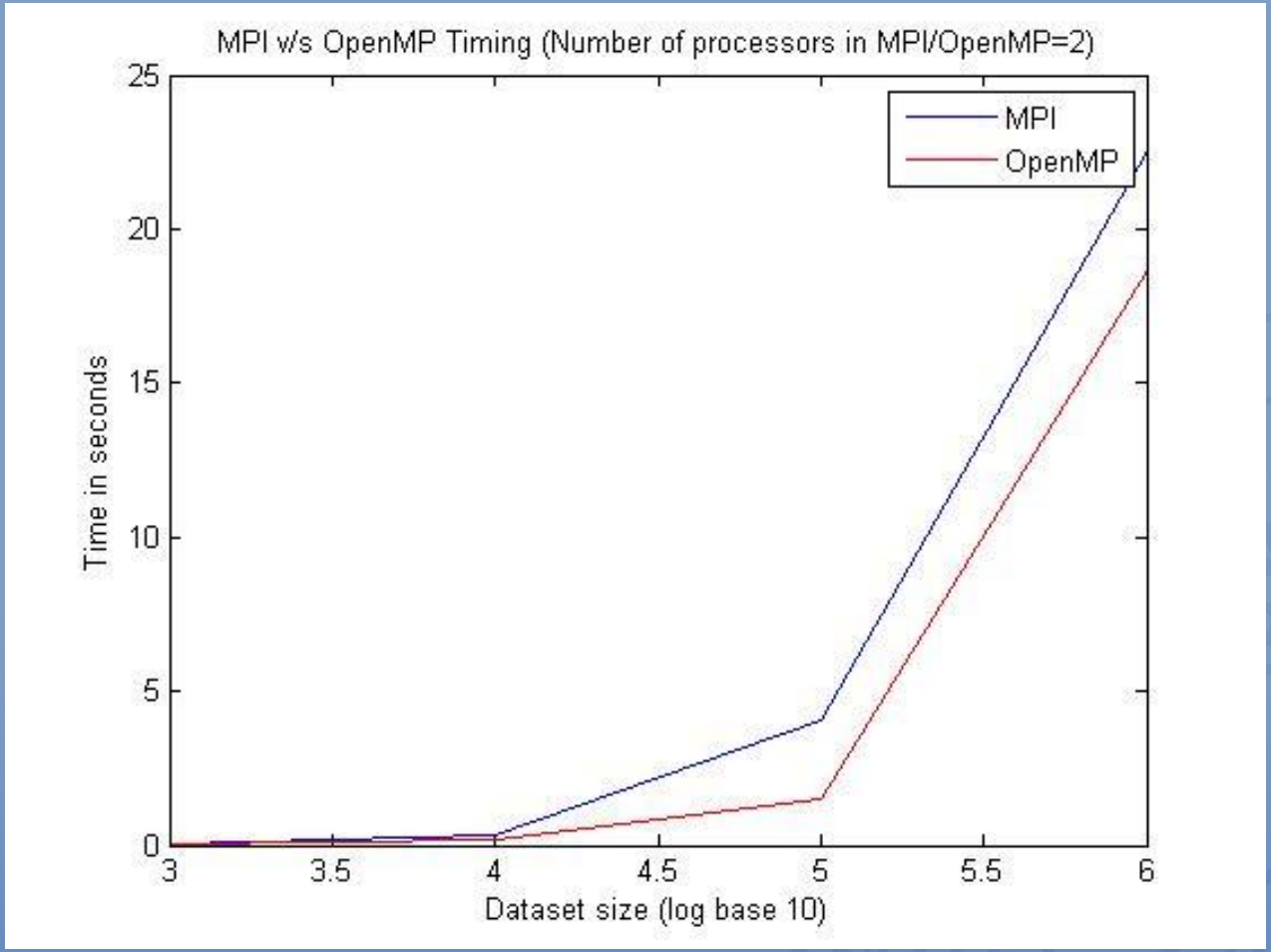

MPI Speedup (Dataset size=1000000,k=3)
Speedup
Number of processors(core)
8
7
6
5
4
3
2
1
0
10
20
30
40
50
60
70

OpenMP Results





MPI v/s OpenMP



OpenMP Time Test Readings (in sec)

Number of threads →

Number of Data points ↓

	2	4	8	16	32
1000	0.0118	0.0087	0.009	0.0066	0.0043
10000	0.1582	0.1234	0.08117	0.0698	0.0407
100000	1.4897	0.9337	0.7925	0.6038	0.3404
1000000	18.7079	9.8705	7.613	5.0962	3.0816

MPI Time Test Readings (in sec)

Number of processors →

Number of Data points ↓

	2	4	8	16	32	64
1000	0.03169	0.0178	0.01065	0.00789	0.01886	0.04289
10000	0.35529	0.21173	0.10899	0.05737	0.2346	0.20442
100000	4.06499	2.1171	1.02921	0.53855	0.89992	0.84488
1000000	22.57271	16.41537	8.12979	5.03628	6.03449	4.98651

Speedup with K

- K refers to number of clusters
- More iterations occur for convergence
- More iterations means more communication
- Speed up decreases as K increases

Conclusions

- K means clustering is not a problem that can be easily parallelizable as it requires frequent communication between each nodes.
- We do not get very great speedup in MPI due to communication overhead.
- OpenMP gives better performance than MPI

Future Work

- We can use a hybrid of OpenMP and MPI to improve the speed.
- The loop where each point is assigned membership to right cluster can be threaded in a particular node.
- Since it is an NP Hard problem we can use a lot of approximation techniques to reduce the number of iterations and increase speedup.

References

"Algorithms Sequential & Parallel: A Unified Approach" – Russ Miller and Lawrence Boxer

"Parallel k-Means Clustering for Quantitative Ecoregion Delineation using Large Data Sets" – Jitendra Kumara, Richard T. Millsa, Forrest M. Homana, William W. Hargroveb

"Parallel *k*/*h*-Means Clustering for Large Data Sets" - Kilian Stoel and Abdelkader Belkoniene

Questions/Comments