ISSN 1512-1232

Parallel Implementation of K-Means on Multi-Core Processors

Fahim Ahmed M.
Faculty of Science, Suez University, Suez, Egypt, ahmmedfahim@yahoo.com

Abstract

Nowadays, all most personal computers have multi-core processors. We try to exploit computational power from the multi-core architecture. We need a new design on existing algorithms and software. In this paper, we propose the parallelization of the well-known k-means clustering algorithm. We employ Parallel for-Loops (parfor) in MATLAB. Where a loop of n iterations could run on a cluster of m MATLAB workers simultaneously, each worker executes only n/m iterations of the loop. The experimental results demonstrate considerable speedup rate of the proposed parallel k-means clustering method run on a multicore/multiprocessor machine, compared to the serial k-means approach.

Keywords: *Clustering Algorithms, High Performance Computing, K-means Algorithm, and Parallel Computing.*

1. Introduction

The huge amount of data collected and stored in databases increases the need for effective analysis methods to use the information contained implicitly there. One of the primary data analysis tasks is cluster analysis, intended to help a user understand the natural grouping or structure in a dataset. Therefore, the development of improved clustering algorithms has received much attention. The goal of a clustering algorithm is to group the objects of a database into a set of meaningful subclasses [6].

Clustering is the process of partitioning or grouping a given set of patterns into disjoint clusters. This is done such that patterns in the same cluster are alike, and patterns belonging to two different clusters are different. Clustering has been a widely studied problem in a variety of application domains including data mining and knowledge discovery [4], data compression and vector quantization [7], pattern recognition and pattern classification [2], neural networks, artificial intelligence, and statistics. Many clustering algorithms proposed, but the most widely used one is the k-means method [13].

The popularity of k-means algorithm is due to its linear computational complexity that is O(*nkt*), where *n* is the number of data points or objects, *k* is the number of desired clusters, and *t* is the number of iterations the algorithm takes for converging to a stable state. The computing process needs improvements to efficiently apply the method to applications with huge number of data objects such as genome data analysis and geographical information systems.

Parallelization is one of the obvious solution to this problem and many researchers have proposed the idea many years ago such as [12] [8] [9] [22] [23] [15]. This paper also focuses on parallelizing k-means algorithm, but we base our study on the multi-core architecture. We implement our extension of the k-means algorithm using parallel MATLAB, where we use parfor loop, part of the `parfor` body is executed on the MATLAB client (where the `parfor` is issued) and part is executed in parallel on MATLAB workers working together as a *parallel pool*. The necessary data on which `parfor` operates is sent from the client to workers, where most of the computation happens, and the results are sent back to the client and pieced together. Because several MATLAB workers can be computing concurrently on the same loop, a `parfor`-loop can provide significantly better performance than its analogous `for`-loop [14].

We are interested in developing k-means algorithm because it is simple and widely used in practice. In addition, it is ideal algorithm for using parallel for in MATLAB. Our contributions in

ISSN 1512-1232
this paper is significantly reduce time complexity of the serial k-means algorithm by data parallelism. The rest of the paper is organized as follows. Discussion of related work in parallel k-means is presented in Section 2. Our proposed algorithm; a parallel k-means method is explained in Section 3. Some experimental results are demonstrated in Section 4. The conclusion appears as the last section of the paper.

2. Related Work

A serial k-means algorithm was proposed in 1967 [13] and since then it has gained great interest from data analysts and computer scientists. The algorithm has been applied to variety of applications ranging from medical informatics [10], genome analysis [15], image processing and segmentation [20, 21], to aspect mining in software design [1]. Despite its simplicity and great success, the k-means algorithm is known to degrade when the dataset grows larger in terms of number of objects and dimensions [8, 11]. To obtain acceptable computational speed on huge datasets, most researchers turn to parallelizing scheme.

Li and Fang [12] are among the pioneer groups on studying parallel clustering. They proposed a parallel algorithm on a single instruction multiple data (SIMD) architecture. Dhillon and Modha [3] proposed a distributed k-means that runs on a multiprocessor environment. Kantabutra and Couch [9] proposed a master-slave single program multiple data (SPMD) approach on a network of workstations to parallel the k-means algorithm. Their experimental results reveal that when on clustering four groups of two-dimensional data the speedup advantage can be obtained when the number of data is larger than 600,000, and their maximum speedup was 2.1, it is very small value. Tian and colleagues [19] proposed the method for initial cluster center selection and the design of parallel k-means algorithm. Stoffel and Belkoniene proposed parallel k-means works on a distributed database, the database was distributed over a network of 32 PCs, their results revealed that as the number of node increase the speedup degrades because of the increase of communication overhead and the variations in the execution times of the different processors [17].

Zhang and colleagues [23] presented the parallel k-means with dynamic load balance that used the master/slave model. Their method can gain speedup advantage at the two-dimensional data of size greater than 700,000. Prasad [16] parallelized the k-means algorithm on a distributed memory multi-processors using the message passing scheme. Farivar and colleagues [5] studied parallelism using the graphic coprocessors to reduce energy consumption of the main processor.

Zhao and colleagues [22] proposed parallel k-means method based on map and reduce functions to parallelize the computation across machines. Tirumala Rao and colleagues [18] studied memory mapping performance on multi-core processors of k-means algorithm. They conducted experiments on quad-core and dual-core shared memory architecture using OpenMP and POSIX threads. The speedup on parallel clustering is observable.

3. The Proposed Algorithm

Given a data set containing n objects, k-means partitions these objects into k groups. Each group is represented by the centroid, or central point, of the cluster. Once cluster means or representatives are selected, data objects are assigned to the nearest centers. The algorithm iteratively selects new better representatives and reassigns data objects until the stable condition has been reached. The stable condition can be observed from cluster assigning that each data object does not change its cluster. The serial k-means algorithm [13], shown in Figure 1, takes much computational time on calculating distances between each of n data points and the current k centers. Then iteratively assign each data point to the closest cluster. The proposed algorithm concentrates on distance calculation between each point and the k centers, performs these calculations in parallel way. If we have m cores and n data points then each core will approximately calculate the distances between n/m points and k centers. As m increase, the amount of calculation

ISSN 1512-1232
per each core will decrease. Since these calculations is iterated *t* times until each point stay in its cluster, this parallel paradigm significantly improves running time of the k-means algorithm.

Serial k-mean(*DS*,*k*)
1. Select initial *k* centers
2. Repeat
3. For *i*=1 to *n*
4. Calculates the distances between the current point and *k* centers
5. Endfor
6. Assign each point to its nearest cluster
7. Calculate the new *k* centers
8. Until stable *k* centers reached

Fig. 1: Serial K-means Algorithm.

Parallel k-mean(*DS*,*k*)
1. Select initial *k* centers
2. Repeat
3. parfor *i*=1 to *n*
4. Calculates the distances between the current point and *k* centers
5. Endfor
6. Assign each point to its nearest cluster
7. Calculate the new *k* centers
8. Until stable *k* centers reached

Fig. 2: Parallel K-means Algorithm.

Because the distance between point *i* and *k* centers is completely independent of the distance between point *j* and *k* centers, so these calculation can be performed in parallel using parallel for in MATLAB. To run code that contains a parallel loop, we first open MATLAB pool. This reserves a collection of MATLAB worker sessions to run loop iterations. The MATLAB pool can consist of MATLAB sessions running on local machine or on a remote cluster. Because the iterations run in parallel in other MATLAB sessions, each iteration must be completely independent of all other iterations. The worker calculating the distance between point *i* and *k* centers might not be the same worker calculating the distance between point *j* and *k* centers. There is no guarantee of sequence [14].

4. Experimental Results

We implemented the proposed algorithm (parallel k-means) and the serial k-means using MATLAB R2009b language. The code is executed on dell inspiron 1525 Laptop, Intel(R) Core(TM)2 Duo Processor 2.00 GHz, 2 MB cache memory, 2GB RAM, 32-bit Windows 7 Ultimate. We generated 19 synthetic two-dimensional datasets containing form 10000 to 1000000 data points with random values grouped into 100 cluster as shown in Figure 3.

ISSN 1512-1232

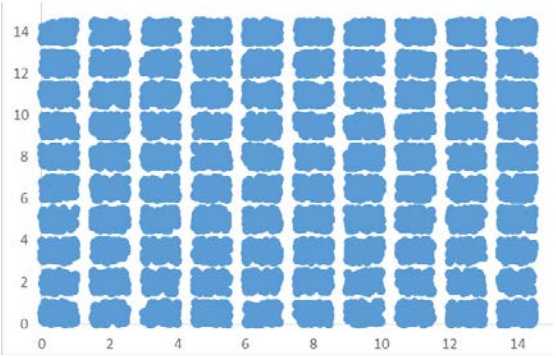

Fig.3: Synthetic Dataset Containing 100 Clusters.

We evaluate performances of the proposed algorithm on synthetic two-dimensional dataset, four and hundred clusters, run concurrently on two lab (session or core). The computational speed of parallel k-means as compared to serial k-means is given in Table1 where we set *k* to 100 cluster, in Table2 where we set *k* to four cluster, and finally in Table3 we set dataset size to 100000 point and *k* varied from 10 to 100 cluster. Ts refers to execution time of serial k-means, Tp refers to execution time of parallel k-means, and TD refers to time difference. The speedup is calculated according to the following equation.

$$speedup = \frac{Ts}{Tp}. \quad (1)$$

Linear speedup or ideal speedup is obtained when speed up is equal to number of processors (cores). In our case, linear speedup is equal to two.

Table 1: The Execution Time of Serial k-means versus Parallel k-means, *k*=100.

Dataset size	Ts	Tp	TD: Time Difference	Speedup
10000	10.631	6.431	4.2	1.65
20000	16.035	9.257	6.778	1.73
30000	31.900	18.161	13.739	1.76
40000	42.148	24.174	17.974	1.74
50000	52.989	29.917	23.072	1.77
60000	63.171	36.364	26.807	1.74
70000	74.130	42.603	31.527	1.74
80000	84.051	47.966	36.085	1.75
90000	95.062	55.008	40.054	1.73
100000	79.339	45.351	33.988	1.75
200000	219.693	127.674	92.019	1.72
300000	240.641	138.382	102.259	1.74
400000	424.670	252.474	172.196	1.68
500000	399.876	229.146	170.73	1.75
600000	637.507	Out of memory		
700000	738.981	Out of memory		
800000	634.420	Out of memory		
900000	951.472	Out of memory		
1000000	798.441	Out of memory		

If we compare the results shown in Table 1 with that of [9] and [23] we get better results from our proposed algorithm, because we get speedup form dataset of size 10000 object compared with dataset of size greater than 600000 and 700000 in [9] and [23] respectively. It is noticeable from Table 1 that at data size from 600000 points, running time of parallel k-means is unobservable

ISSN 1512-1232
because the machine is out of memory. This is due to the overhead of communication between cores requires some memory. It is also noticeable from Table 2 that running time of parallel k-means is a little bit longer than that of the serial k-means for the smallest dataset (10000 object). This is due to the overhead of communication between cores (labs, sessions, workers) requires some communication time.

Table 2: The Execution Time of Serial k-means versus Parallel k-means, *k*=4.

Dataset size	Ts	Tp	TD: Time Difference	Speedup
10000	1.044	1.051	-0.007	0.99
20000	2.005	1.718	0.287	1.17
30000	2.974	2.299	0.675	1.29
40000	3.991	2.880	1.111	1.39
50000	4.964	3.490	1.474	1.42
60000	6.651	4.548	2.103	1.46
70000	6.861	4.706	2.155	1.46
80000	7.867	5.327	2.54	1.48
90000	8.903	5.924	2.979	1.50
100000	9.914	6.539	3.375	1.52
200000	22.080	14.316	7.764	1.54
300000	29.423	18.780	10.643	1.57
400000	39.150	25.225	13.925	1.55
500000	48.788	31.113	17.675	1.57
600000	58.792	38.139	20.653	1.54
700000	68.459	43.508	24.951	1.57
800000	78.352	49.591	28.761	1.58
900000	87.921	55.677	32.244	1.58
1000000	97.795	62.205	35.59	1.57

Running time comparison of parallel against serial k-means is graphically shown in Figure 4 where *k*=100, and in Figure 5 where *k*=4. There is large variance in running time due to the large variance of required clusters. Figure 6 shows the behavior of running time as the number of cluster increase. This Figure is a little bit strange; this is appear because of large variance of k-mean iteration. Percentage of running time speedup is shown in Figure 7. Speedup average is very high. Figure 8 and Figure 9 graphically show the resulting clusters for *k*=4, and *k*=100 respectively.

Table 3: The Execution Time of Serial k-means versus Parallel k-means, Dataset Contains 100000 Point.

K	Ts	Tp	TD: Time Difference	Speedup
10	119.618	73.292	46.326	1.63
20	309.670	182.368	127.302	1.70
30	444.802	259.670	185.132	1.71
40	1094.742	696.940	397.802	1.57
50	1432.632	880.163	552.469	1.63
60	639.321	400.931	238.39	1.59
70	516.367	320.706	195.661	1.61
80	1668.612	1037.094	631.518	1.61
90	683.952	401.470	282.482	1.70
100	77.784	43.822	33.962	1.77

ISSN 1512-1232

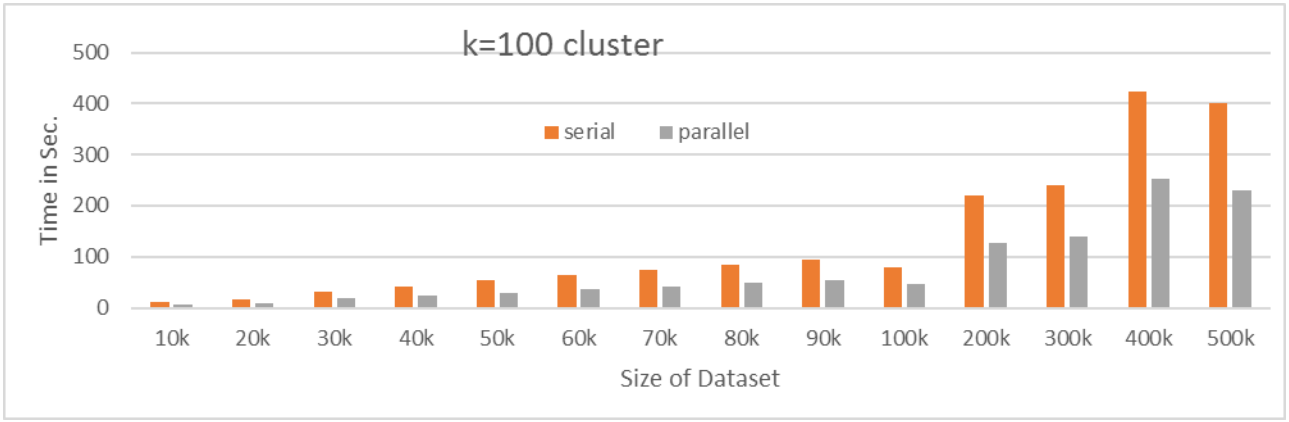

Fig. 4: Running Time Comparisons of Serial versus Parallel k-means, *k*=100 Clusters.

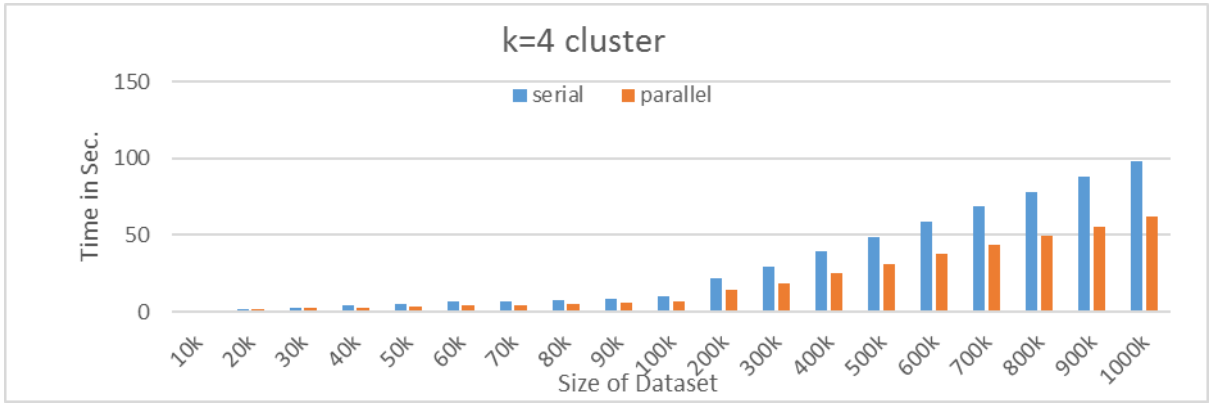

Fig. 5: Running Time Comparisons of Serial versus Parallel k-means, k=4 Clusters.

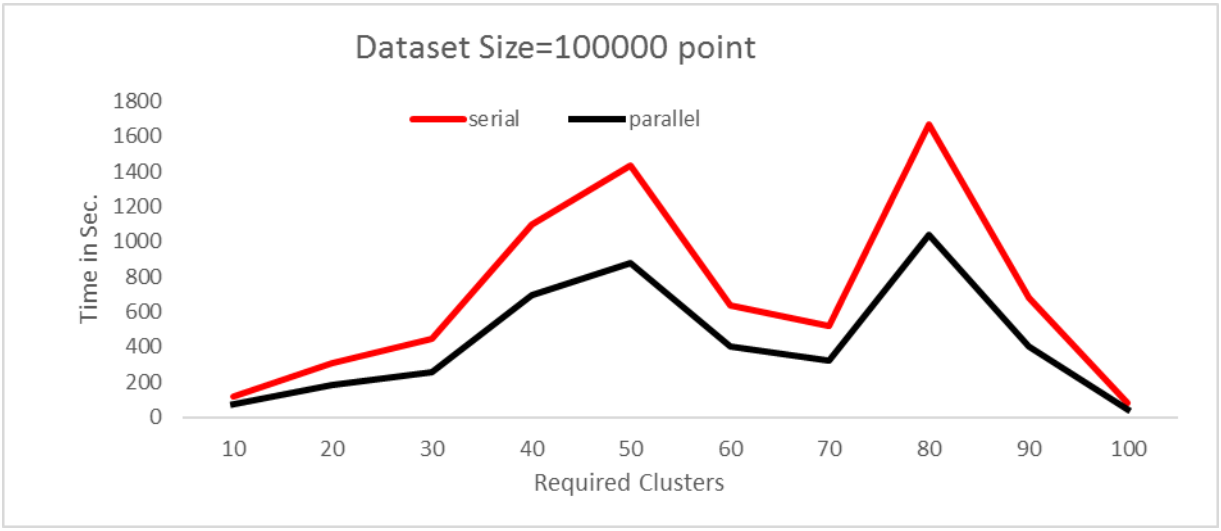

Fig. 6: Effect of Increasing Clusters on Running Time.

ISSN 1512-1232

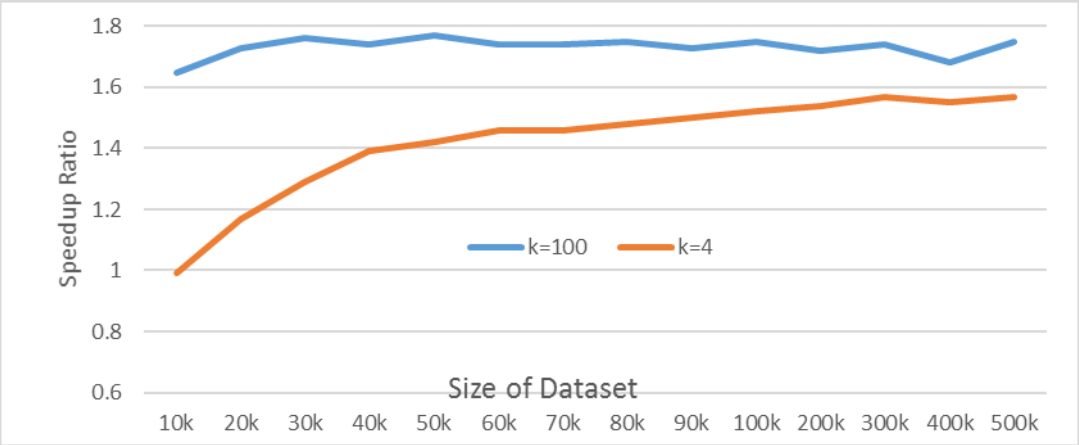

Fig. 7: Speedup Ratio at Different Sizes of Dataset.

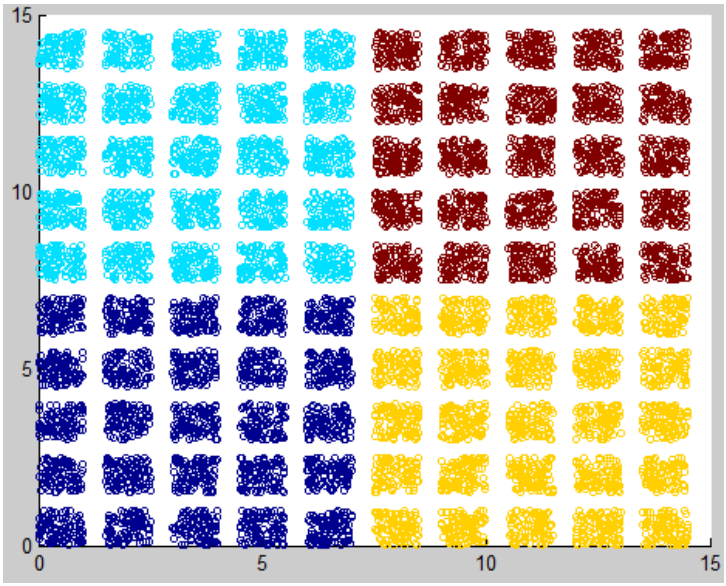

Fig. 8: Resulting Clusters for *k*=4.

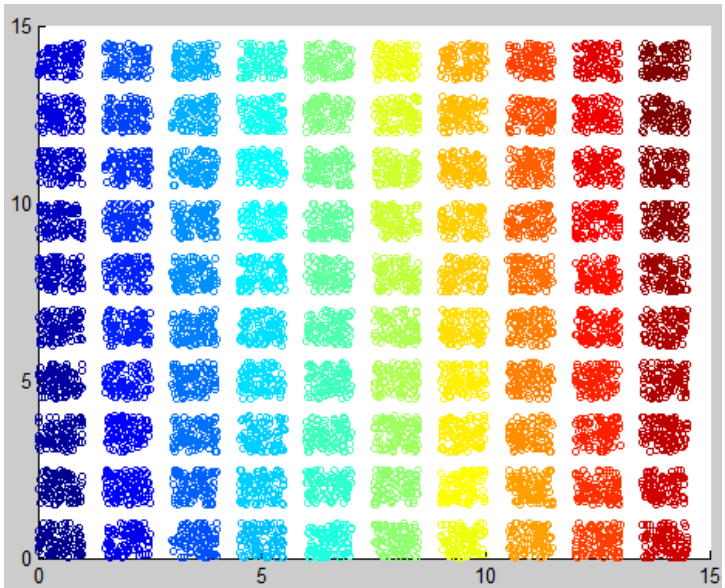

Fig. 9: Resulting Clusters for *k*=100.

ISSN 1512-1232

5. Conclusion

The k-means algorithm is simple but it performs intensive calculation on computing distances between data points and cluster central points. For the dataset with *n* data points and *k* clusters, each iteration of k-means requires as much as (*n*×*k*) computations.

Fortunately, the distance computation of one data point does not interfere the computation of other points. Therefore, k-means clustering is a good candidate for parallelism.

In this paper, we propose the design and implementation of parallel k-means algorithm: we paralyzed the k-means method by using parallel for that run distance computation concurrently on multi-cores machine. The parallel programming model used in our implementation is based on parallel MATLAB programming.

The experimental results reveal that the parallel method considerably speedups the computation time. Our future work will focus on the real applications. We will test our algorithm with a genome dataset.

6. References

1. Czibula G., Cojocar G., and Czibula I., "Identifying Crosscutting Concerns using Partitional Clustering", *WSEAS Transactions on Computers,* 2009, 8(2): pp. 386-395.
2. Duda R.O., Hart P.E., "Pattern Classification and Scene Analysis". John Wiley & Sons, New York. 1973
3. Dhillon and Modha D., "A Data-Clustering Algorithm on Distributed Memory Multiprocessors", Proceedings of ACM SIGKDD Workshop on LargeScale Parallel KDD Systems, 1999, pp. 47-56.
4. Fayyad U.M., Piatetsky-Shapiro G., Smyth P., Uthurusamy R., "Advances in Knowledge Discovery and Data Mining" AAAI/MIT Press, 1996.
5. Farivar R., Rebolledo D., Chan E., and Campbell R., "A Parallel Implementation of k-means Clustering on GPUs", Proceedings of International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA), 2008, pp. 340-345.
6. Fahim A. M., Salem A. M., Torkey F. A., Ramadan M. A., "An efficient enhanced k-means clustering algorithm". *Journal of Zhejiang University SCIENCE A*, 2006, 7(10): pp. 1626-1633.
7. Gersho A., Gray R. M., "Vector Quantization and Signal Compression". Kluwer Academic, Boston, 1992.
8. Joshi M., "Parallel k-means algorithm on distributed memory multiprocessors", Technical Report, University of Minnesota, 2003, pp. 1-12.
9. Kantabutra S. and Couch A., "Parallel k-means clustering algorithm on NOWs", *NECTEC Technical Journal*, 2000, 1(6): pp. 243-248.
10. Kerdprasop N. and Kerdprasop K., "Knowledge Induction from Medical Databases with Higher-order Programming", *WSEAS Transactions on Information Science and Applications*, 2009, 6(10): pp. 1719-1728.
11. Kerdprasop K., Kerdprasop N., and Sattayatham P., "Weighted k-means for density-biased clustering", Lecture Notes in Computer Science, Vol.3589: pp. 488-497, Data Warehousing and Knowledge Discovery (DaWaK), August 2005,.
12. Li X. and Fang Z., "Parallel clustering algorithms", *Parallel Computing*, 1989, 11(3): pp. 275-290.
13. MacQueen J., "Some methods for classification and analysis of multivariate observations", Proceedings of the 5th Berkeley Symposium on Mathematical Statistics and Probability, 1967, pp. 281-297.
14. MathWorks, Parallel Computing Toolbox™ User's Guide R2011b,[Online]. Available: http://www.mathworks.com/help/distcomp/getting-started-with-parfor.html?s_tid=doc_12b

ISSN 1512-1232

15. Othman F., Abdullah R., Abdul Rashid N., and Abdul Salam R., "Parallel k-means clustering algorithm on DNA dataset", Proceedings of the 5th International Conference on Parallel and Distributed Computing: Applications and Technologies (PDCAT), 2004, pp. 248-251.

16. Prasad, "Parallelization of k-means clustering algorithm", Project Report, University of Colorado, 2007, pp. 1-6.

17. Stoffel Kilian and Belkoniene Abdelkader, "Parallel k/h-Means Clustering for Large Data Sets", Euro-Par'99, LNCS 1685, 1999, pp. 1451-1454.

18. Tirumala Rao S., Prasad E., and Venkateswarlu N., "A critical performance study of memory mapping on multi-core processors: An experiment with k-means algorithm with large data mining data sets", *International Journal of Computer Applications*, 2010, 1(9): pp. 1–8.

19. Tian J., Zhu L., Zhang S., and Liu L., "Improvement and parallelism of k-means clustering algorithm", *Tsignhua Science and Technology*, 2005, 10(3): pp. 277-281.

20. Wang H., Zhao J. , Li H., and Wang J., "Parallel clustering algorithms for image processing on multicore CPUs", Proceedings of International Conference on Computer Science and Software Engineering (CSSE), 2008, pp. 450-53.

21. Ye Z., Mohamadian H., Pang S. , and Iyengar S., "Contrast enhancement and clustering segmentation of gray level images with quantitative information evaluation", *WSEAS Transactions on Information Science and Applications*, 2008, 5(2): pp. 181-188.

22. Zhao W., Ma H., and He Q., "Parallel k-means clustering based on MapReduce", Proceedings of the First International Conference on Cloud Computiong (CloudCom), 2009, pp. 674-679.

23. Zhang Y., Xiong Z., Mao J., and Ou L., "The study of parallel k-means algorithm", Proceedings of the 6th World Congress on Intelligent Control and Automation, 2006, pp. 5868-5871.

Article received: 2014-02-20