Mühendislik Bilimleri ve Tasarım Dergisi
8(3), 791 – 798, 2020
e-ISSN: 1308-6693

Journal of Engineering Sciences and Design
DOI: 10.21923/jesd.748209

Araştırma Makalesi Research Article

PARALLEL K-MEANS CLUSTERING WITH NAÏVE SHARDING FOR UNSUPERVISED IMAGE SEGMENTATION VIA MPI

Ahmet Esad TOP, F. Şükrü TORUN, Hilal KAYA*

Ankara Yildirim Beyazit University, Department of Computer Engineering, Ankara, Turkey

Keywords

Clustering, Image Segmentation, K-means Clustering, Distributed Memory, Parallel Computing.

Abstract

In digital image processing, image segmentation is an essential step in which an image is partitioned into groups of pixels. *k*-means clustering algorithm, which is often considered as fast and efficient, is one of the most widely used clustering algorithms to segment an image. However, as the problem size gets larger, the *k*-means starts to spend a significant amount of time to process. At this point, parallelization techniques should be applied to reduce the required time. Designing an efficient parallel and distributed model is not a trivial job since it should correspond to the parallel computer architecture and take communication and load balancing among processors into account. In this study, we propose a parallel and distributed *k*-means clustering algorithm with naïve sharding centroid initialization for image segmentation. The proposed algorithm adopts the Message Passing Interface (MPI) standard to take advantage of the computational power of distributed computing nodes in a High-Performance Computing Cluster. We demonstrate the parallel scalability of the proposed algorithm using up to 128 cores and it achieves 104.23 times faster clustering time.

DENETİMSİZ GÖRÜNTÜ BÖLÜTLEME İÇİN NAÏVE SHARDING İLE MPI KULLANARAK PARALEL K-MEANS KÜMELEMESİ

Anahtar Kelimeler

Kümeleme, Görüntü Bölütleme, K-means Kümeleme, Dağıtılmış Bellek, Paralel İşleme.

Öz

Dijital görüntü işlemede, görüntü bölütleme, görüntünün piksel gruplarına ayrıldığı önemli bir adımdır. Verimli bir kümeleme algoritması kabul edilen *k*-means algoritması, bir görüntüyü bölümlere ayırmak için en yaygın kullanılan kümeleme algoritmalarından birisidir. Bununla birlikte, problem boyutu büyüdükçe, *k*-means, görüntü işlemek için önemli miktarda zaman harcamaya başlar. Bu noktada, gerekli zamanı azaltmak için paralelleştirme teknikleri uygulanmalıdır. Verimli bir paralel ve dağıtılmış model tasarlamak, paralel bilgisayar mimarisini karşılayabilmesi ve işlemciler arasındaki iletişim ve yük dengelemesini dikkate alması nedeniyle önemli bir iştir. Bu çalışmada, görüntü bölütlemesi için naïve sharding kullanarak orta nokta belirleme ile paralel ve dağıtılmış bir *k*-means kümeleme algoritması öneriyoruz. Önerilen algoritma, Yüksek Performanslı Bilgi İşleme Kümesindeki dağıtılmış bilgi işleme düğümlerinin hesaplama gücünden yararlanmak için Mesaj Geçirme Arayüzü (MGA) standardını kullanır. 128 adede kadar çekirdek kullanarak 104.23 kat daha hızlı kümeleme süresi sağlayan önerilen algoritmanın paralel ölçeklenebilirliğini gösterdik.

Alıntı / Cite

Top, A.E., Torun, F.Ş., Kaya, H., (2020). Parallel K-means Clustering with Naïve Sharding for Unsupervised Image Segmentation via MPI, Journal of Engineering Sciences and Design, 8(3), 791-798.

Yazar Kimliği / Author ID (ORCID Number)

A. E. Top, 0000-0001-5017-1594
F. Ş. Torun, 0000-0002-6662-2502
H. Kaya, 0000-0003-4787-105X

Makale Süreci / Article Process	
Başvuru Tarihi / Submission Date	04.06.2020
Revizyon Tarihi / Revision Date	18.08.2020
Kabul Tarihi / Accepted Date	09.09.2020
Yayım Tarihi / Published Date	24.09.2020

* İlgili yazar / Corresponding author: hilalkaya@ybu.edu.tr, +90-312-906-2207

1. Introduction

Image segmentation is used in many application areas such as medical imaging (Forouzanfar et al., 2010), object detection (Delmerico et al., 2011), image recognition (Tu et al., 2005), and traffic control (Khan et al.2010). Image segmentation is a task to partition an image into multiple segments. This partitioning allows us to analyze images more easily or use them in an effective manner. When the task is finished, the resulting segments are set of pixels and all pixels in the same set share certain characteristics. One way to segment an image is to use clustering algorithms.

Clustering is a branch of machine learning, particularly a method of unsupervised learning, it splits the dataset into groups according to similarities. Among the unsupervised clustering algorithms, *k*-means is one of the most popular algorithms due to its simplicity and elegance. The *k*-means is an algorithm to cluster n objects based on attributes into k partitions, where $k < n$. Each cluster is represented by the center of the cluster. The algorithm converges to stable centroids of clusters.

k-means clustering algorithm for image segmentation is generally used to segment the interest area from the background. When dealing with image segmentation, there exists a serious concern about the running time of the application. Images need to be processed in a reasonable time in order to use them in any area more effectively and efficiently.

The classical *k*-means algorithm has 3 main steps (Jain et al., 1988). In the first step, the algorithm randomly chooses *k* data points from the input data as initial centroids which is the center of each cluster. In the second step, the algorithm assigns a centroid to all the data points that are the closest (i.e. the most similar) to the centroid. All points which are assigned to the same centroid form a cluster. In the third step, the algorithm updates centroids by computing the mean of all the examples in the cluster. Second and third steps are kept repeating until stable centroids are found. Therefore the computational complexity of one iteration of *k*-means algorithm is $k \times n \times t$, where n is the number of objects, t is the time of computing the Euclidean distance between 2 points and the asymptotic time complexity is $O(ikn)$, where i is the number of iterations required for the convergence.

While the classical *k*-means algorithm uses random centroid initialization, different techniques can be used as initial centroids directly affect the number of iterations needed to converge hence the clustering time. An effective alternative to random centroid initialization is the Naïve Sharding method (Mayo, 2016). It employs summation of attributes of each instance to sort all instances in the dataset. Then the dataset is divided into k equal pieces (i.e. shards). Finally, the initial centroids are chosen by getting the mean of instances in each shard. This method generally reduces the required iteration number and clustering time simultaneously, where its effect depends on the used dataset.

Image segmentation task is often computationally complex, especially when segmenting into a high number of clusters. Therefore, using an algorithm with low time complexity thus reducing the execution time is important. Although the classical *k*-means clustering algorithm is assumed fast when compared to other clustering algorithms, it also needs excessive time when the size of data is enormous. One intelligent way to reduce the execution time of the algorithm is to use parallel programming techniques that exploit the processing power of multiple processing units simultaneously. The biggest advantage of parallel programming is that it reduces the application solution time to a minimum time (Onder and Goksu, 2019).

There are two common methods in parallel computing; shared memory computing (i.e. multi-core) and distributed memory computing (i.e. cluster). Distributed memory computing needs explicit message passing to share data between distributed nodes. The distributed memory system can have hundreds or thousands of processing units by combining the power of distributed nodes. In this study, we adopt distributed memory parallelism which allows us to write parallel programs that can run on a large number of processing units. While implementing, message passing interface (MPI) library is used which provides a mechanism for synchronizing processes among distributed computing nodes.

In this study, a distributed and parallel *k*-means clustering algorithm to segment images based on colors is proposed and implemented in order to reduce the execution time of the *k*-means algorithm.

2. Literature Survey

The studies of (Olson, 1995) and (Rasmussen and Willett, 1989) were one of the first attempts to parallelize clustering algorithms. Parallel *k*-means clustering methods have been studied by many scholars (Dhillon and Modha, 2002; Xu and Zhang, 2004; Stoffel and Belkoniene, 1999; Zhao et al., 2009). On the other hand, in the

literature, there are very few studies (Kantabutra and Couch, 2000; Joshi, 2003; Zhang et al., 2011) which focus on parallelizing the *k*-means algorithm with MPI for massively parallel machines, such as high-performance computing (HPC) clusters.

The first parallel *k*-means clustering algorithm that adopts MPI is proposed in (Kantabutra and Couch, 2000). There is also a study, (Joshi, 2003), that used a heuristic approach to choose initial centroids and implemented parallel *k*-means clustering algorithm with MPI. In (Zhang et al., 2011), they presented a parallelized *k*-means algorithm, MKmeans, which focuses on how to efficiently handle huge volumes of data by using MPI. In (Kantabutra and Couch, 2000; Joshi, 2003; Zhang et al., 2011), they all use data parallelism with MPI. The implications from the results of these studies are in common. The studies show that if the problem size is small, communication overhead between processors becomes dominant and the parallel algorithm does not scale up well due to the sequential cost of calculation of initial centroids. Although the efficiency of the algorithms is improved on larger problems, the achieved speedup values remain very limited (i.e. the best result among these three studies is a speedup of 2.10).

Exploiting clustering techniques is one of the most widely used methods for image segmentation. When the subject is applying *k*-means for Image segmentation, there exist some sequential implementations (Ng et al., 2006; Dhanachandra et al., 2015), some multi-threaded applications (Bose et al., 2013; Wang et al., 2008; Bhimani et al., 2015) focused thread-level parallelism, and some graphics processing unit (GPU) parallelization implementations (Sirotkovic´ et al., 2012; Backer et al., 2013; Bhimani et al., 2015). To our knowledge, there exists only one study (Bhimani et al., 2015) that uses an MPI-based parallel *k*-means algorithm for image segmentation.

A multi-threaded *k*-means algorithm which utilizes only thread-level parallelism for image segmentation was proposed in (Bose et al., 2013). In their work, the image is divided into parts which are assigned into threads. Then the *k*-means clustering algorithm is used to segment them on each thread. With 6 threads, clustering the image into 6 clusters on 4000*4000 image is almost 2 times faster than the single-threaded version. Also in (Wang et al., 2008), they implemented a multi-threaded version of the *k*-means clustering algorithm to exploit the performance gain of multi-core central processing units (CPUs). They get a speedup of 3.89 with four processors.

In (Sirotkovic´ et al., 2012), they implemented a parallel *k*-means algorithm to segment images using GPU with compute unified device architecture (CUDA). Their results show that execution time is improved as between 2.3 and 600 times better than the sequential version. In (Backer et al., 2013), the GPU-based parallel *k*-means algorithm was proposed to segment images according to colors. When clustering is done, their algorithm assigns an average color to region features. Except for 64*64 image, their method beats the sequential CPU-based method for all other resolutions. Their GPU-based algorithm achieves approximately 8.6 times faster than the sequential CPU-based algorithm for 2048*2048 image.

The first study that compares different parallel approaches by implementing OpenMP, MPI, and GPU parallelization is (Bhimani et al., 2015). They also focused on evaluating different means initializations in parallel. They get the best speed-ups from OpenMP for small images and from GPU for larger images. Their approaches give approximately 30 times speed-up compared to a sequential implementation of *k*-means. In detail, they get 29.6 speed-up for OpenMP, 28.2 speed-up for MPI, 30.3 speed-up for CUDA implementation on the same 1164*1200 image.

3. Material and Method

This section presents the proposed parallel *k*-means algorithm that segments images based on red-green-blue (RGB) values, whose color intensities ranging from 0 to 255. Furthermore, the implementation details of the algorithm are also presented here.

In the *k*-means algorithm, the most computationally intensive part is defining the centroids which describe the clusters. At the beginning of the algorithm, the initial centroids are determined by the Naïve Sharding method and it is presented in Algorithm 2. The centroids are recalculated by using an objective function in each iteration. In this study, we used Euclidean distance (see Equation (1)) as an objective function. While there are many alternatives to the objective function, where it measures the distance between data points, we chose Euclidean distance due to our problem is related to geometrical separation rather than dependent on correlation. For geometrical separation, we could use another distance measurement technique like Manhattan distance, but we preferred to use Euclidean distance because it is generally chosen in similar studies and it is a default and most preferred distance function (Yildiz et al., 2019) of *k*-means algorithm. Furthermore, if two points are very close on most of the attributes but more diverse on one of them (e.g. in our problem, so similar in Red and Green, but so dissimilar in Blue color space), Euclidean distance will exaggerate that diversity while Manhattan distance is more

influenced by the closeness of other attributes. Our problem focuses on colors, where one color space can highly affect the actual color hence, it is so important to uncover and exaggerate the dissimilarity even only one color space is diverse.

$$d(x,y) = \sqrt{\sum_{i=1}^{m} (x_i - y_i)^2} \quad (1)$$

In the Introduction section, the principal steps of the classical *k*-means algorithm are presented. The main difference between the classical *k*-means and the *k*-means that is used for image segmentation is dealing with RGB values of pixels while running the algorithm (i.e. equivalent to 3-dimensional data in classical *k*-means). Furthermore, at the end of the algorithm, the color values of all pixels in the same cluster are adjusted as the centroid's color values, which shape the output image.

Algorithm 1. Parallel *k*-means Clustering for process p_i

Input: input image, number of clusters as *k*
Output: output image
Choose initial *k* centroids by using Naïve Sharding method
Divide and distribute pixels equally among all processes
while not converged **do**
  Broadcast all centroids from the process p_0
  Calculate Euclidean distance between each centroid *c* and each pixel
  Assign each pixel to the nearest cluster
  Reduce the sum of RGB values of all pixels in each cluster on p_0
  if $p_i = p_0$ **then** // Master process
    Update k centroids by averaging RGB values of the pixels in each cluster
    Check convergence
  end if
  Broadcast the convergence state from p_0
end while
Assign each pixel to the nearest cluster considering final centroids
Broadcast all centroids from p_0
for all *k* clusters do
  Set RGB values of all pixels in the cluster as the RGB values of the centroid c of the cluster
end for
Gather all resulting pixel values to p_0
if $p_i = p_0$ **then** // Master process
  Save output image
end if

The pseudocode of the parallel version of *k*-means algorithm that segments the image is shown in Algorithm 1. The parameter p_i indicates that the rank of the process with 'i'. In other words, p_i can be referred to as i^{th} process in the parallel system, similarly, p_0 indicates the process with rank '0' which is commonly called as the master process.

The initial centroids are selected by using the Naïve Sharding method. Naïve Sharding centroid initialization method (see Algorithm 2) sorts the pixels in ascending order according to their RGB value summations and dividing them into shards to choose centroids rather than selecting them randomly, which is the case in the traditional *k*-means algorithm. Here, we employed a parallel merge sort algorithm to achieve a more scalable clustering algorithm. When computations are finished, all pixels in the same cluster are set as the centroid of that cluster. Consequently, the resulting image has the same color for all pixels in the same cluster.

Algorithm 2. Choosing initial k centroids by using Naïve Sharding method with parallel Merge Sort for process p_i

Input: input image
Output: chosen *k* centroids
Divide and distribute pixels equally among all processes
Calculate sum of RGB values for all pixels
Sort pixels in ascending order according to sum of RGB values with parallel merge sort algorithm
if $p_i = p_0$ **then** // Master process
  Slice the dataset into *k* shards evenly
  Find *k* initial centroids by averaging RGB values of the pixels in each shard
  Return chosen *k* centroids
end if

4. Experimental Results

In experiments, we use two digital images, which are shown in Figure 1. Images in Figure 1a contains approximately 5 million pixels (2738*1826) while images in Figure 1b contains approximately 10 million pixels (3873*2582). Proposed algorithm is tested on RGB images with variant number of clusters for all experiments to be able to compare the results more reasonably. Obtained segmented images for different number of clusters can be seen in Figure 1.

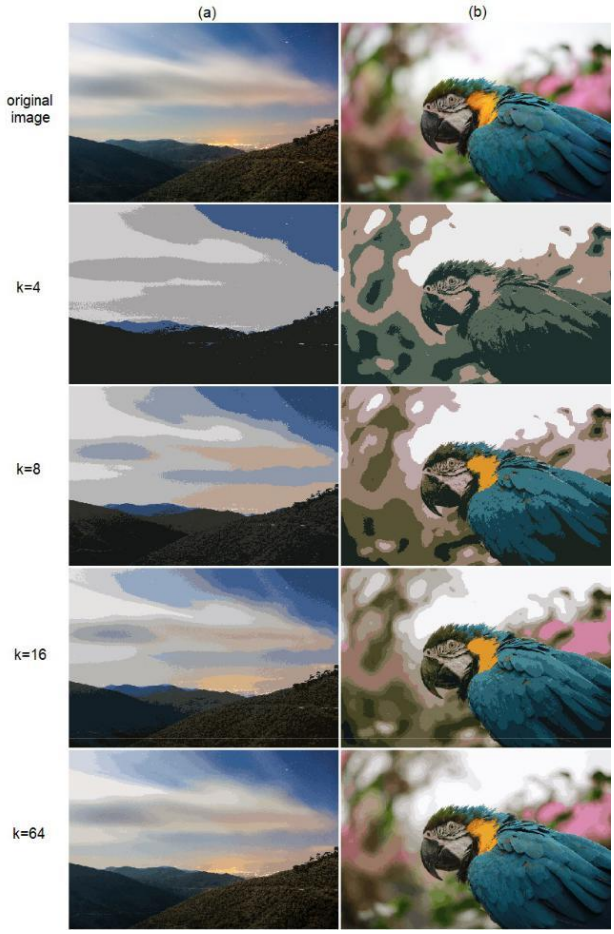

Figure 1. Output images of parallel *k*-means algorithm for the different number of clusters. The images on the left (a) and the images on the right (b) have approximately 5M and 10M pixels, respectively.

The experiments were conducted on Sardalya HPC Supercomputer in TUBITAK ULAKBIM High Performance and Grid Computing Center (TRUBA). Sardalya consists of 128 distributed nodes where each node contains two 14-core Intel Xeon E5-2690 processors and 256GB RAM.

In the strong scalability test, the problem size is fixed and the number of processes are varied. Figure 2 and 3 display the speedup values of the algorithm for 4 different clustering problems. The experiments were conducted on using 6 different number of cores and the speedup is calculated as follows:

$$speedup = \frac{t_{sequential}}{t_{parallel}} \quad (2)$$

where $t_{sequential}$ and $t_{parallel}$ are the sequential execution time on 1-core and parallel execution time on multiple processors, respectively.

As shown in Figure 2 and 3, when k is 64, the algorithm scales properly as the number of processes increases. Also, 10M image has higher speedup values than 5M image for each corresponded number of clusters and cores. However, when the number of cores is 96 or 128, the algorithm gives relatively lower speedup values for clustering the images into 4 or 8 clusters for both images and 16 clusters for 5M image. This experimental finding is expected due to the nature of the proposed parallel algorithm because when the number of clusters increases, the problem space, which is affecting the parallel portion of the algorithm, increases but the communication overhead remains almost the same. In other words, when the problem space becomes smaller, the communication overhead will be

more apparent which leads to low parallel performance. As a performance comparison of Figure 2 and 3, the same rule applies here too. The algorithm achieves better speedup values for the bigger image than the smaller one especially on relatively larger number of processes. As shown in Figure 3, on 128 cores, the proposed algorithm achieves more than 104.23 times faster execution than the sequential algorithm.

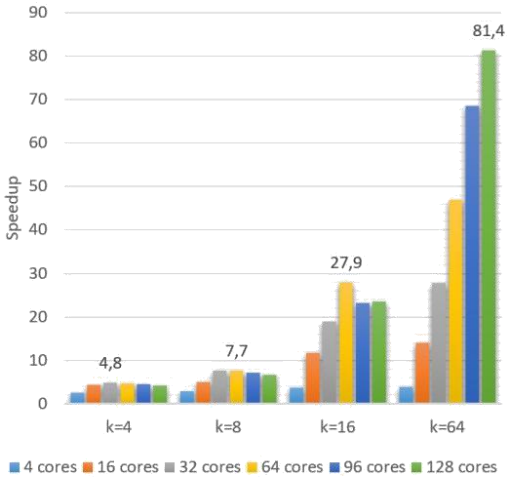

Figure 2. Speedup values for 5M image

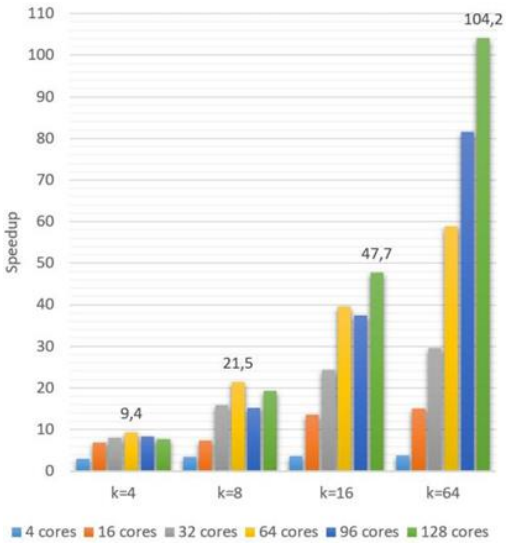

Figure 3. Speedup values for 10M image

Table 1 shows a comparison of the parallel speedup of the proposed algorithm against other studies in the literature. We note that (Wang et al., 2008) uses *k*-means algorithm to cluster 8965 e-mail messages into 3 different clusters in their provided results and (Backer et al., 2013) does not specify the number of clusters exactly however it is smaller than 1024 clusters and GPU-based parallelism is used. In the last column of Table 1, we depict the best speedup values reported by the studies. As seen in the table, the highest speedup is achieved by our proposed algorithm.

Table 1. Comparison of speedup values of our algorithm against (Bose et al., 2013; Wang et al., 2008; Backer et al., 2013; Bhimani et al., 2015)

Studies	*k*	Resolution	# Cores/Threads	Speedup
(Bose et al., 2013)	6	16,000,000	6	1.94
(Wang et al., 2008)	3	896,500	4	3.89
(Backer et al., 2013)	<1024	4,194,304	512 (CUDA)	8.66
(Bhimani et al., 2015)	240	1,396,800	160 (MPI)	28.23
(Bhimani et al., 2015)	240	1,396,800	2496 (CUDA)	30.26
Our alg.	64	10,000,086	128	104.23

5. Result and Discussion

We show the effectiveness and scalability of the proposed parallel *k*-means clustering algorithm that uses naïve sharding centroid initialization for 8 different image segmentation problems on distributed memory machines. The proposed parallel algorithm has *O(ikn/p)* average-case asymptotic time complexity where p is the number of processors, *n* is the number of objects, *k* is the number of clusters and *i* is the number of iterations required for the convergence. Worst-case complexity of the algorithm has not changed as the changes made to adapt the algorithm to MPI are constant-time operations. The proposed algorithm scales well when the problem size and the number of processors is large. It achieves a speedup value of 104.23 on 128 cores with respect to the sequential version. Our study outperforms other studies in the literature to the best of our knowledge in terms of speedup values. Table 1 compares those studies against our algorithm. Multi-threaded implementations (Bose et al., 2013; Wang et al., 2008; Bhimani et al., 2015) (i.e. the best one is (Bhimani et al., 2015) with 29.6 speedup), which use OpenMP, attain limited speedup values compared to our study. The only study that uses MPI (Bhimani et al., 2015) obtains lower speedup values compared to ours.

As future work, we consider designing a different parallel image segmentation algorithm, which may be using Fuzzy C-means clustering, Mean-shift algorithm, or Watershed segmentation algorithm.

Acknowledgement

The numerical calculations reported in this paper were performed at TUBITAK ULAKBIM, High Performance and Grid Computing Center (TRUBA resources).

Conflict of interest

No conflict of interest was declared by the authors.

References

Backer, M., Tünnermann, J., Mertsching, B., 2013. Parallel k-means image segmentation using sort, scan and connected components on a gpu. Springer, Facing the multicore-challenge III, 108–120.

Bhimani, J., Leeser, M., Mi, N., 2015. Accelerating k-means clustering with parallel implementations and gpu computing. IEEE, 2015 IEEE High Performance Extreme Computing Conference (HPEC), 1–6.

Bose, S., Mukherjee, A., Chakraborty, S., Samanta, S., Dey, N., et al., 2013. Parallel image segmentation using multi-threading and k-means algorithm. IEEE, 2013 IEEE International Conference on Computational Intelligence and Computing Research, 1–5.

Delmerico, J. A., David, P., Corso, J. J., 2011. Building facade detection, segmentation, and parameter estimation for mobile robot localization and guidance. IEEE, 2011 IEEE/RSJ International Conference on Intelligent Robots and Systems, 1632–1639.

Dhanachandra, N., Manglem, K., Chanu, Y. J., 2015. Image segmentation using k-means clustering algorithm and subtractive clustering algorithm. Procedia Computer Science, 54, 764–771.

Dhillon, I. S., Modha, D. S., 2002. A data-clustering algorithm on distributed memory multiprocessors. Springer, Large-scale parallel data mining, 245–260.

Forouzanfar, M., Forghani, N., Teshnehlab, M., 2010. Parameter optimization of improved fuzzy c-means clustering algorithm for brain mr image segmentation. Engineering Applications of Artificial Intelligence, 23(2), 160–168.

Jain, A. K., Dubes, R. C., et al., 1988. Algorithms for clustering data, volume 6. Prentice hall Englewood Cliffs.

Joshi, M. N., 2003. Parallel k-means algorithm on distributed memory multiprocessors. Computer, 9.

Kantabutra, S., Couch, A. L., 2000. Parallel k-means clustering algorithm on nows. NECTEC Technical journal, 1(6), 243–247.

Khan, J. F., Bhuiyan, S. M., Adhami, R. R., 2010. Image segmentation and shape analysis for road-sign detection. IEEE Transactions on Intelligent Transportation Systems, 12(1), 83–96.

Mayo, M. M., 2016. An arithmetic-based deterministic centroid initialization method for the k-means clustering algorithm. Columbus State University ePress.

Ng, H., Ong, S., Foong, K., Goh, P., Nowinski, W., 2006. Medical image segmentation using k-means clustering and improved watershed algorithm. IEEE, 2006 IEEE Southwest Symposium on Image Analysis and Interpretation, 61–65.

Olson, C. F., 1995. Parallel algorithms for hierarchical clustering. Parallel computing, 21(8), 1313–1325.

Onder, A. S., Goksu, T., 2019. Real-time natural stone classification with opencl and performance analysis. Muhendislik Bilimleri ve Tasarim Dergisi, 7, 689 – 698.

Rasmussen, E. M., Willett, P., 1989. Efficiency of hierarchic agglomerative clustering using the icl distributed array processor. Journal of Documentation, 45(1), 1–24.

Sirotkovic,´ J., Dujmic,´ H., and Papic,´ V. (2012). K-means image segmentation on massively parallel gpu architecture. IEEE, 2012 Proceedings of the 35th International Convention MIPRO, 489–494.

Stoffel, K., Belkoniene, A., 1999. Parallel k/h-means clustering for large data sets. Springer, European Conference on Parallel Processing, 1451–1454.

Tu, Z., Chen, X., Yuille, A. L., Zhu, S.-C., 2005. Image parsing: Unifying segmentation, detection, and recognition. International Journal of computer vision, 63(2), 113–140.

Wang, H., Zhao, J., Li, H., Wang, J., 2008. Parallel clustering algorithms for image processing on multi-core cpus. IEEE, 2008 International Conference on Computer Science and Software Engineering, volume 3, 450–453.

Xu, S., Zhang, J., 2004. A parallel hybrid web document clustering algorithm and its performance study. The Journal of Supercomputing, 30(2), 117–131.

Yildiz, M. S., Kekezoglu, B., Isen, E., 2019. Determination of maintenance strategy for power transformers with k-means clustering method. Muhendislik Bilimleri ve Tasarim Dergisi, 7, 505 – 513.

Zhang, J., Wu, G., Hu, X., Li, S., Hao, S., 2011. A parallel k-means clustering algorithm with mpi. IEEE, 2011 Fourth International Symposium on Parallel Architectures, Algorithms and Programming, 60–64.

Zhao, W., Ma, H., He, Q., 2009. Parallel k-means clustering based on mapreduce. Springer, IEEE International Conference on Cloud Computing, 674–679.