Knowledge Engineering and Data Science (KEDS)
Vol 3, No 1, July 2020, pp. 40–49

pISSN 2597-4602
eISSN 2597-4637

Parallelization of Partitioning Around Medoids (PAM) in K-Medoids Clustering on GPU

Adhi Prahara [a, 1, *], Dewi Pramudi Ismi [a, 2], Ahmad Azhari [a, 3]

[a] *Informatics Department, Faculty of Industrial Technology, Universitas Ahmad Dahlan Jalan Ring Road Selatan, Tamanan, Banguntapan, Bantul Yogyakarta 55166, Indonesia*

[1] *adhi.prahara@tif.uad.ac.id *; [2] dewi.ismi@tif.uad.ac.id; [3] ahmad.azhari@tif.uad.ac.id*
** corresponding author*

ARTICLE INFO

Article history:
Received 25 June 2020
Revised 15 July 2020
Accepted 9 August 2020
Published online 17 August 2020

Keywords:
k-medoids
PAM
parallel computing
CUDA
GPU

ABSTRACT

K-medoids clustering is categorized as partitional clustering. K-medoids offers better result when dealing with outliers and arbitrary distance metric also in the situation when the mean or median does not exist within data. However, k-medoids suffers a high computational complexity. Partitioning Around Medoids (PAM) has been developed to improve k-medoids clustering, consists of build and swap steps and uses the entire dataset to find the best potential medoids. Thus, PAM produces better medoids than other algorithms. This research proposes the parallelization of PAM in k-medoids clustering on GPU to reduce computational time at the swap step of PAM. The parallelization scheme utilizes shared memory, reduction algorithm, and optimization of the thread block configuration to maximize the occupancy. Based on the experiment result, the proposed parallelized PAM k-medoids is faster than CPU and Matlab implementation and efficient for large dataset.

I. Introduction

Clustering is the task of assigning unlabeled data points into a finite number of clusters. The assignment is usually based on similarity or distance, so data points located in the same cluster are similar to each other. Clustering techniques have been implemented and play important roles in a wide range of application domains, such as image segmentation [1][2][3][4], image clustering [5], bioinformatics [6][7] and data mining [3][8]. Through clustering, the underlying patterns of the data can be revealed. Clustering is unsupervised learning that gives information based on the intrinsic properties of the data when no labels are assigned to the data.

The most widely studied clustering algorithms are partitional clustering and hierarchical clustering [9]. Partitional clustering, such as k-means and k-medoids is the most widely used in practice. Partitional clustering divides the dataset into a number of partitions. The number of partitions must be smaller than the number of data points in the dataset. K-means clustering is simpler than k-medoids clustering, but its main drawback is the sensitivity to outliers. K-medoids clustering offers better result when dealing with outliers [10] and arbitrary distance metrics also in the situation when the mean or median does not have a clear definition. However, k-medoids clustering suffers high computational complexity.

Due to the high computational complexity, the efficiency of k-medoids clustering becomes a major concern in the k-medoids algorithm improvement. Researchers have been working on attempts to improve the performance of k-medoids clustering [11][12]. In general, the efforts on improving k-medoids clustering focus on three different approaches [13] such as 1) empowering the local search and global search for medoids selection, 2) the number of data to be used for the medoids calculation: use the entire data (PAM: Partitioning Around Medoids) algorithm or just a sample of the data (CLARA: Clustering Large Application) algorithm, and 3) the computation method: serial or parallel.

There have been studies conducted by researchers to implement the clustering method using parallel computing approaches [13][14][15][16][17], especially k-medoids clustering, that will be the focus of this research. One of the technologies that used to develop parallel k-medoids clustering is Hadoop-MapReduce [18][19][20][21][22][23]. MapReduce consists of map function and reduces function. Map function computes the distances between each data, and the medoids then assign the data to their clusters. Reduce function checks the results of Map function then search for new medoids. This function will return the results to the Map function in the next MapReduce process. In [14], the optimal search of medoids is performed based on the basic properties of triangular geometry. The speed of k-medoids clustering is improved when the validity of the clustering result is maintained [18]. Parallel k-medoids clustering can also be implemented on Graphics Processing Unit (GPU). Several GPU accelerated researchers have developed k-medoids clustering: parallel PAM implementation using CUDA [24][25], GPU based parallel k-medoids (combined PAM-CLARA) clustering for remote sensing data [26], and GPU accelerated parallel clustering algorithms including k-means clustering, k-medoids clustering, and hierarchical clustering [27].

This work focuses on the development of parallel k-medoids clustering to increase the efficiency of partitioning around medoids (PAM) in k-medoids clustering. Our main contribution is to speed up the computation of the PAM algorithm using a parallelization scheme on GPU. The proposed parallelization scheme can handle large dataset with n number of data without creating $n{\times}n$ table of distance that consume huge memory but still maintain the computation speed. This paper is organized as follows: Section 2 presents the proposed parallelized PAM in k-medoids clustering, Section 3 presents the results and discussion, and the conclusion of this work is described in Section 4.

II. Method

A. K-Medoids Clustering

K-medoids clustering is a partitional clustering method and is similar to k-means clustering because the goal of both methods is to divide a set of measurements or observations into k subsets or clusters so that the subsets minimize the sum of distances between a measurement and a center of the measurement's cluster. Unlike k-means clustering, which minimizes the distance between data points within a cluster with the mean value of those data points, which called centroid, k-medoids clustering attempts to minimize the distance between data points within a cluster with a representative data point in the same cluster. This representative data point has a minimum total dissimilarities/distances to the other data points in the same cluster. This representative data point is called the medoid. Each generated cluster in k-medoids clustering has a representative data point (medoid). Thus, k-medoids clustering guarantees that the center of a cluster is the most centrally located data point.

Medoids are initialized by selecting k data points arbitrarily. K-medoids clustering iterates until the objective function returns minimum value. The objective function / absolute-error criterion (AEC), E is defined in (1).

$$E = \sum_{j=1}^{k} \sum_{p \in c_j} |p - o_j| \tag{1}$$

where p is the data point in the cluster c_j, o_j is the medoid of c_j.

K-medoids function involves several iterative algorithms that minimize the sum of distances from each object to its cluster's medoid, over all of the clusters. A well-known implementation of k-medoids clustering is the Partitioning Around Medoids (PAM) algorithm, which was developed by [28]. It takes two inputs: the number of clusters to generate (known as k) and a dataset D, which contains the data points. It generates k different clusters. The detail method of PAM is:

1) Build Step. Select k data points arbitrarily from the dataset D as the initial medoids.

2) Swap Step. Repeat
   a) Assign each non-medoid data point to the cluster, which has the most similar/nearest medoid.
   b) Randomly select a non-medoid data point.
   c) Compute the total cost of swapping that is the difference between the AEC calculated using the current medoid data point and the AEC calculated using the non-medoid data point selected in step b.

d) If the AEC calculated using a non-medoid data point is lower than the AEC calculated using the medoid data point, swap the medoid to that selected non-medoid data point. Then, the non-medoid data point is set to be the new medoid of the cluster.

3) Until no more change on clusters.

K-medoids clustering is better than k-means clustering when dealing with outliers [10]. K-medoids is the opposite of k-means clustering, that is sensitive to outliers. The presence of outliers does not influence medoids. K-medoids also useful for categorical clustering data where the mean of the data does not exist within the dataset. However, k-medoids is costlier than k-means clustering due to the iteration that examines every data point.

The computational complexity of k-medoids clustering is $O(k(n-k)^2)$ where k is the number of clusters, and n is the number of data [29]. For a large value of n and k, the computation is very costly. For a large number of data, the computation time will increase quickly as the number of data grows. Thus, k-medoids clustering is only suitable for small data and suffers inefficiency for big data.

B. CUDA Parallel Computing

GPU (Graphics Processing Unit) is a high-level parallel architecture used to do a fast operation in computer graphics, but now it can be used to perform computation other than graphics, which known as GP-GPU (General Purpose-Graphics Processing Unit) [30]. The well-known general-purpose parallel computing platform and programming model is Compute Unified Device Architecture (CUDA) from NVidia. GPU is highly parallel, multithreaded, has many-core processor and very high memory bandwidth. The difference between how CPU and GPU process the data is shown in Fig. 1 (a) and (b). GPU devotes more transistor to data processing rather than data caching and flow control. GPU is built on array of Streaming Multiprocessors (SM) and it is organized into grids, blocks, and threads. Data-parallel processing maps data elements to parallel processing threads. Fig. 1 (c) shows the parallel processing threads in GPU. A multithreaded program is partitioned into blocks of threads that execute independently from each other.

III. Results and Discussions

The proposed parallelized k-medoids are written using CUDA based on Matlab's PAM implementation [31][32], and runs on Core-i7 7700K processor, 16 GB of RAM, and NVidia GTX

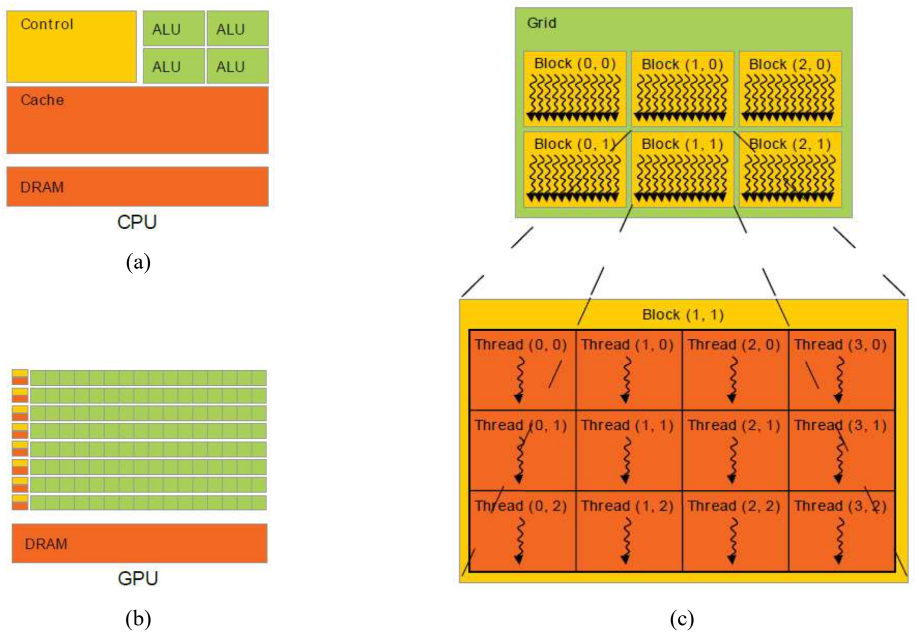


Fig. 1. GPU devotes more transistors to data processing; (a) CPU configuration; (b) GPU configuration; and (c) grid of threadblocks in GPU [30]

1070. The parallel implementation and performance evaluation is explained in the following subsections.

A. Parallelized K-Medoids Clustering

The implementation of parallelized PAM k-medoids is shown in Algorithm 1. It consists of three kernels that compute the gain of medoids to each data within a cluster, computes the gain of non-medoids to each data, and computes the new medoids. In algorithm 1, the data and initial cluster medoids that randomly picked from data are copied from host to device. The algorithm computes new medoid per cluster and iterates until no medoid is changed. The data partition in the swap step is used to reduce the number of data processed by thread blocks.

Algorithm 1. Proposed parallelized PAM k-medoids clustering.

```
READ parameters of k-medoids and GPU configuration

SET random data as initial cluster medoids
COPY data and initial medoids from host to device
WHILE cluster medoids changed DO
  FOR EACH cluster DO
    compute gain of medoid to each data within cluster
    FOR n = 0 TO number of data partition DO
      compute gain of non-medoids to each data …
        in n-th partition
    END
    compute new medoid
  END
END
COPY cluster medoids and labels from device to host
```

Algorithm 2. Compute the gain of medoids to each data within the cluster.

```
GPU CONFIGURATION
  blocks ← 256
  grids ← (number of data + blocks – 1)/ blocks

gain medoids ← 0

READ  data,
      cluster medoids,
      cluster labels,
      current cluster index,
      number of data,
      number of cluster

ALLOCATE shared memory (smem) to store
  smem cluster medoids ← cluster medoids
  smem gain ← 0

i ← index of data
IF i < number of data THEN
  min distance index ← 0
  min distance ← maximum value of type data
  FOR k = 0 TO number of cluster DO
    distance ← compute distance between …
      i-th data and k-th smem cluster medoids
    IF distance < min distance THEN
      min distance index ← k
      min distance ← distance
    END
  END
  i-th cluster labels ← min distance index

  IF min distance index = current cluster index THEN
    ATOMICADD(smem gain, min distance)
  END
END
SYNCHRONIZE the threads

IF threadIdx.x = 0 THEN
  gain medoids ← smem gain
END
```

Algorithm 3. Compute the gain of non-medoids to each data.

```
GPU CONFIGURATION
  blocks ← 256
  grids ← number of data after data partition

READ  data,
      cluster medoids,
      cluster labels,
      gain of potential medoids,
      current cluster index,
      number of data,
      number of cluster

ALLOCATE shared memory (smem) to store
  smem cluster medoids ← cluster medoids
  smem gain int ← 0
  smem gain ext ← 0

i ← index of data after data partition
i-th gain of non medoids ← 0
IF i is not medoid THEN
  gain int ← 0
  gain ext ← 0
  j ← index of data
  IF j < number of data THEN
    distance ← compute distance between …
      i-th data and j-th data
    p ← j-th cluster labels
    IF p = current cluster index THEN
      min distance ext ← maximum value of type data
      FOR k = 0 TO number of cluster DO
        IF k ≠ current cluster index THEN
          distance ext ← compute distance between …
            j-th data and k-th smem cluster medoids
          IF distance ext < min distance ext THEN
            min distance ext ← distance ext
          END
        END
      END
      IF min distance ext < maximum value of type …
        data THEN
        min value ← minimum value of …
          min distance ext and distance
        gain int ← gain int + min value
      END
    ELSE
      distance int ← compute distance between …
        i-th data and p-th smem cluster medoids
      max value ← maximum value of …
        (distance int – distance) and 0
      gain ext ← gain ext + max value
    END
  END

  q ← threadIdx.x
  q-th smem gain int ← gain int
  q-th smem gain ext ← gain ext
  SYNCHRONIZE the threads
  gain int ← sum reduction of smem gain int
  gain ext ← sum reduction of smem gain ext
  SYNCHRONIZE the threads

  IF threadIdx.x = 0 THEN
    i-th gain of potential medoids ← gain ext + …
      gain medoids – gain int
  END
END
```

Algorithm 4. Compute the new medoids.

```
GPU CONFIGURATION
  blocks ← 256
  grids ← 1

stop iteration ← true

READ  data,
      cluster medoids,
      cluster labels,
      medoid labels,
      gain of non medoids,
      current cluster index,
      number of data,
      number of cluster

ALLOCATE shared memory (smem) to store
  smem max gain ← 0
  smem max gain index ← 0

i ← index of data
j ← current cluster index
max gain ← 0
max gain index ← j-th cluster labels
IF i < number of data THEN
  IF i-th gain of potential medoids > max gain THEN
    max gain index ← i
    max gain ← i-th gain of potential medoids
  END
END

k ← threadIdx.x
k-th smem max gain ← max gain
k-th smem max gain index ← max gain index
SYNCHRONIZE the threads

max gain ← max reduction of smem max gain
max gain index ← index of max reduction value …
  of smem max gain
SYNCHRONIZE the threads

p ← max gain index
IF max gain > 0 THEN
  IF threadIdx.x = 0 THEN
    j-th medoid labels ← p
    stop iteration ← false
  END
  j-th cluster medoids ← p-th data
END
```

Based on the complexity of k-medoids, the most complex computation is in the process of finding new medoids thus, it will be parallelized. If n is the number of data, the PAM algorithm can be optimized using $n \times n$ table of distance that pre-calculated before k-medoids computation executed. However, creating $n \times n$ table of distance requires a large amount of memory. The goal is to optimize the $O(n-k)^2$ using parallelization scheme without creating a table of distance to avoid large memory consumption on GPU.

Algorithm 2 shows the computation of the gain of medoids to each data within the cluster. Gain is the sum of the distance between the medoids and each data in the same cluster. Shared memory is utilized to store the cluster medoids that repeatedly used in the calculation of distance. This will reduce the latency of global memory access to faster-shared memory access. Data are partitioned and processed by the thread blocks. Each thread computes the nearest medoid to each data, assigns the nearest medoids index to cluster label, and sum up the gain using atomic addition from CUDA in shared memory. The total gain that summed from each data then copied to the device memory.

Algorithm 3 shows the computation of the gain of non-medoids to each data. The configuration of blocks and grids is different from Algorithm 2. While n is the number of data and k is the number of clusters, then Algorithm 3 requires $(n-k)$ data compared to n data. Here, Algorithm 2 only requires n data compare to k data, where k is usually a small number. Therefore in Algorithm 2, the thread blocks assigned to handle the outer (n) loop and serialize the inner (k) loop while in Algorithm 3, the threads assigned to handle the inner (n) loop and the blocks assigned to handle the outer $(n-k)$ loop. With this configuration, maximum occupancy can be achieved.

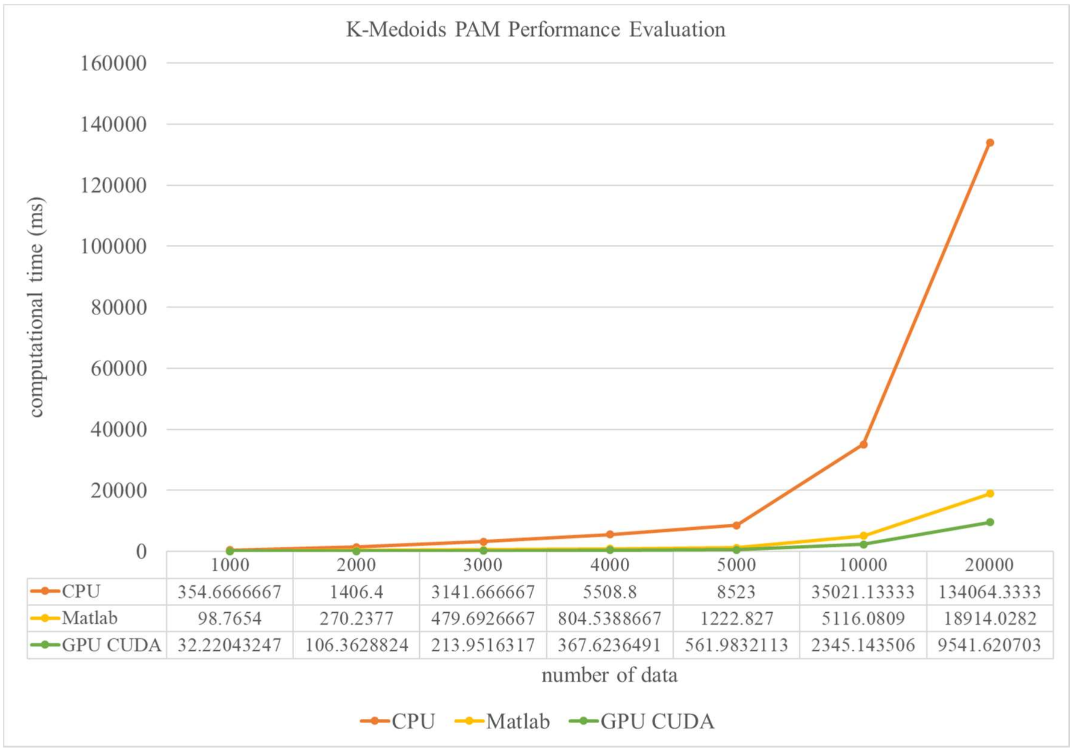
In the inner loop computation, each non-medoids is compared to the entire dataset. Total gain of each non-medoids is computed by adding the gain of medoid to each data within a cluster (from Algorithm 2) with the gain of non-medoids to each data from outer cluster then subtracted by the gain of non-medoids to each data within a cluster. Because the threads handle the inner loop, sum reduction can be used in shared memory to sum up the gain of non-medoids to each data from the outer cluster with the gain of non-medoids to each data within the cluster. Shared memory also utilized to store the cluster medoids to provide faster access in the distance calculation that similar in Algorithm 2. The total gain of each non-medoids then copied to device memory to be used in the computation of new medoids.

Algorithm 4 shows the computation of new medoids. New medoid is computed by finding the maximum gain that greater than zero from the gain of non-medoids in Algorithm 3. One block is used for the kernel to perform max reduction on n data. The index of maximum gain indicates that the data which corresponds to that index has the highest potential as the new cluster medoid. The index and the new cluster medoid then copied to the device memory. If the maximum gain of non-medoids is less than zero for all cluster, then k-medoids iteration will stop otherwise the iteration continues.

B. Performance Comparison

The proposed parallelized PAM in k-medoids is tested using KDDCup dataset. The dataset is modified into a smaller set with a various number of data. We use $k = 5$ to cluster 1,000, 2,000, 3,000, 4,000, 5,000, 10,000, and 20,000 data with 41 attributes. The same initial cluster medoids are used, and the algorithms are computed 10 times to get the average computational time. The experiment compares computational time between CPU implementation of PAM k-medoids with the proposed GPU implementation of PAM k-medoids without creating a pre-calculated table of distance. Fig. 2 shows the performance, is measured based on the computational time in milliseconds against the number of data. The red, orange, and green lines present the CPU, Matlab, and the proposed GPU implementation, respectively.

Based on Fig. 2, the proposed parallelized PAM k-medoids achieves more than 11 times speed up from the CPU implementation. For the larger dataset, the computational time of CPU implementation rises significantly, which indicates the proposed GPU implementation is efficient in dealing with this problem. Fig. 2 also shows the performance evaluation of the proposed GPU and Matlab implementation. Matlab uses a table of distance in its PAM k-medoids computation. Matlab recommends the implementation of PAM to be used in a small dataset, which less than 3000 data. The speedup gain using this method compared to the CPU implementation is approximately 3 times and more for more massive datasets. By accessing the lookup table, distance computation on each iteration can be simplified to only reading the data, thus reduce the computational time. However, by using a pre-calculated table of distance, space complexity increases significantly for large datasets. The proposed GPU implementation achieves approximately 2~3 times speedup compared to the Matlab implementation. The result is acceptable because the proposed parallelized PAM does not use a pre-calculated table of distance; thus, the potential implementation for a larger dataset is possible.



	1000	2000	3000	4000	5000	10000	20000
CPU	354.6666667	1406.4	3141.666667	5508.8	8523	35021.13333	134064.3333
Matlab	98.7654	270.2377	479.6926667	804.5388667	1222.827	5116.0809	18914.0282
GPU CUDA	32.22043247	106.3628824	213.9516317	367.6236491	561.9832113	2345.143506	9541.620703

Fig. 2. PAM K-medoids performance evaluation

IV. Conclusion

In this research, parallelized PAM k-medoids on GPU is proposed. PAM algorithm for k-medoids returns the best medoids compared to the other algorithms because it uses the entire dataset to find the best potential medoids. However, it has a high computational complexity. PAM algorithm usually implemented using a pre-calculated table of distance to avoid repeated distance calculation, which leads to massive memory consumption. The proposed implementation optimizes the distance computation of the PAM algorithm using a parallel scheme without the pre-calculated table of distance. From the experiment, the proposed parallelized PAM k-medoids is faster 2~3 times from Matlab and 11~15 times from CPU implementation. The proposed method can handle large datasets by performing data partition and process the data in parallel. For future work, the proposed method will be improved to handle categorical dataset, increase the performance using multi GPUs, and compared to the other parallel k-medoids clustering libraries.

Acknowledgment

This research is supported by The Indonesian Ministry of Research, Technology and Higher Education (RISTEK-DIKTI) research grant no. PEKERTI-056/SP3/LPP-UAD/IV/2017.

Declarations

Author contribution

All authors contributed equally as the main contributor of this paper. All authors read and approved the final paper.

Funding statement

This research receives funding from The Indonesian Ministry of Research, Technology and Higher Education (RISTEK-DIKTI) research grant no. PEKERTI-056/SP3/LPP-UAD/IV/2017.

Conflict of interest

The authors declare no conflict of interest.

Additional information

No additional information is available for this paper.

References

[1] Y. Zou and B. Liu, “Survey on clustering-based image segmentation techniques,” in *Proceedings of the 2016 IEEE 20th International Conference on Computer Supported Cooperative Work in Design, CSCWD 2016*, Sep. 2016, pp. 106–110, doi: 10.1109/CSCWD.2016.7565972.

[2] N. Dhanachandra and Y. J. Chanu, “A Survey on Image Segmentation Methods using Clustering Techniques,” *Eur. J. Eng. Res. Sci.*, vol. 2, no. 1, p. 15, Jan. 2017, doi: 10.24018/ejers.2017.2.1.237.

[3] A. Saxena *et al.*, “A review of clustering techniques and developments,” *Neurocomputing*, vol. 267, pp. 664–681, Dec. 2017, doi: 10.1016/j.neucom.2017.06.053.

[4] A. Prahara, I. T. R. Yanto, and T. Herawan, “Histogram thresholding for automatic color segmentation based on k-means clustering,” in *Advances in Intelligent Systems and Computing*, 2017, vol. 549 AISC, pp. 344–354, doi: 10.1007/978-3-319-51281-5_35.

[5] S. Wazarkar and B. N. Keshavamurthy, “A survey on image data analysis through clustering techniques for real world applications,” *J. Vis. Commun. Image Represent.*, vol. 55, pp. 596–626, Aug. 2018, doi: 10.1016/j.jvcir.2018.07.009.

[6] D. K. Tasoulis, V. P. Plagianakos, and M. N. Vrahatis, “Unsupervised clustering of bioinformatics data,” in *European Symposium on Intelligent Technologies, Hybrid Systems and their implementation on Smart Adaptive Systems, Eunite*, 2004, pp. 47–53.

[7] J. D. MacCuish and N. E. MacCuish, *Clustering in bioinformatics and drug discovery*. CRC Press, 2010.

[8] P. Berkhin, “A survey of clustering data mining techniques,” in *Grouping Multidimensional Data: Recent Advances in Clustering*, Springer Berlin Heidelberg, 2006, pp. 25–71.

[9] C. K. Reddy and B. Vinzamuri, “A Survey of Partitional and Hierarchical Clustering Algorithms.,” *Data Clust. Algorithms Appl.*, vol. 87, 2013.

[10] P. Arora, Deepali, and S. Varshney, “Analysis of K-Means and K-Medoids Algorithm for Big Data,” in *Physics Procedia*, Jan. 2016, vol. 78, pp. 507–512, doi: 10.1016/j.procs.2016.02.095.

[11] E. Schubert and P. J. Rousseeuw, “Faster k-Medoids Clustering: Improving the PAM, CLARA, and CLARANS Algorithms,” in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, Oct. 2019, vol. 11807 LNCS, pp. 171–187, doi: 10.1007/978-3-030-32047-8_16.

[12] P. O. Olukanmi, F. Nelwamondo, and T. Marwala, “PAM-lite: Fast and accurate k-medoids clustering for massive datasets,” in *Proceedings - 2019 Southern African Universities Power Engineering Conference/Robotics and Mechatronics/Pattern Recognition Association of South Africa, SAUPEC/RobMech/PRASA 2019*, May 2019, pp. 200–204, doi: 10.1109/RoboMech.2019.8704767.

[13] H. Song, J.-G. Lee, and W.-S. Han, “PAMAE: Parallel k-Medoids Clustering with High Accuracy and Efficiency,” in *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2017, pp. 1087–1096, doi: 10.1145/3097983.3098098.

[14] Ying-ting Zhu, Fu-zhang Wang, Xing-hua Shan, and Xiao-yan Lv, “K-medoids clustering based on MapReduce and optimal search of medoids,” in *2014 9th International Conference on Computer Science & Education*, Aug. 2014, pp. 573–577, doi: 10.1109/ICCSE.2014.6926527.

[15] A. Martino, A. Rizzi, and F. M. Frattale Mascioli, “Efficient approaches for solving the large-scale k-medoids problem: Towards structured data,” in *Studies in Computational Intelligence*, Nov. 2019, vol. 829, pp. 199–219, doi: 10.1007/978-3-030-16469-0_11.

[16] A. Prahara, D. P. Ismi, A. I. Kistijantoro, and M. L. Khodra, “Parallelized k-means clustering by exploiting instruction level parallelism at low occupancy,” in *Proceedings - 2017 2nd International Conferences on Information Technology, Information Systems and Electrical Engineering, ICITISEE 2017*, Feb. 2018, vol. 2018-January, pp. 30–34, doi: 10.1109/ICITISEE.2017.8285516.

[17] X. Wang, “A Survey of Clustering Algorithms Based on Parallel Mechanism,” Apr. 2018, pp. 119–122, doi:10.2991/cmsa-18.2018.28.

[18] Y. Jiang and J. Zhang, “Parallel K-Medoids clustering algorithm based on Hadoop,” in *2014 IEEE 5th International Conference on Software Engineering and Service Science*, Jun. 2014, pp. 649–652, doi: 10.1109/ICSESS.2014.6933652.

[19] M. O. Shafiq and E. Torunski, “A Parallel K-Medoids Algorithm for Clustering based on MapReduce,” in *2016 15th IEEE International Conference on Machine Learning and Applications (ICMLA)*, Dec. 2016, pp. 502–507, doi: 10.1109/ICMLA.2016.0089.

[20] X. Yue, W. Man, J. Yue, and G. Liu, “Parallel K-Medoids++ Spatial Clustering Algorithm Based on MapReduce,” pp. 1–8, Aug. 2016, Accessed: Jun. 21, 2020. [Online]. Available: http://arxiv.org/abs/1608.06861.

[21] D. Rajendran, S. Jangiti, S. Muralidharan, and M. Thendral, “Incremental MapReduce for K-Medoids Clustering of Big Time-Series Data,” in *Proceedings of the 2nd International Conference on Trends in Electronics and Informatics, ICOEI 2018*, Nov. 2018, pp. 1143–1146, doi: 10.1109/ICOEI.2018.8553756.

[22] Y. Zhao, B. Chen, and M. Li, “Parallel K-Medoids Improved Algorithm Based on MapReduce,” in *Proceedings - 2018 6th International Conference on Advanced Cloud and Big Data, CBD 2018*, Nov. 2018, pp. 18–23, doi: 10.1109/CBD.2018.00013.

[23] R. Wu, B. Zhang, and M. Hsu, “Clustering billions of data points using GPUs,” in *Proc. Combined Workshops on UnConventional High Performance Computing Workshop Plus Memory Access Workshop, UCHPC-MAW ’09, Co-located with the 2009 ACM Int. Conf. on Computing Frontiers, CF’09*, 2009, pp. 1–5, doi: 10.1145/1531666.1531668.

[24] E. Zhou, S. Mao, M. Li, and Z. Sun, “PAM spatial clustering algorithm research based on CUDA,” in *International Conference on Geoinformatics*, Sep. 2016, vol. 2016-September, doi: 10.1109/GEOINFORMATICS.2016.7578971.

[25] Y. Li, K. Zhao, X. Chu, and J. Liu, “Speeding up k-Means algorithm by GPUs,” *J. Comput. Syst. Sci.*, vol. 79, no. 2, pp. 216–229, Mar. 2013, doi: 10.1016/j.jcss.2012.05.004.

[26] K. R. Kurte and S. S. Durbha, “High resolution disaster data clustering using Graphics Processing Units,” in *2013 IEEE International Geoscience and Remote Sensing Symposium - IGARSS*, Jul. 2013, pp. 1696–1699, doi: 10.1109/IGARSS.2013.6723121.

[27] K. J. Kohlhoff, M. H. Sosnick, W. T. Hsu, V. S. Pande, and R. B. Altman, “CAMPAIGN: an open-source library of GPU-accelerated data clustering algorithms,” *Bioinformatics*, vol. 27, no. 16, pp. 2321–2322, Aug. 2011, doi: 10.1093/bioinformatics/btr386.

[28] L. Kaufman and P. J. Rousseeuw, *Clustering by Means of Medoids*. Faculty of Mathematics and Informatics, 1987.

[29] R. T. Ng and Jiawei Han, “CLARANS: a method for clustering objects for spatial data mining,” *IEEE Trans. Knowl. Data Eng.*, vol. 14, no. 5, pp. 1003–1016, Sep. 2002, doi: 10.1109/TKDE.2002.1033770.

[30] J. Fung and S. Mann, “Using graphics devices in reverse: GPU-based Image Processing and Computer Vision,” in *2008 IEEE International Conference on Multimedia and Expo*, Jun. 2008, pp. 9–12, doi: 10.1109/ICME.2008.4607358.

[31] L. Kaufman, P. J. R. Leonard Kaufman, and P. J. Rousseeuw, *Finding Groups in Data: An Introduction to Cluster Analysis*. Wiley, 1990.

[32] H.-S. Park and C.-H. Jun, “A simple and fast algorithm for K-medoids clustering,” *Expert Syst. Appl.*, vol. 36, no. 2, pp. 3336–3341, Mar. 2009, doi: 10.1016/j.eswa.2008.01.039.