Hindawi
Journal of Electrical and Computer Engineering
Volume 2022, Article ID 8260283, 11 pages
https://doi.org/10.1155/2022/8260283

Research Article

FCM Clustering Approach Optimization Using Parallel High-Speed Intel FPGA Technology

Abedalmuhdi Almomany,[1] **Amin Jarrah**,[1] **and Anwar Al Assaf**[2]

[1]*Department of Computer Engineering, Yarmouk University, Irbid, Jordan*
[2]*Aviation Sciences Dean/AMMAN Arab University, Amman, Jordan*

Correspondence should be addressed to Abedalmuhdi Almomany; emomani@yu.edu.jo

Received 3 February 2022; Accepted 19 April 2022; Published 11 May 2022

Academic Editor: Jose R. C. Piqueira

Fuzzy C-Means (FCM) is a widely used clustering algorithm that performs well in various scientific applications. Implementing FCM involves a massive number of computations, and many parallelization techniques based on GPUs and multicore systems have been suggested. In this study, we present a method for optimizing the FCM algorithm for high-speed field-programmable gate technology (FPGA) using a high-level C-like programming language called open computing language (OpenCL). The method was designed to enable the high-level compiler/synthesis tool to manipulate a task-parallelism model and create an efficient design. Our experimental results (based on several datasets) show that the proposed method makes the FCM execution time more than 186 times faster than the conventional design running on a single-core CPU platform. Also, its processing power reached 89 giga floating points operations per second (GFLOPs).

1. Introduction

Clustering is a topic of great interest in machine learning fields dealing with the process of partitioning sets of data into homogeneous groups or clusters based on the similarities between data points. Clustering techniques are useful because they allow the exploration of labeled and unlabeled data to find similarities and assign observations to corresponding clusters. The distance measure between elements in a dataset is commonly used in cases where points that are close to each other are assigned to the same cluster. Clustering algorithms are widely used in many fields such as medical applications [1], computer vision [2], data segmentation [3], marketing [4], networking [5], and security [6]. Some of the most commonly used clustering algorithms are K-means clustering [7], fuzzy clustering [8], hierarchical clustering [9], two-step clustering [10], and k-harmonic clustering [11]. While the overall goals of different clustering algorithms tend to be similar, diverse starting points and rules usually result in diverse taxonomies of clustering algorithms [12].

They are various kinds of clustering methods in which objects can be arranged into distinct groups based on a set of strategies and rules. In hierarchical clustering, the data objects are organized into a tree of clusters using top-down or bottom-up approaches by splitting clusters recursively until there are no more clusters. While it is easy to generate the tree without determining the number of clusters in advance, most of the hierarchical algorithms have high time complexity (could be $O(n^3)$) with a challenge that once the process of clusters combining or splitting is done, it cannot be undone [13, 14]. Centroid clustering algorithms are based on the distance calculation between the dataset objects and the proposed centroids; generally, these approaches give a reasonable performance, but they are very sensitive to noise and outliers [15]. Density-based clustering approaches are less effective to noise and outliers but it is not easy to work with large datasets; they group objects into separate regions based on their density; in grid-based clustering, the data space of the objects is mapped to a grid space with a limited number of cells and the clustering approach is performed on the whole cells which make this clustering model a fast

approach but with a challenge of the inability to find clusters in a low dimensional subspace [16, 17]. Model-based algorithms use statistical or neural learning strategies to choose a certain model for each cluster; they provide flexible implementation but with a low prediction quality relatively [16].

Several studies provided comparative analyses between many clustering algorithms by considering various parameters [18–21]. In [20], Abaas introduced a comparative analysis where the most popular clustering approaches are compared in terms of the ability to handle large datasets, type of dataset, and the number of clusters to classify. Results indicated that these algorithms vary in their performance and their classification accuracy degree. In [19], the authors presented a performance comparison study between nine of the most well-known clustering approaches to provide guidance on how to choose the clustering approach based on dataset characteristics, where the data have a normal distribution. In [21], authors investigated the effect of choosing clustering algorithms that may employ within pattern recognition applications on the overall recognition accuracy and clustering quality, the validity of the clustering approach, and the execution time of the clustering method; the outcome of the results showed that the effect of choosing clustering algorithm is essential in the case of small models' size. In general, there are many challenges associated with using most of the clustering approaches such as the noise problem and the process of initializing several parameters [22]; however, the FCM clustering approach is among the popular clustering approaches that has a reasonable performance and accuracy attributes [23].

The K-means clustering algorithm is probably the most common procedure, and several research studies have proposed different techniques for enhancing the traditional algorithm's performance [24–29]. Given a dataset X with n-points, each of which is a vector of d-dimensions, the K-means algorithm divides these points into k clusters via several steps. In the first step, the initial centers are chosen randomly. In the following steps, each point in the dataset is assigned to a single cluster (hard clustering) according to which center each data point is closest to. New centers are then determined for each cluster. These steps are repeated r times, and the overall complexity of this algorithm is calculated as $O(nkdr)$.

However, there are many limitations associated with using the K-means algorithm. Among these are the sensitivity of choosing the initial centroids, the weak ability to deal with noise and outlier data points, and the challenges that arise when dealing with a large data size [30]. The selection of initial centroids is poor in that there are many outliers. Also, applying this algorithm to a large dataset increases the number of iterations and the overall execution time, thus making it difficult to meet the desired level of performance.

Another common clustering technique is the fuzzy c-means (FCM) algorithm, by which computations are accelerated using high-speed FPGA technology. The FCM algorithm differs from the K-means in that the data points could belong to more than one cluster with varying probabilities (soft clustering).

Several studies that have used clustering approaches for diverse purposes have concluded that the FCM produced better results than the K-means algorithm based on its selection of a set of predefined parameters such as dealing with outliers and stability. However, the FCM algorithm's computation time is higher [31–38]. The longer computational time is linked to the complexity of the FCM algorithm, which is $O(nk^2dr)$, whereas that of the K-means algorithm is $O(nkdr)$.

Therefore, the use of parallel algorithms and high-speed computation platforms is suggested to overcome the relatively high computation time. The concept of fuzzy data was introduced in 1965 [39], which provides a mechanism for manipulating and interpreting complex multifeature data objects [40]. The fuzzy c-means algorithm [41] starts by determining the number of clusters k, whose initial centroids are chosen randomly. Those centroids are updated with every iteration of the algorithm, as is the membership function of each data point in the dataset. This updating process continues until the centroid values change. These changes, known as centroids movements, are minimal and must be less than a predefined threshold value. The degree of member function for each observation is equal to a value that belongs to the interval [0, 1]. A greater value toward a specific cluster means a stronger relationship with the center of that cluster. FCM algorithm determines whether the object belongs to a particular class based on its membership function [36]. The member function vector depends mainly on the Euclidian distance between the corresponding target object and the current centroid in each class; however, the object could belong to more than one class with a different possibility [42, 43]. The FCM algorithm is an iterative approach where centroids and member functions are updated in every iteration as described in equations (5) and (6); the member function degree of each object is estimated such that the value is bounded between 0 and 1. Finally, when almost there are no changes in the centroids list, objects are classified into a set of K classes according to the principle of highest membership.

The steps of the FCM algorithm can be summarized as below:

A dataset X contains n vectors of observations such that $X = \{x_1, x_2, \ldots, x_n\}$ and K clusters with initial centroids C, where $C = \{c_1, \ldots, c_K\}$. Also, we can define the 2D array member function F, where F is an array of n rows and K columns based on equation

$$F = \begin{bmatrix} f_{1,1} & \cdots & f_{1,k} \\ \vdots & \vdots & \vdots \\ f_{n,1} & \cdots & f_{n,k} \end{bmatrix}, \quad \text{where } 0 \le f_{i,j} \le 1,\ \forall\ i \in [1, N],\ j \in [1, k]. \tag{1}$$

At first, all elements in the F array are filled randomly; however, the conditions in equations (2) and (3) should still be satisfied.

$$\sum_{j=1}^{j=k} f_{i,j} = 1, \quad \forall\, i = 1, \ldots, N, \tag{2}$$

$$0 \leq \sum_{i=1}^{i=N} f_{i,j} \leq N, \quad \forall j = 1, \ldots, k. \tag{3}$$

The FCM objective function is applied to minimize the equation as follows:

$$J_{\min} = \sum_{i=1}^{i=N} \sum_{j=1}^{j=K} f_{ij}^{m} \left\| x_i - c_j \right\|^2, \tag{4}$$

where m is the fuzzier exponent. It is generally hard to select the optimal value of m, though since it could be in the ranges of 1.5 to 4 [44, 45] in this study, we chose $m = 3$. The term $\|x_i - c_j\|^2$ refers to the Euclidian distance between data point x_i and center c_j $\forall i = 1, \ldots, N,\ j = 1, \ldots, k$.

We can summarize the steps of the FCM algorithm as below:

Step 1: Initialize $F^{(\text{iter})} = [f_{ij}]$, where iter = 0.

Step 2: Calculate the new centers vector $C^{(\text{iter})} = [c_{ij}]$

$$c_{ij}^{(\text{iter})} = \frac{\sum_{i=1}^{i=N} f_{ij}^{m} \cdot x_i}{\sum_{i=1}^{i=N} f_{ij}^{m}}. \tag{5}$$

Step 3: Update the new F or $F^{(\text{iter}+1)}$

$$f_{ij}^{(\text{iter}+1)} = \frac{1}{\sum_{a=1}^{a=K} \left(\left\| x_i - c_j \right\|^2 / \left\| x_i - c_a \right\|^2 \right)^{(1/m-1)}}. \tag{6}$$

Increase the number of iterations (iter = iter + 1).

Step 4: If the stop condition is not satisfied, return to step 2; otherwise, STOP.

In this study, the stop condition is met when there are almost no changes in the centroids or when the average changes of all centroids are less than the threshold value ε, where $\varepsilon = 10^{-7}$.

The remainder of this paper is organized as follows: Section 2 introduces the FPGA computing platform and its characteristics. The OpenCL is presented in Section 3. In Section 4, we discuss related work and literature reviews. The proposed approach for the performance is discussed in Section 5. Experimental results are presented in Section 6. Section 7 provides a conclusion of the proposed study.

2. FPGA Computing Platform

High-speed computing platforms such as FPGAs have become popular in many applications, including image and signal processing, machine learning, security, pattern recognition, and scientific problems [46–51]. FPGA technology creates a customized hardware design that reflects desired objectives. FPGA technology also has software flexibility. At the same time, it is still possible for the platform to be reconfigured many times with many possible configurations, thus making the process of optimizing the proposed design more comfortable.

In Intel FPGA devices, the structure usually incorporates adaptive logic modules (ALMs), RAM blocks, and extensive digital signal processing (DSP) blocks. FPGAs can also carry other kinds of blocks, such as phase-lock loops (PLLs), which can adjust the internal clock frequency. ALMs contain at least one lookup table (LUT), each of which is made of one or more flip-flops (FFs). These ALMs are diffused throughout the FPGA fabric, making the FPGAs very amenable for temporally parallel (systolic or pipelined) computations that can be applied to monopolize loop-level concurrency in several applications.

In such cases, the body of the loop is divided into executable pieces, with each piece targeted for execution on a different stage of computational logic generated within the FPGA. The data passed along pipelined stages are stored in discrete and accessible ALM flip-flop resources. Generally, when it is completely pipelined, it takes one clock cycle to pass an item of data from one stage to another in a mere temporal pipeline.

All stages concurrently perform their computations with different data. In such cases, the expected number of clock cycles required to handle any single item (usually referred to as pipeline latency) equals the number of stages in the system carried out to treat the body of the loop.

However, if there are a substantial number of elements in the loop, then the most important metric is the initiation interval (II). This metric reflects how many clock cycles the system should wait for, on average, before permitting the next item to enter the pipeline. A custom-created pipeline within an FPGA effectively reveals the low-level structure of an application. The device used—namely, Intel De5a-Net Arria-10—has 427,200 ALMs, which are used to implement several hardware circuits functions, 1518 DSP blocks to ensure the efficient implementation of several floating points operators, and 2713 RAM blocks to store data for the synthesized design.

2.1. OpenCL. Open Computing Language (OpenCL) is a programming framework that simplifies the distribution of works among multiple and different/similar kinds of computation platforms [52]. For the FPGA-based platform, the most influential benefits come from abstracting most of the hardware details and significantly reducing the development time [46, 52]. The OpenCL framework also enables various number of threads to be created such that the number of created threads can be set by the programmer according to the platform's architecture. A single thread could be applied, such as when using a task-parallel model in FPGA, or several threads can be created for each core in multicore systems. In some cases, such as when a graphical processing units (GPUs) computation model is used, millions of threads might be created.

The OpenCL programming model has two main parts: the host and device programs. The host code is usually written in a C/C++ programming language, and it manages all communication with the device. This part of the code is

compiled using the GNU g++ compiler. The device code is an OpenCL-based program that implements the segment of code that should be accelerated using a high-speed computation device (in this study, the De5a-Net FPGA device). The device code is compiled using the Intel FPGA compiler. As this process could take hours to complete, it should be carried out before compiling the host code. Figure 1 depicts the programming model flow.

2.2. Related Work. The FCM yields good clustering results but requires a long computation time. Therefore, many studies have attempted to improve the speed of the FCM algorithm. High-speed computation platforms such as GPUs [54–58], multicores [53], and FPGAs are utilized to run these complex computations by which the FCM algorithm is modified to tolerate the hardware features and achieve reasonable improvements.

Among these platforms, the FPGA provides an especially high-speed hardware solution that can be customized according to the algorithm's specifications. Afshin introduced a hardware solution approach allowing the FCM to be utilized for brain MR image segmentation [59]. In this case, the *Xilinx FPGA Virtex7* was used with the Modelsim tool to obtain the simulation results. MATLAB is used to run the FCM algorithm on an *Intel Core i5* machine to compare with the software. Although the speed improved approximately 100 times, the design was not implemented on a real hardware device.

In the field of image processing, an improved FCM algorithm was introduced that uses a pulse mode hardware structure to save resource usage and offers a reasonable speed improvement. The proposed design was tested and verified on the *Virtex-6 FPGA* platform [60].

Another study discussed the development of an improved FCM algorithm applicable to real-time applications such as video processing [61]. Specifically, the study discussed two hardware approaches to tune the performance of two different hardware structure devices from Xilinx and Altera.

In other work, Hwang et al. [62] discussed a hardware approach to optimize the FCM algorithm by creating a well-pipelined hardware circuit designed to perform the overall FCM computation steps. The proposed circuit design was implemented on a Stratix III device, with the results indicating the favorable performance of the proposed design. It also consumed a relatively small percentage of the available resources.

This study introduces the ability of a parallel high-level computing language (OpenCL) to tune the performance of a common heavy computation clustering algorithm to run on high-speed Intel FPGA technology. In addition to reducing latency, FPGAs are also widely utilized to reduce the overall energy consumption as proofed in [46].

2.3. Methods of Optimization and the Proposed Approach Performance Tuning. The continuously improving performance and effectiveness of the FPGAs platform have promoted their extensive use to accelerate diverse heavy computation applications. Several optimization techniques can be used for performance tuning and to maximize the possible benefits of different high-speed processing platforms. The Intel FPGA compiler generates a pipelined datapath hardware circuit in which several operations of different loops' iterations are executed concurrently (as shown in Figure 2).

This model of parallelism is called a task-parallel model (single work-item), by which processing is sped up by overlapping the loop iterations' execution in a given piece of code. The execution of each loop iteration is divided into multiple steps, each of which involves one or more instructions. Also, each step is executed in a one clock cycle time. In an ideal case, the next loop iteration starts the execution of the first step by shifting one clock cycle (one step) from the previous iteration. This is commonly referred to as an II and should be equal to 1 if the pipelined circuit is created perfectly.

As indicated in Figure 1, the execution of the proposed loop iterations overlaps, meaning the total execution time for the whole loop iterations is approximately equal to M clock cycles, where $M >> N$.

An effective pipelined datapath is created when the Intel compiler generates multiple files. Among these files is the optimization report file, which can be interpreted to modify the design so that the best possible pipelined circuit is created. The loop unrolling technique can also be utilized to improve the performance of the created design by increasing the amount of work performed per clock cycle. This reduces the total number of clock cycles by a factor of X when the main loop is unrolled X times. As shown in Figure 3, the number of iterations is reduced by half as the loop is unrolled by a factor of two.

However, the loop unrolling technique increases resource usage and may also increase the clock cycle time. If the clock cycle time for a given design is C, then the new clock cycle time after loop unrolling is α. C, where $\alpha \geq 1$. Loop unrolling (where U is the unrolling factor) is feasible if it reduces the overall time (T), given that

$$T_{\text{no\_loop\_unroll}} = C.M \text{ and } T_{\text{loop\_unroll\_}U} = \frac{\alpha.C.M}{U}. \quad (7)$$

Then, $T_{\text{no\_loop\_unroll}}$ should be greater than $T_{\text{loop\_unrolled\_U}}$, given that

$$\frac{\alpha.C.M}{U} \leq C.M, \quad U \geq \alpha. \quad (8)$$

If the condition in equation (8) is satisfied, the application of the loop unrolling technique is feasible.

Data dependencies between consecutive operations prevent the creation of effective pipeline circuits with an II that is close to one. A possible solution to this problem is to use the shift register concept, as it eliminates dependencies and allows the compiler to create a robust design with a better II value [52]. For example, if II is equal to four because there is a dependency between the output and at least one of the inputs, the next iteration waits four clock cycles before starting its execution (see Figure 4).

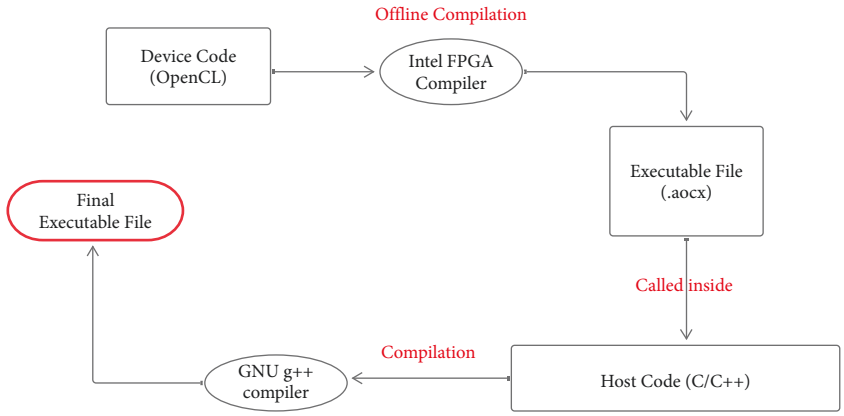

Figure 1: OpenCL project compilation process.

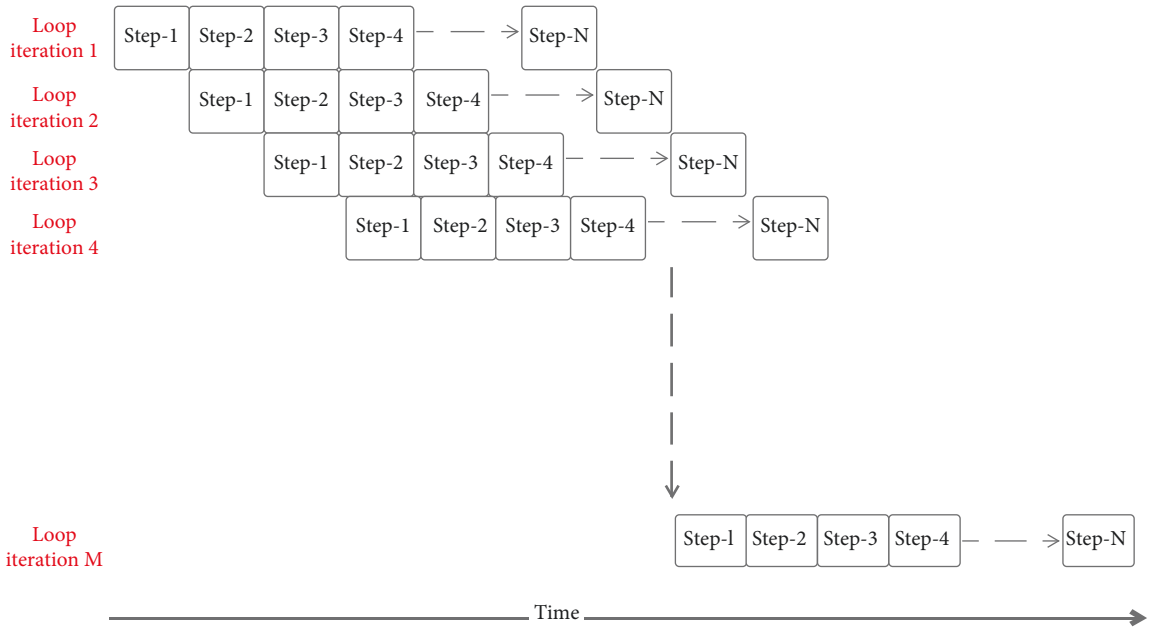

Figure 2: The pipelined data created by the Intel FPGA compiler.

Other techniques might also resolve this problem, such as using local memory, which reduces a significant part of executions by reducing data transfer time and moving the dependencies from global memory to local memory.

Using the mentioned optimization techniques together with the Intel compiler, we created an effective hardware design to run the fuzzy K-means algorithm much more quickly than a similar algorithm based on a general-purpose CPU-based computation platform. The characteristics of the FPGA device used in this study which has a substantial amount of local memory and the ability to create an effective pipeline circuit with a potentially large number of pipeline stages make the FPGA platform a favorable choice for accelerating complex computation-based algorithms.

Figure 5 shows the optimization process of a significant part of the code. In Figure 5(a), the loop runs without performing any optimization technique other than the Intel FPGA compiler's optimization of the hardware design to maximize performance. The II equals 9 because of the data's dependency on the variable that accumulates the sum of weights (*sumW*). Using loop unrolling (Figure 5(b)) reduces the execution time. However, the II increases to approximately 51, as more dependencies are generated in each iteration, and there is still an area of optimization. By combining local memory and shift register with loop unrolling (Figure 5(c)), we can create a more powerful pipeline design that solves the data dependencies obstacle with an II of *one*. The size of the shift register is set so that the number of pipeline stages is increased to hide the latency and eliminate pipeline stalls. As a result, the speed is increased by more than two times, as will be discussed further in the results and discussion section.

The primary challenge associated with optimization is the amount of resource usage in the first useable FPGA device (De5-Net Stratix V GX FPGA). Unrolling the loop five times increases the number of ALUTs from 169 k (36% of the total ALUTs) to 495 k (105%) and the number of DSPs from 200 blocks (78% of the total DSP blocks) to 584 blocks (228% of the total DSP blocks). This challenge motivates the

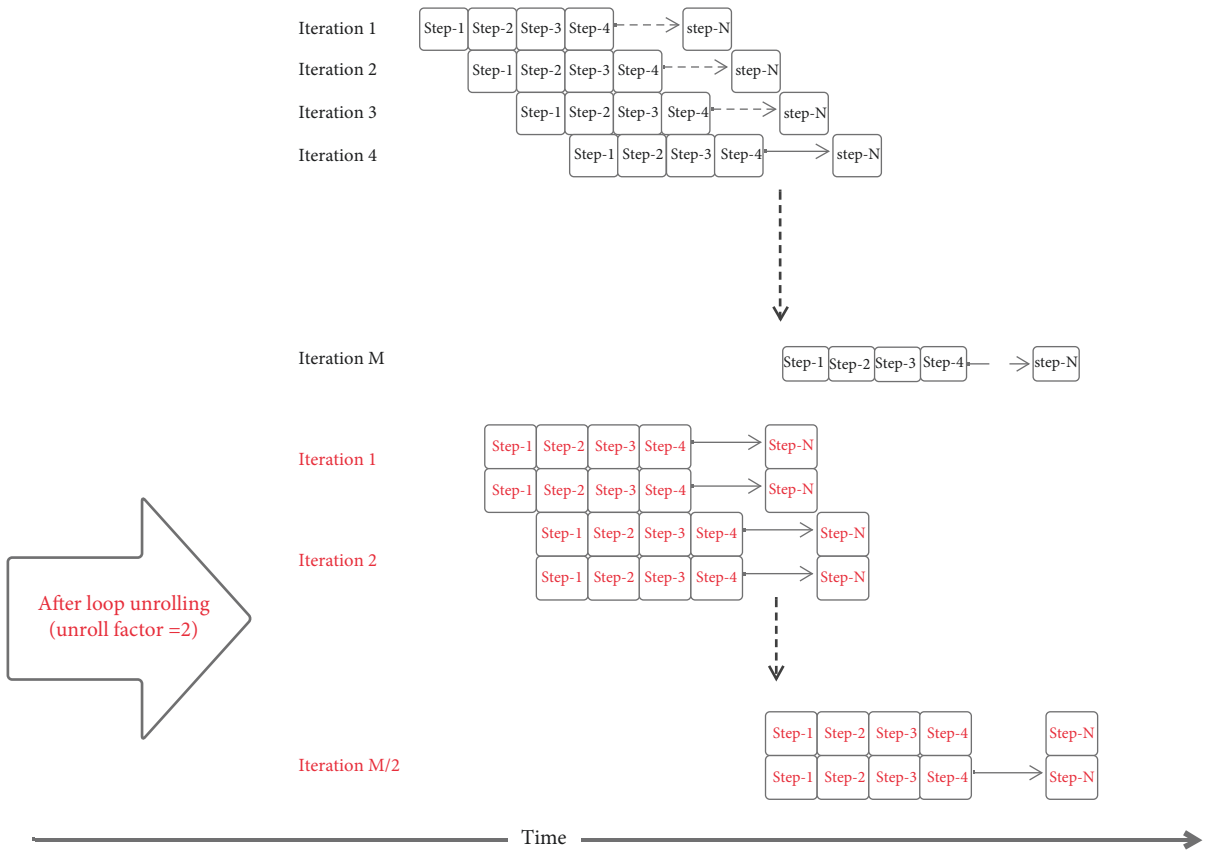

Figure 3: Reducing the number of iterations using the loop unrolling technique.

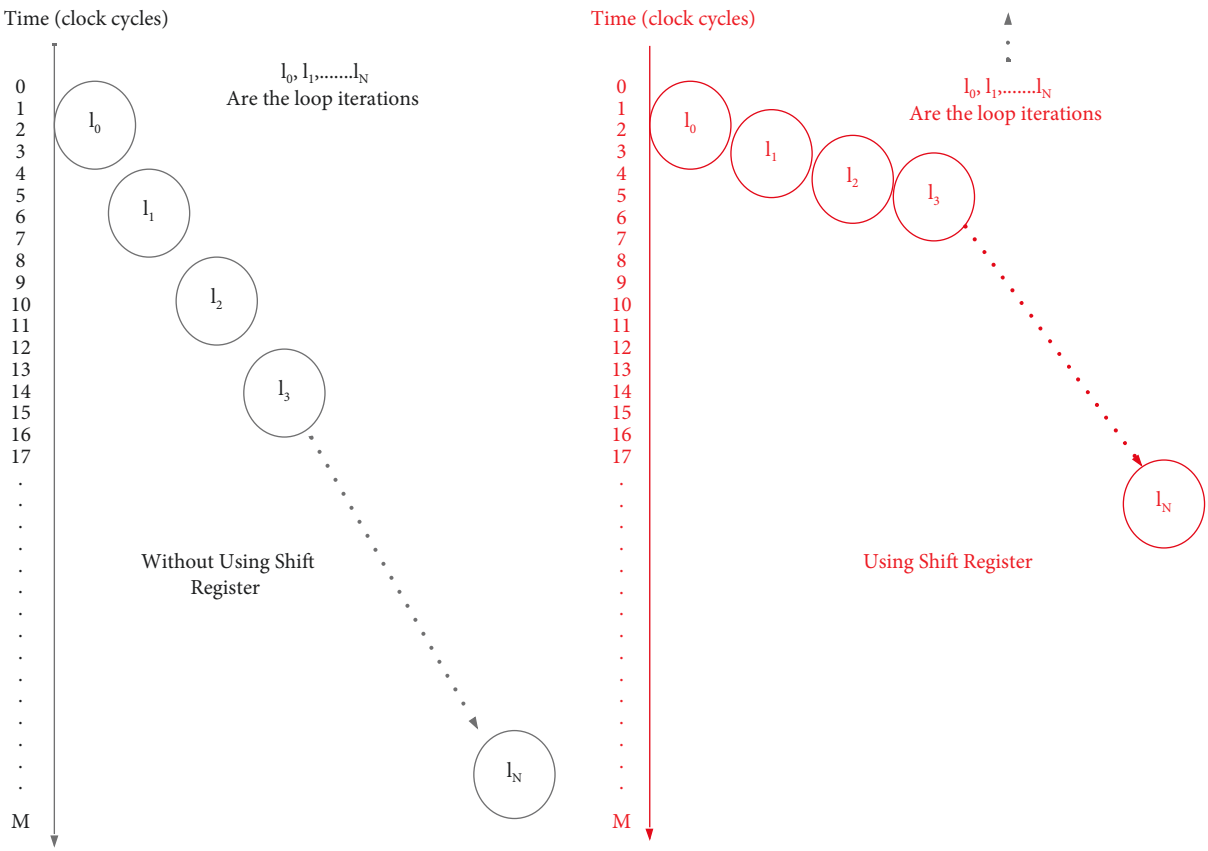

Figure 4: Enhancing the initiation interval (II) by using the shift register technique.

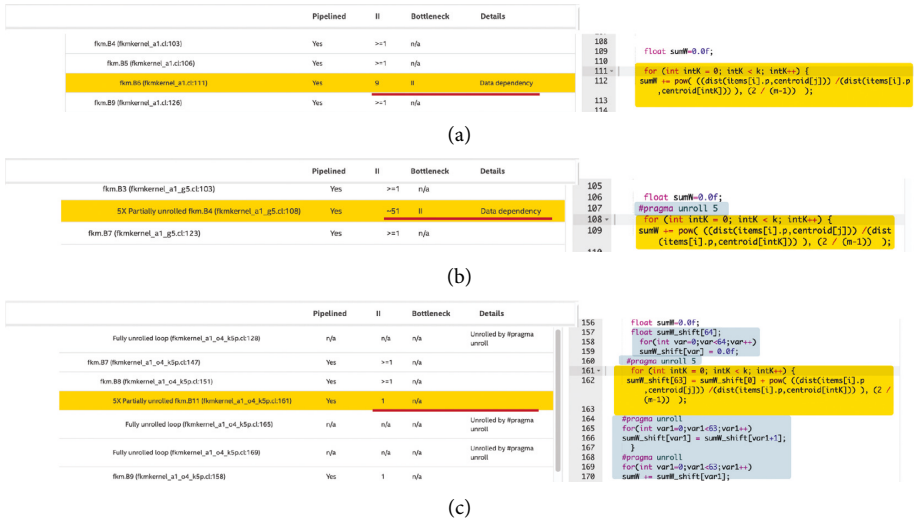

FIGURE 5: Using optimization techniques in a significant part of code. (a) Unoptimized loop. (b) Using loop unrolling. (c) Using Loop unrolling, shift register, and local memory.

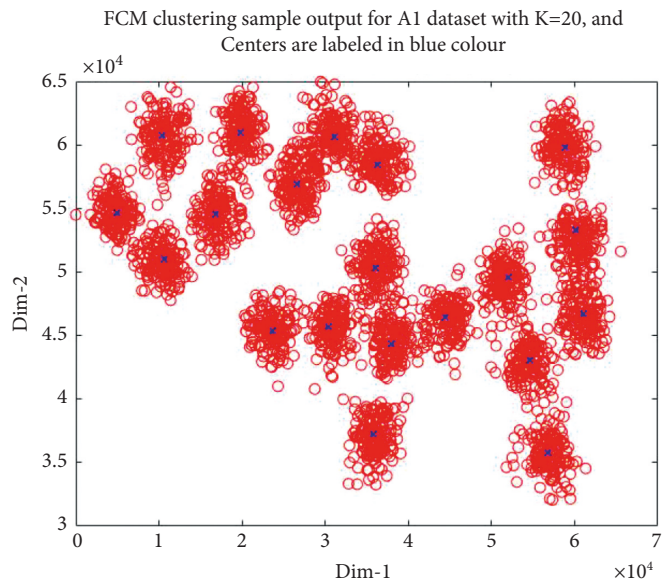

FIGURE 6: Sample clustering output for the A1 dataset.

use of the De5a-Net Arria-10 FPGA device, which has more ALUTs and DSP blocks. Specifically, the optimized code resource usage demands are 379 k of ALUs (49%), 623 RAM blocks (25%), and 682 DSP blocks (47% of total DSP blocks).

3. Experimental Results

Several datasets are considered in this study to compare the performance comparison of the FPGA with that of single-core CPU-based platforms. The first dataset is the balanced iterative reducing and clustering using hierarchies (BIRCH), which is a two-dimensional dataset containing 100 clusters and 100,000 vectors in a regular grid structure [63]. The A1 and A3 datasets are two-dimensional datasets with circular clusters. A1 has 20 clusters and 3000 vectors of elements, whereas A3 contains 7500 vectors with 50 clusters [64]. The last dataset is the unbalanced dataset, which is a two-dimensional dataset with eight clusters and 6500 vectors [65]. A sample output that presents the FCM clustering approach for one of the used datasets is shown in Figure 6.

The traditional serial-based code [66] is modified, compiled, and run on a CPU with an Intel Core i7-6700@ 3.4 GHZ with 16 GB of memory. The code is compiled using the GNU Compiler Collection (GCC/G++) with an "*O3*" standard level of optimization. The second code is optimized to run on the FPGA platform using a high-capability Intel FPGA device, namely, a De5a-Net Arria-10 powerful device.

Table 1 shows that the speedup factor is increased from 25 (for a smaller number of computations) to 39 (for a larger number of computations) when the optimized version of sequential code is compared. Speed is increased by up to 186-fold when FPGA is used to accelerate the computations

Table 1: Benchmarks execution times for both the CPU and FPGA computation platforms.

Benchmark	Number of clusters (K)	Number of iterations	Data size (number of vectors)	Execution time (seconds)			FPGA speedup	
				Serial code (CPU)		Parallel code (FPGA)		
				Optimized compiled with O3	Default level of optimization		Optimized	Default
BIRCH (DS-1)	100	735	100,000	20,526	99488	534	38.44	186.3
A3	50	374	7,500	202	970	5.31	38.04	182.7
A1	20	287	3,000	10.6	42.7	0.345	30.72	123.8
Unbalanced	8	442	6,500	6.24	29.2	0.247	25.26	118.2

Table 2: (Number of arithmetic operations $\times 10^6$) per each iteration.

Benchmark	Add	Sub	Mul	Div	Sqrt	Power
BIRCH (DS-1)	5551	5520	20	2211	2210	5540
A3	105	103.8	0.75	41.6	41.6	104.6
A1	6.9	6.7	0.12	2.7	2.7	6.8
Unbalance	2.5	2.4	0.1	.97	0.96	2.5

Table 3: GFLOPs performance comparison.

Benchmark	Number of clusters (K)	Number of iterations	GFLOP	GFLOPs (CPU)	(FPGA)
BIRCH (DS-1)	100	735	47225.2	2.3	88.44
A3	50	374	453.57	2.24	85.4
A1	20	287	22.65	2.14	65.65
Unbalance	8	442	12.75	2.04	51.62

instead of the sequential code compiled using the gnu compiler g++ with a default level of optimization.

The other measurement that can be used to compare performance is the number of floating-point operations (addition or multiplication) per second (FLOPs) performed. All high-order functions, such as square-root and exponential functions, can be constructed using basic adder and multiplier components. For the DE5a-Net board used in this research, the maximum theoretical FLOPs can be calculated by multiplying the total number of DSP blocks (each of which can perform two FLOPs per clock cycle) by the maximum clock cycle frequency (which is approximately 400 MHz). Thus, the maximum theoretical GFLOPs is 1248.

It is hard to achieve the maximum possible number of FLOPs in practice because it is not probable that all computational units will work at the same time [67]. The communication overheads between the utilized components and the data transfer, which generates more timing constraints, also prevent the maximum number of FLOPs from being achieved.

Using the performance application program interface (PAPI) [68], we can measure the number of FLOPs and other performance parameters such as the number of cache hits and misses. Table 2 shows the frequency of each arithmetic function used inside the proposed FCM algorithm for every iteration in the unit of MFLOP. From Table 2, we can see that the processing speed is increased by up to 89 GFLOPs when the FPGA accelerator device is used. All basic operations (add, sub, mul, and div) are considered as one floating-point operation. Furthermore, the square-root function is three floating-point operations, and the power function is eight floating-point operations.

The total number of floating-point operations carried out during all iterations is shown in Table 3. Table 3 also shows the GFLOPs performance of the PAPI tool.

4. Conclusion

The study introduced a hardware solution approach that utilizes a high-level abstraction language (namely, OpenCL) to accelerate the computation heavy FCM algorithm. The study also provides a simple way to create an efficient design that can be synthesized on the FPGA acceleration platform. The results demonstrate the effectiveness of the acceleration device and the optimization, indicated by speed improvements of up to 187 times when compared to a regular single-core CPU platform.

Data Availability

Datasets are derived from public resources and made available with the article through providing the required references.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Acknowledgments

This study was supported by the Intel FPGA University Program Ticket nos. LR4043 and BR 11211.

References

[1] H. Alashwal, M. El Halaby, J. J. Crouse, A. Abdalla, and A. A. Moustafa, "The application of unsupervised clustering methods to Alzheimer's disease," *Frontiers in Computational Neuroscience*, vol. 13, p. 31, 2019.

[2] D. K. Iakovidis, N. Pelekis, E. Kotsifakos, and I. Kopanakis, "Intuitionistic fuzzy clustering with applications in computer vision," in *Advanced Concepts for Intelligent Vision Systems. ACIVS 2008. Lecture Notes in Computer Science*, J. Blanc-Talon, S. Bourennane, W. Philips, D. Popescu, and P. Scheunders, Eds., vol. 5259, Berlin, Heidelberg, Springer, 2008.

[3] F. Yoseph, N. H. Ahamed Hassain Malim, M. Heikkilä, A. Brezulianu, O. Geman, and N. A. Paskhal Rostam, "The impact of big data market segmentation using data mining and clustering techniques," *Journal of Intelligent & Fuzzy Systems*, vol. 38, no. 5, pp. 6159–6173, 2020.

[4] M. Pondel and J. Korczak, "Collective clustering of marketing data-recommendation system upsaily," in *Proceedings of the 2018 Federated Conference on Computer Science and Information Systems (FedCSIS)*, pp. 801–810, Poznań, Poland, September 2018.

[5] Y. Sun, J. Han, P. Zhao, Z. Yin, H. Cheng, and T. Wu, "RankClus: integrating clustering with ranking for heterogeneous information network analysis," in *Proceedings of the 12th International Conference on Extending Database Technology: Advances in Database Technology (EDBT '09)*, pp. 565–576, Association for Computing Machinery, New York, NY, USA, March 2009.

[6] G. Mohler, "Modeling and estimation of multi-source clustering in crime and security data," *The Annals of Applied Statistics*, vol. 7, no. 3, pp. 1525–1539, 2013.

[7] A. Likas, N. Vlassis, and J. Verbeek, "The global k-means clustering algorithm," *Pattern Recognition*, vol. 36, no. 2, pp. 451–461, 2003.

[8] A. Gosain and S. Dahiya, "Performance analysis of various fuzzy clustering algorithms: a review," *Procedia Computer Science*, vol. 79, pp. 100–111, 2016, ISSN 1877-0509.

[9] F. Nielsen, "Hierarchical clustering," in *Introduction to HPC with MPI for Data Science. Undergraduate Topics in Computer Science*Springer, Cham, 2016.

[10] M. Benassi, S. Garofalo, F. Ambrosini et al., "Using two-step cluster Analysis and latent class cluster Analysis to classify the cognitive heterogeneity of cross-diagnostic psychiatric inpatients," *Frontiers in Psychology*, vol. 11, Article ID 1085, 2020.

[11] B. Zhang, M. Hsu, and U. Dayal, *K-harmonic Means—A Data Clustering Algorithm*, Technical Report HPL-124, Hewlett-Packard Labs, Palo Alto, CA, USA, 1999.

[12] R. Xu and D. WunschII, "Survey of clustering algorithms," *IEEE Transactions on Neural Networks*, vol. 16, no. 3, pp. 645–678, May 2005.

[13] D. P. Dabhi and M. R. Patel, "Extensive survey on Hierarchical Clustering methods in data mining," *International Research Journal of Engineering and Technology (IRJET)*, vol. 3, no. 11, pp. 659–665, Nov. 2016.

[14] S. Patel, S. Sihmar, and A. Jatain, "A study of hierarchical clustering algorithms," in *Proceedings of the 2015 2nd International Conference on Computing for Sustainable Global Development (INDIACom)*, pp. 537–541, New Delhi, India, March 2015.

[15] S. K. Uppada, "Centroid based clustering algorithms- A clarion study," *International Journal of Computer Science and Information Technologies*, vol. 5, no. 6, pp. 7309–7313, 2014.

[16] A. C. Benabdellah, A. Benghabrit, and I. Bouhaddou, "A survey of clustering algorithms for an industrial context," *Procedia Computer Science*, vol. 148, pp. 291–302, 2019.

[17] D. Brown, A. Japa, and Y. Shi, "A fast density-grid based clustering method," in *Proceedings of the 2019 IEEE 9th Annual Computing and Communication Workshop and Conference (CCWC)*, pp. 0048–0054, Las Vegas, NV, USA, January 2019.

[18] H. G. Garima and P. K. Singh, "Clustering techniques in data mining: a comparison," in *Proceedings of the 2015 2nd International Conference on Computing for Sustainable Global Development (INDIACom)*, pp. 410–415, New Delhi, India, March 2015.

[19] M. Z. Rodriguez, C. H. Comin, D. Casanova et al., "Clustering algorithms: a comparative approach," *PLoS One*, vol. 14, no. 1, Article ID e0210236, 2019.

[20] O. A. Abbas, "Comparisons between data clustering algorithms," *International Arab Journal of Information Technology*, vol. 5, no. 3, pp. 320–325, May 2008.

[21] T. Kinnunen, I. Sidoroff, M. Tuononen, and P. Fränti, "Comparison of clustering methods: a case study of text-independent speaker modeling," *Pattern Recognition Letters*, vol. 32, no. 13, pp. 1604–1617, 2011.

[22] S. Deng, "Clustering with Fuzzy C-means and common challenges," *Journal of Physics: Conference Series*, vol. 1453, no. 1, Article ID 012137, 2020.

[23] M. S. Choudhry and R. Kapoor, "Performance analysis of fuzzy C-means clustering methods for MRI image segmentation," *Procedia Computer Science*, vol. 89, pp. 749–758, 2016.

[24] M. S. Mahmud, M. M. Rahman, and M. N. Akhtar, "Improvement of K-means clustering algorithm with better initial centroids based on weighted average," in *Proceedings of the 2012 7th International Conference on Electrical and Computer Engineering*, pp. 647–650, Dhaka, Bangladesh, December 2012.

[25] L. Zhang, J. Qu, M. Gao, and M. Zhao, "Improvement of K-means algorithm based on density," in *Proceedings of the 2019 IEEE 8th Joint International Information Technology and Artificial Intelligence Conference (ITAIC)*, pp. 1070–1073, Chongqing, China, May 2019.

[26] H. Zhang, Yu Hong, Y. Li, and B. Hu, "Improved K-means algorithm based on the clustering reliability analysis," in *Proceedings of the 2015 International Symposium on Computers & Informatics*, January 2015.

[27] K. Mumtaz and Dr. K. Duraiswamy, "A novel density based improved k-means clustering algorithm dbkmeans," *International Journal on Computer Science and Engineering*, vol. 2, 2010.

[28] O. Sangita and J. Dhanamma, "An improved K-means clustering approach for teaching evaluation," in *Advances in Computing, Communication and Control. ICAC3 2011*, S. Unnikrishnan, S. Surve, and D. Bhoir, Eds., vol. 125, Berlin, Heidelberg, Springer, 2011.

[29] X. B. Liu, B. B. Deng, and L. N. Shen, "The improved K-means cluster Analysis on diagnosis data fusion of the aero-engine," *Applied Mechanics and Materials, Trans Tech Publications, Ltd.*vol. 328, pp. 463–467, June 2013.

[30] A. Ashabi, S. B. Sahibuddin, and M. Salkhordeh Haghighi, "The systematic review of K-means clustering algorithm," in

Proceedings of the 2020 The 9th International Conference on Networks, Communication and Computing (ICNCC 2020), vol. 13–18, 2020.

[31] U. Baid, S. Talbar, and S. Talbar, "Comparative study of K-means, Gaussian mixture model, fuzzy C-means algorithms for brain tumor segmentation," in *Proceedings of the International Conference on Communication and Signal Processing 2016 (ICCASP 2016)*, 2017.

[32] T. C. Hakyemez, A. Bozanta, and M. Coşkun, "K-means vs. Fuzzy C-means: a comparative analysis of two popular clustering techniques on the featured mobile applications benchmark," *IMISC. Journal Contribution*, 2019.

[33] A. K. Dubey, U. Gupta, and S. Jain, "Comparative study of K-means and fuzzy C-means algorithms on the breast cancer data," *International Journal on Advanced Science, Engineering and Information Technology*, vol. 8, no. 1, p. 18, 2018.

[34] S. A. Mingoti and J. O. Lima, "Comparing SOM neural network with Fuzzy c-means, K-means and traditional hierarchical clustering algorithms," *European Journal of Operational Research*, vol. 174, no. 3, pp. 1742–1759, 2006.

[35] Y. Zhang and J. Han, "Differential privacy fuzzy C-means clustering algorithm based on Gaussian kernel function," *PLoS One*, vol. 16, no. 3, Article ID e0248737, 23 Mar. 2021.

[36] H. R.M, F. Abbas, and A. Abdulkarem, "Performance evaluation of K-mean and fuzzy C-mean image segmentation based clustering classifier," *International Journal of Advanced Computer Science and Applications*, vol. 6, no. 12, 2015.

[37] H. Purnawansyah, A. F. O. Gafar, and I. Tahyudin, "Comparison between K-means and fuzzy C-means clustering in network traffic activities," in *Lecture Notes on Multidisciplinary Industrial Engineering*, J. Xu, M. Gen, A. Hajiyev, and F. Cooke, Eds., , 2018.

[38] W. Wiharto and E. Suryani, "The comparison of clustering algorithms K-means and fuzzy C-means for segmentation retinal blood vessels," *Acta Informatica Medica*, vol. 28, no. 1, pp. 42–47, 2020.

[39] L. A. Zadeh, "Fuzzy sets," *Information and Control*, vol. 8, no. 3, pp. 338–353, June 1965.

[40] C. Bai, D. Dhavale, and J. Sarkis, "Complex investment decisions using rough set and fuzzy c-means: an example of investment in green supply chains," *European Journal of Operational Research*, vol. 248, no. 2, pp. 507–521, 2016.

[41] J. C. Bezdek, *Pattern Recognition with Fuzzy Objective Function Algorithms*, Springer US, Boston, MA, USA, 1981.

[42] P. Liu, L. Duan, X. Chi, and Z. Zhu, "An improved fuzzy C-means clustering algorithm based on simulated annealing," in *Proceedings of the 2013 10th International Conference on Fuzzy Systems and Knowledge Discovery (FSKD)*, pp. 39–43, Shenyang, China, July 2013.

[43] S. Askari, "Fuzzy C-Means clustering algorithm for data with unequal cluster sizes and contaminated with noise and outliers: review and development," *Expert Systems with Applications*, vol. 165, Article ID 113856, 2021.

[44] J. Pei, X. Yang, X. Gao, and W. Xie, "Weighting exponent m in fuzzy C-means (FCM) clustering algorithm," *Proceedings of the SPIE 4554, Object Detection, Classification, and Tracking Technologies*, vol. 4554, pp. 246–251, 24 September 2001.

[45] K.-L. Wu, "Analysis of parameter selections for fuzzy c-means," *Pattern Recognition*, vol. 45, no. 1, pp. 407–415, 2012.

[46] A. Almomany, A. Al-Omari, A. Jarrah, M. Tawalbeh, A. Alqudah, and A. Alqudah, "An OpenCL-based parallel acceleration of aSobel edge detection algorithm Using IntelFPGA technology," *South African Computer Journal*, vol. 32, no. 1, pp. 3–26, 2020.

[47] A. Jarrah, A. Almomany, A. M. R. Alsobeh, and E. Alqudah, "High-performance implementation of wideband coherent signal-subspace (CSS)-Based DOA algorithm on FPGA," *Journal of Circuits Systems and Computers*, vol. 30, Article ID 2150196, 2021.

[48] A. Abedalmuhdi, B. E. Wells, and K. I. Nishikawa, "Efficient particle-grid space interpolation of an FPGA-accelerated particle-in-cell plasma simulation," in *Proceedings of the 2017 IEEE 25th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pp. 76–79, Napa, CA, USA, April 2017.

[49] S. M. Trimberger and J. J. Moore, "FPGA security: motivations, features, and applications," *Proceedings of the IEEE*, vol. 102, no. 8, pp. 1248–1265, Aug. 2014.

[50] T. Wang, C. Wang, X. Zhou, and H. Chen, "An overview of FPGA based deep learning accelerators: challenges and opportunities," in *Proceedings of the 2019 IEEE 21st International Conference on High Performance Computing and Communications; IEEE 17th International Conference on Smart City; IEEE 5th International Conference on Data Science and Systems (HPCC/SmartCity/DSS)*, pp. 1674–1681, Zhangjiajie, China, August 2019.

[51] X. Yang, T. Levi, and T. Kohno, "Real-time pattern recognition implementation on FPGA in multi-SNNs," in *Proceedings of the International Conference on Artificial Life and Robotics*, vol. 25, pp. 151–154, Beppu, Japan, Jan 2020.

[52] H. M. Waidyasooriya, M. Hariyama, and K. Uchiyama, "Design of FPGA-based computing systems with OpenCL," *Design of FPGA-Based Computing Systems with openCL*, Springer International Publishing, New York, NY, USA, 2018.

[53] A. M. Jamel and B. Akay, "A survey and systematic categorization of parallel K-means and fuzzy-c-means algorithms," *Computer Systems Science and Engineering*, vol. 34, no. 5, pp. 259–281, 2019.

[54] M. Alandoli, M. Shehab, M. Al-Ayyoub, Y. Jararweh, and M. Al-Smadi, "Using GPUs to speed-up FCM-based community detection in Social Networks," in *Proceedings of the 2016 7th International Conference on Computer Science and Information Technology (CSIT)*, pp. 1–6, Amman, Jordan, July 2016.

[55] M. Al-Ayyoub, Q. Yaseen, M. A. Shehab, Y. Jararweh, F. Albalas, and E. Benkhelifa, "Exploiting GPUs to accelerate clustering algorithms," in *Proceedings of the 2016 IEEE/ACS 13th International Conference of Computer Systems and Applications (AICCSA)*, pp. 1–6, Agadir, Morocco, 2016.

[56] S. A. A. Shalom, M. Dash, and M. Tue, "Graphics hardware based efficient and scalable fuzzy C-means clustering," in *Proceedings of the Seventh Australasian Data Mining Conference (AusDM 2008)*, J. F. Roddick,, J. Li,, P. Christen,, and P. J. Kennedy,, Eds., ACS, Glenelg, South Australia, pp. 179–186, 2008.

[57] N. Ait Ali, B. Cherradi, A. El Abbassi, O. Bouattane, and M. Youssfi, "GPU fuzzy c-means algorithm implementations: performance analysis on medical image segmentation," *Multimedia Tools and Applications*, vol. 77, no. 16, pp. 21221–21243, 2018.

[58] T. Kalaiselvi and P. Sriramakrishnan, "Rapid brain tissue segmentation process by modified FCM algorithm with CUDA enabled GPU machine," *International Journal of Imaging Systems and Technology*, vol. 28, no. 3, pp. 163–174, 2018.

[59] A. Shoeibi, N. Ghassemi, H. Hosseini-Nejad, and M. Rouhani, "An efficient brain MR images segmentation hardware using kernel fuzzy C-means," in *Proceedings of the 2019 26th National and 4th International Iranian Conference on Biomedical Engineering (ICBME)*, pp. 93–99, Tehran, Iran, November 2019.

[60] M. Krid, M. Karray, and D. S. Masmoudi, "FPGA pulse mode implementation of a Gaussian Fuzzy C-Means algorithm," in *Proceedings of the 2015 IEEE 12th International Multi-Conference on Systems, Signals & Devices (SSD15)*, pp. 1–6, Mahdia, Tunisia, March 2015.

[61] J. Lázaro, J. Arias, J. L. Martín, C. Cuadrado, and A. Astarloa, "Implementation of a modified Fuzzy C-Means clustering algorithm for real-time applications," *Microprocessors and Microsystems*, vol. 29, no. 8-9, pp. 375–380, 2005.

[62] W. J. Hwang, Z.-C. Fan, and T.-M. Shen, "Unsupervised image segmentation circuit based on fuzzy C-means clustering," in *Proceedings of the Fifth International Conference on Advances in Circuits, Electronics and Micro-electronics*, Venice, Italy, October 2012.

[63] T. Zhang, R. Ramakrishnan, and M. Livny, "A new data clustering algorithm and its applications," *Data Mining and Knowledge Discovery*, vol. 1, no. 2, pp. 141–182, 1997.

[64] I. Kärkkäinen and P. Fränti, *A Dynamic Local Search Algorithm for the Clustering Problem*, Joensuu, Finland, University of Joensuu, 2002.

[65] M. Rezaei and P. Franti, "Set matching measures for external cluster validity," *IEEE Transactions on Knowledge and Data Engineering*, vol. 28, no. 8, pp. 2173–2186, 1 Aug. 2016.

[66] R. Brandao, *Fuzzy-k-Means: C++ Implementation of Fuzzy K-Means Clustering Algorithm*, 2016, https://github.com/programonauta/fuzzy-k-means.

[67] M. Parker, *Understanding peak floating-point performance claims*, intel programmable solutions group, 2017.

[68] D. Terpstra, H. Jagode, H. You, and J. Dongarra, *Collecting Performance Data with PAPI-C", Tools for High-Performance Computing 2009*, pp. 157–173, Springer Berlin/Heidelberg, 3rd Parallel Tools Workshop, Dresden, Germany, 2010.