sensors

MDPI

Article

# A Parallel Architecture for the Partitioning Around Medoids (PAM) Algorithm for Scalable Multi-Core Processor Implementation with Applications in Healthcare

**Hassan Mushtaq [1], Sajid Gul Khawaja [2], Muhammad Usman Akram [2,*], Amanullah Yasin [1], Muhammad Muzammal [3,*], Shehzad Khalid [4] and Shoab Ahmad Khan [2]**

[1] Department of Electrical & Computer Engineering, Sir Syed CASE Institute of Technology, Islamabad 44000, Pakistan; hassan.mushtaq@ymail.com (H.M.); amanyasin@case.edu.pk (A.Y.)
[2] Department of Computer & Software Engineering, CE&ME, National University of Sciences & Technology, Islamabad 44000, Pakistan; sajid.gul@ceme.nust.edu.pk (S.G.K.); shoabak@ceme.nust.edu.pk (S.A.K.)
[3] Department of Computer Science, Bahria University, Islamabad 44000, Pakistan
[4] Department of Computer Engineering, Bahria University, Islamabad 44000, Pakistan; shehzad@bahria.edu.pk
* Correspondence: usmakram@gmail.com (M.U.A.); muzammal@bui.edu.pk (M.M.)

Received: 15 August 2018; Accepted: 1 November 2018; Published: 25 November 2018

**Abstract:** Clustering is the most common method for organizing unlabeled data into its natural groups (called clusters), based on similarity (in some sense or another) among data objects. The Partitioning Around Medoids (PAM) algorithm belongs to the partitioning-based methods of clustering widely used for objects categorization, image analysis, bioinformatics and data compression, but due to its high time complexity, the PAM algorithm cannot be used with large datasets or in any embedded or real-time application. In this work, we propose a simple and scalable parallel architecture for the PAM algorithm to reduce its running time. This architecture can easily be implemented either on a multi-core processor system to deal with big data or on a reconfigurable hardware platform, such as FPGA and MPSoCs, which makes it suitable for real-time clustering applications. Our proposed model partitions data equally among multiple processing cores. Each core executes the same sequence of tasks simultaneously on its respective data subset and shares intermediate results with other cores to produce results. Experiments show that the computational complexity of the PAM algorithm is reduced exponentially as we increase the number of cores working in parallel. It is also observed that the speedup graph of our proposed model becomes more linear with the increase in number of data points and as the clusters become more uniform. The results also demonstrate that the proposed architecture produces the same results as the actual PAM algorithm, but with reduced computational complexity.

**Keywords:** clustering; partitioning around medoids; scalable; parallel; reconfigurable; FPGA; MPSoCs; multi-core processor; time complexity; speedup

## 1. Introduction

In recent years, the Internet of Things (IoT) has rapidly grown into one of the most beneficial and dominant communication models for wireless communication applications [1–4]. Our everyday life is becoming associated with IoT-based entities where the Internet provides a computational platform for data gathered from various sensors. The IoT has hence enhanced the horizon of the Internet making it suitable for different applications. Health care applications in particular have risen

in popularity because of rapid development of IoT-based wireless sensor networks, medical devices and wireless technologies [5–7].

Healthcare applications mainly deal with large sets of data that need to be classified into various classes. It frequently happens that the output classes are not known a priori, thus unsupervised learning models such as clustering are preferred. Clustering is one of the basic techniques in the data mining domain aimed at organizing a set of data objects into their natural subsets (called clusters), when labels for the data are unavailable. An ideal cluster is an isolated set of data points which are similar to one another, but dissimilar to the points in other clusters [8,9]. Similarity is commonly defined in terms of how close the objects are and is based on a specified distance metric [8].

The most fundamental way of clustering is partitioning, which organizes the data into several exclusive groups. More formally, given a set of $N$ objects, a partitioning algorithm makes $K$ partitions of the data, where each partition represents a cluster. A data point is assigned to a cluster based on the minimum distance measure between the point and the cluster center. The cluster centers are initially chosen either randomly or by using some initialization technique. The algorithm then iteratively improves the formulation of clusters by finding new cluster centers using the objects assigned to the clusters in the previous iteration. All the objects are then reassigned to clusters using updated cluster centers. This process continues until there is no change in the calculated cluster centers in the current iteration compared to the previous one [8]. Clustering has been used in many applications around the globe encompassing many of our real-life applications, including but not limited to healthcare, IoT, academics, search engines, wireless sensor networks, etc. [10–18].

Partitioning-based clustering algorithms differ in the way of computing cluster centers, e.g., K-Means [19,20] and K-Modes [21] clustering algorithms use mean and mode values of the clustered data points, respectively, while in K-Medoids [8], clustering method clusters are characterized by their most centrally located objects (called medoids). The first practical realization of the K-Medoids method was introduced as the Partitioning Around Medoids (PAM) algorithm. PAM is more robust than K-Means against noise and outliers, but this robustness comes at the expense of more computations.

Considering a PAM algorithm which is clustering a dataset with $n$ points into $k$ clusters, the time complexity required for it to complete its task is roughly $O(k(n-k)^2)$. This makes the algorithm resource and time intensive, especially in the present age where huge amounts of data are at our disposal for processing, thus it becomes a major bottleneck for real time implementation. The utility of the PAM algorithm demands an implementation of the PAM algorithm which is computationally less intensive. This timing complexity of PAM algorithm can be reduced by the introduction of parallel processing and multi-core solutions to the problem. Furthermore, such as design is also suited to today's state-of-the art multicore and reconfigurable computing hardware platforms such as Graphics Processing Units (GPUs) and Field Programmable Gate Arrays (FPGAs), essentially leading to a computationally inexpensive implementation of the PAM algorithm targeted for real-time processing. The use of FPGAs and reconfigurable computing in faster implementations of various algorithms has gained popularity in recent years. They have been used in various domains ranging from biomedical image/signal processing to data analytics and more [22–24]. FPGAs along with SoC designs have also been used extensively in optimizing various algorithms, such as deep learning and HoG-based segmentation, for application development [25–30].

In this paper we present a scalable parallel architecture of the PAM algorithm which can be implemented either on a multi-core processor system or on a reconfigurable device. The proposed algorithm makes use of divide and conquer approach using multiple processing units. The multiple cores perform homogeneous operations on independent set of data and share results with each other to finalize a single iteration.

The remainder of the paper is organized as follows: Section 2 provides an overview of the background related to acceleration of the PAM algorithm, Section 3 provides an overview of the PAM algorithm and our proposed architecture of parallel PAM. The results are presented and thoroughly discussed in Section 4, followed by the conclusions of the paper in the last section.

## 2. Background

The complexity of the PAM algorithm has forced researchers to come up with various modifications in order to speed up the algorithm. In this respect a basic multi-core implementation of K-Medoids was proposed which divides the algorithm into sub-tasks, where each sub-task is implemented on a separate core. The proposed design provides improved speedup (4× while utilizing 16 cores) but is limited by the number of hardware cores available at a user's disposal. A similar multi-core solution has been provided by Rechkalov in [31] for the Intel Xeon Phi Many-Core Coprocessor. The proposed system makes use of OpenMP parallelizing technology and loop vectorization within the algorithm along with the tiling approach. Experimentation showed that the performance of the optimized version of the algorithm is improved but the overall performance depends on the nature of the data to be clustered [31]. Velmurugan et al. discussed the performance improvements that can be made in K-Medoids by simply changing how the data is distributed. In order to prove this concept, normal and uniform distributions of data were used showing that execution time varies as the selected distribution changes [32]. Park et al. proposed an alternate way of computing the K-Medoids algorithm which is similar to the K-Means algorithm [33]. In the proposed solution Euclidean distance is calculated once and then used to calculate new medoids at each iteration. Experiments showed that the execution time of the algorithm was reduced significantly in comparison to the original algorithm.

The increase in popularity of reconfigurable devices, multi-core systems and Graphic Processing Units (GPUs) has seen researchers focusing their efforts on finding parallel models of various machine learning algorithms to improve their performance. A scalable multi-core hardware architecture for implementation of K-Means is proposed in [14] which makes use of tiling to perform multiple tasks in parallel (cores) to speed up the system. The cores are further interconnected using a Network-on-Chip (NoC) interconnect network to offload traffic and provide scalability to the design by minimizing the message passing bottlenecks. The experimentation yielded a near linear speedup with increase in number of cores while the hardware resources and clock speed were not affected. Similarly, a parallel processing model for implementation of the mean shift clustering algorithm for FPGA implementation was proposed by Tehreem et al. in [34]. The proposed model consists of homogenous processing cores running in parallel on independent data subsets. These cores were connected through a bus for data sharing. The proposed model worked in a collaborative working environment where each core works independently and shares its data with others for finalization of results. This model provided a significant speedup while utilizing only 10.31% of the total device slice registers and 33% of total slice LUTs of a Spartan 6 FPGA. A multi-processor architecture having heterogeneous tiles for real time image processing was proposed in [28]. Each tile of the proposed architecture provided computational and memory capabilities. These tiles were connected via a novel NoC structure named Spidergon. The design provided support for different algorithmic classes and ran at 400 MHz ensuring real time processing of up to 30 VGA frames/s [35]. Similar efforts have been made to an optimized hardware design of the Particle Swarm Optimization (PSO) algorithm by Mehmood et al. in [36]. A modified PSO-based technique based on multi-core sequential architecture is presented in the paper. The processing cores implementing the sequential architecture were connected via NoC for implementing a parallel architecture. The architecture was benchmarked against a pure software-based implementation indicating an average speed-up of 22.53 and 29.37 for non-NoC-based HMPSO and a NoC-based MPSO, respectively, over 25 experiments [32]. Li et al. proposed in [37] an efficient error rate, in a proposed VLSI architecture of FCM with Spatial constraints (FCM-S) for image segmentation. To lower the segmentation architecture, the spatial information is used during the FCM training process. In addition, the architecture employs a high throughput pipeline to enhance the computation speed. Experimental results revealed that the proposed architecture implemented on a SoPC architecture attains a speedup of up to 342.51 over its software counterpart. The proposed architecture therefore is an effective alternative for applications requiring real-time image segmentation and analysis.

Existing PAM algorithms and their respective architectures are not that scalable and have an upper bound on the reduction in computational complexity they can achieve. In the proposed system,

we present a scalable parallel architecture of PAM algorithm which can exponentially reduce the computational complexity. It introduces the concept of working in a collaborative environment approach by dividing data into multiple processing units which perform homogeneous operations independently and finally give a combined result.

## 3. Theory and Design of the Parallel PAM Algorithm

### *3.1. Overview of the PAM Algorithm*

We will use the following notation to formally describe the PAM algorithm. Let $X = \{x_1, x_2, x_3, ..., x_N\}$ be the set of $N$ data points to be clustered where each data point consists of $d$ real-valued attributes. Let $M = \{m_1, m_2, m_3, ..., m_K\}$ is a set of $K$ medoids such that $M \subset X$, where $K$ is the number of clusters such that $K << N$. $D: X \times M \rightarrow R$ is a distance metric, usually it is a matrix of Euclidean distances from each object $x_i$ to its nearest medoid $m_j$. In each iteration of algorithm, a pair of medoid object $m_j$ and non-medoid object $x_i$ is selected which produces the best clustering when their roles are swapped. The objective function used is the sum of the distances from each object to its closest medoid:

$$Cost = \sum_{i=1}^{N} \min_{1 \le j \le K} D(x_i, m_j) \quad (1)$$

The algorithm PAM proceeds in the following manner:

i. In the first phase (called Build Phase) an initial clustering is obtained by the successive selection of $K$ medoids. The first medoid is the one for which the sum of distances to all non-medoid objects is minimum. This is actually the most centrally located data point in set $X$. Subsequently, at each step another object is selected as a medoid, for which the objective function is minimum. The process is continued until $K$ medoids have been found.
ii. In the second phase of the algorithm (called Swap Phase), it is attempted to improve the set $M$ of medoids and therefore the clustering obtained by this set. The algorithm goes through each pair of objects ($m_j$, $x_h$), where $m_j$ is a medoid and $x_h$ is non-medoid object and $x_h$ belongs to cluster $j$. The effect on the objective function is determined when a swap is carried out i.e., when object $x_h$ is considered as a medoid in place of object $m_j$. For each cluster $j$, the object $x_h$ is selected as its new medoid for which the objective function is minimized and thus the set $M$ is updated. This process is iterated until no further decrease in objective function value is possible or in other words there is no update in set $M$ between two consecutive iterations.

### *3.2. Proposed Design Flow*

The aim of this paper is to propose a model to make PAM algorithm computationally less expensive by parallelizing its functionality such that it uses less resources when implemented on reconfigurable hardware. The whole working of our research revolves around the concept that how well we parallelize the PAM algorithm so that its overall computational complexity can be significantly reduced. We concluded that this task can be performed well by following these steps:

1. Dividing the algorithm into a well-defined sequence of subtasks.
2. Identifying the portions of the algorithm which can be executed in parallel.
3. Running these subtasks for equal subsets of data simultaneously on multiple homogeneous cores.
4. Combining the intermediate results from different PEs to produce a final clustering.

#### 3.2.1. Sub-Tasking of PAM Algorithm

As we discussed in the previous section, the PAM consists of two phases called (1) Build Phase and (2) Swap Phase. The complete flow chart of the Partitioning Around Medoids algorithm in terms of this is shown in Figure 1.

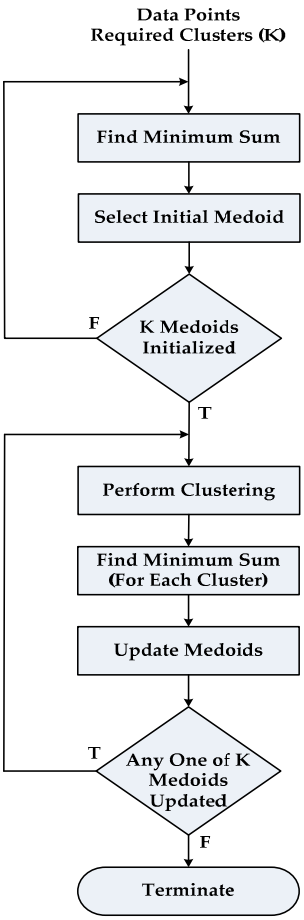



**Figure 1.** Flow chart of the sequential PAM algorithm.

The peudo-code of PAM is given in Algorithm 1 below.

**Algorithm 1.** Pseudo-code of PAM

**Procedure:** Partitioning Around Medoids (PAM)
**Input:** *K* (No. of clusters), *X* (Data Set)
**Output:** *C* (Vector containing cluster tags against each data object), *M* (Medoids)
1. Initialize *M* **/* Build Phase */**
2. **repeat** **/* Swap Phase */**
3. Find clusters
4. Perform swapping and update Medoids
5. **until** no update in any of *K* Medoids

We start by splitting the working Build Phase into two subtasks namely: (1) Find Minimum Sum and (2) Select Initial Medoid. In the first subtask, one by one each object is temporarily selected as a medoid and the minimum value of the objective function is computed for the set of medoids selected up to current step. The second subtask selects the temporary medoid as the actual medoid for which the objective function value is minimum. Algorithm 2 depicts the pseudo-code of the Build Phase.

Then the Swap Phase is split into three subtasks called: (1) Perform Clustering, (2) Find Minimum Sum (For Each Cluster) and (3) Update Medoid. The first Subtask assigns each object to its closest medoid to form clusters. In the second subtask, one by one the role of each object within a cluster is swapped with its medoid and smallest value of the cost function is computed.

**Algorithm 2.** Pseudo-code of the Build Phase

```
1. repeat
2.    for a := 1 → N do                                   /* Find Minimum Sum */
3.        Select Xa as temporary Medoid
4.        for i := 1 → N do
5.            for each Medoid selected yet (including Xa) do
6.                Find the minimum Euclidean distance b/w Xi and Medoid
7.            endfor
8.            Find sum of minimum distances
9.        endfor
10.       Find minimum sum
11.   endfor
12.   Select Xa as actual Medoid for which the sum is minimum        /* Select Initial Medoid */
13. until 'K' initial Medoids are selected
```

The last subtask of this phase updates the medoid of the current cluster for which the cost function value is minimum. Subtasks 2 and 3 are repeated for all clusters. The corresponding pseudo-code is shown as Algorithm 3.

**Algorithm 3.** Pseudo-code of the Swap Phase

```
1. repeat
2.    for i := 1 → N do                                   /* Perform Clustering */
3.            for j := 1 → K do
4.                Find Euclidean distance b/w Xi and Mj
5.        Tag Xi with j for which this distance is minimum
6.    endfor
7.    endfor
8.    for j := 1 → K do                          /* Find Minimum Sum (For Each Cluster)*/
9.        for each data point Xa ϵ cluster j do
10.           for each data point Xi ϵ cluster j do
11.               Find sum of Euclidean distances b/w Xa and Xi
12.           endfor
13.           Find minimum sum
14.       endfor
15.       Update Xa as jth Medoid for which the sum is minimum        /* Update Medoid*/
16.   endfor
17. until no update in any of 'K' Medoids
```

Now in order to reduce the computational complexity and improve the execution time, the PAM algorithm needs parallelism. We have identified from Algorithm 2 that the subtask 1 of Build Phase can be executed in parallel on multiple PEs for equal subsets of data, while subtask 2 will compute the final result of this phase. Similarly, it is clear from Algorithm 3 that subtasks 1 and 2 of the Swap Phase can be parallelized well.

### 3.2.2. Paralleling the PAM Algorithm and Proposed Architecture

Our proposed architecture uses *P* number of homogeneous cores or Processing Elements (PEs) which are connected through an interconnect network such as a bus as shown in Figure 2. Each PE has given access to all data points thus it can work in parallel with other PEs to achieve faster convergence and eventually an increased throughput. A Global Control Unit is used to control the overall flow of the algorithm. The interconnecting network used in the design can be a bus-based, point to point or network-on-chip-based interface. The choice of interconnection is based on the

complexity and requirement of the applications, e.g., a NoC based interface will provide better concurrent message passing at the cost of area and power overhead.

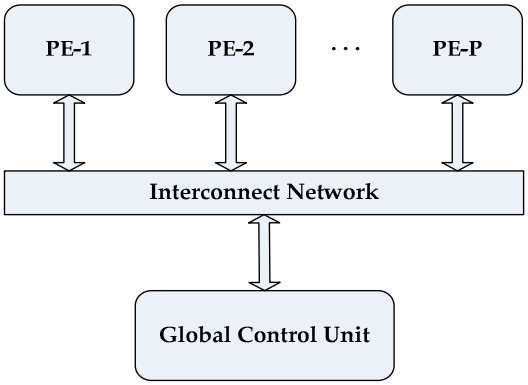



**Figure 2.** Top level block diagram of the proposed architecture.

The overall working of the parallel PAM for this multi-core processor model is described in the following steps:

1. A data set of size $N$ is made completely divisible into the number of available cores $P$ by appending zeros at the end of the data set so that equal subsets can be assigned to each core.
2. The complete data set $X$ is replicated in all available PEs and equal partitions of $X$ are assigned to each PE.
3. Each PE then executes the subtask "Find Minimum Sum" of the Build Phase for its respective data subset of size $\frac{N}{P}$ in parallel. Master PE (any processing element can be assigned to perform as master PE because all PEs are homogenous) will collect the results of the first subtask from each PE and perform the subtask "Select Initial Medoid". This step is repeated until $K$ medoids are initialized, as described in Algorithm 4 below.
4. Final results of Build Phase are sent to all PEs so that they can proceed to the next phase of algorithm.
5. Each PE will tag all its assigned data objects with their closest cluster numbers. These tags are stored in local memory associated with each data object. All PEs one by one broadcast their $\frac{N}{P}$ tags over the interconnect network so that each PE can have complete result of clustering.
6. The subtask "Find Minimum Sum (For Each Cluster)" of the Swap Phase is executed by each core in parallel. A master PE will perform the subtask "Update Medoid" after receiving results from other PEs. Steps 5 and 6 are repeated until no update in any of $K$ medoids is reported. Algorithm 5 depicts the working of the Swap Phase in case of parallel PAM.

**Algorithm 4.** Pseudo-code of the Build Phase for Parallel PAM

```
1. repeat
2.    for p := 1 → P do in parallel
3.        for each data point Xa ϵ p do                    /* All PEs do in parallel*/
4.            Select Xa as temporary Medoid
5.            for i := 1 → N do
6.                for each Medoid selected yet (including Xa)do
7.                    Find the minimum Euclidean distance b/w Xi and Medoid
8.                endfor
9.                Find sum of minimum distances
10.           endfor
11.           Find minimum sum
12.       endfor
13.   endfor
14.   for p := 1 → P do                                   /* Only master PE will do this */
```

15. Select $X_a$ as actual Medoid for which the sum is minimum from all PEs
16. **endfor**
17. **until** 'K' initial Medoids are selected

**Algorithm 5.** Pseudo-code of the Swap Phase for Parallel PAM

```
1. repeat
2.    for p := 1 → P do in parallel
3.       for each data point Xi ϵ p do                    /* All PEs do in parallel*/
4.          for j := 1 → K do
5.             Find Euclidean distance b/w Xi and Mj
6.             Tag Xi with j for which this distance is minimum
7.          endfor
8.       endfor
9.    endfor
10.   Concatenate clustering results from all PEs          /* All PEs do this*/
11.   for p := 1 → P do in parallel
12.      for j := 1 → K do                                /* All PEs do in parallel*/
13.         for each data point Xa ϵ p & cluster j do
14.            for each data point Xi ϵ cluster j do
15.               Find sum of Euclidean distances b/w Xa and Xi
16.            endfor
17.            Find minimum sum for each cluster
18.         endfor
19.      endfor
20.   endfor
21.   for j := 1 → K do                                   /* Only master PE will do this */
22.      for p := 1 → P do
23.         Update Xa as j^th Medoid for which the sum is minimum from all PEs
24.      endfor
25.   endfor
26. until no update in any of 'K' Medoids
```

The complete work flow of the parallel PAM algorithm is depicted in Figure 3 below.

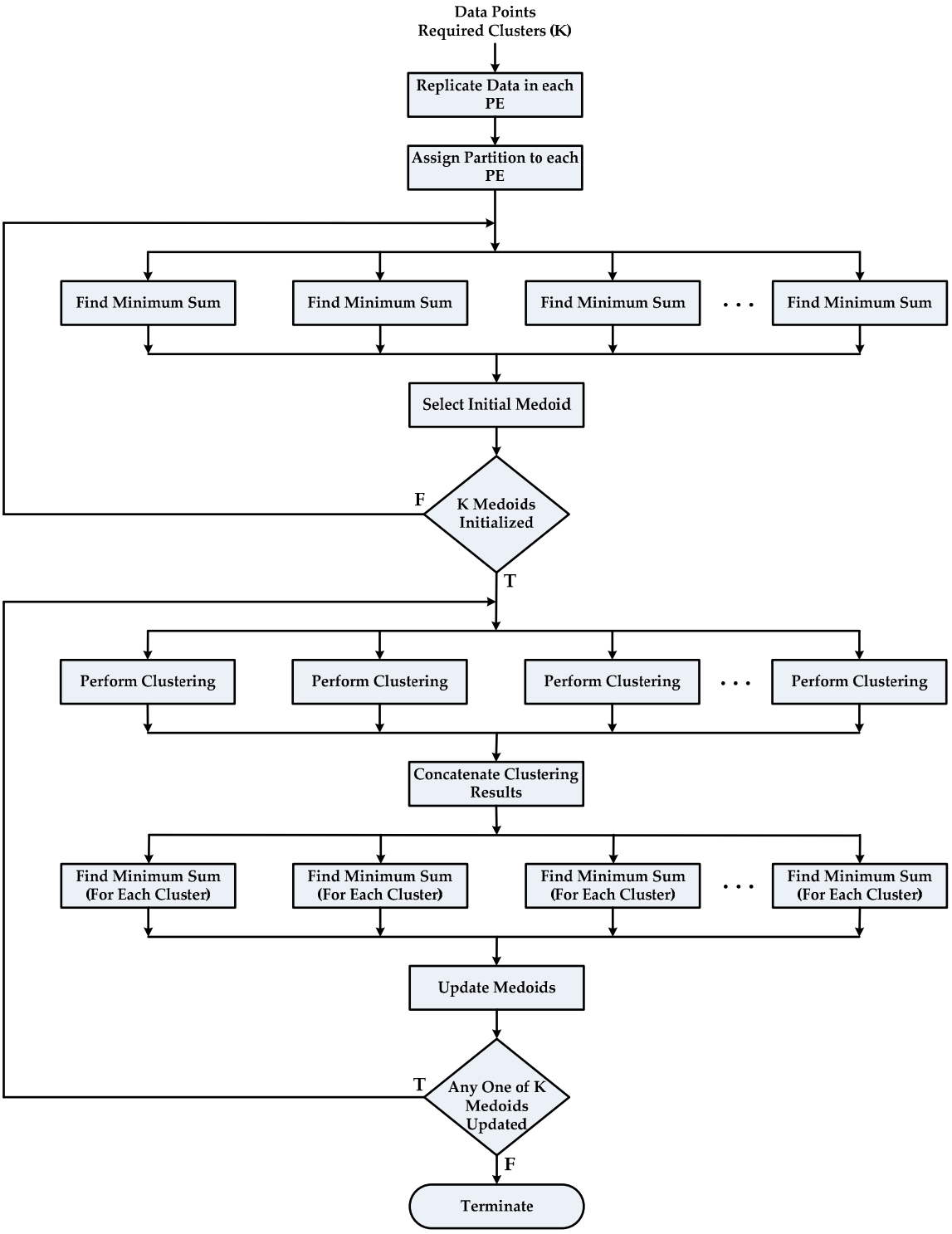



**Figure 3.** Flow chart of the parallel PAM algorithm.

At this stage we can explore the internal structure of a processing element at an abstract level. As shown in Figure 4, each PE is composed of three sections (1) Controller, (2) Datapath and (3) Memory. The Controller section consists of a local control unit to manage the overall sequencing of subtasks within a processing element and to manage communication with other PEs. The Datapath section contains sub-blocks which are basically hardware sub-modules implementing the functionality of different sub-tasks of the algorithm. Finally, each core has a memory section which consists of a memory block of size $N \times d$ to hold the complete data set, a memory block of size $M \times d$ to store final values of medoids and an $N \times 1$ sized block of memory to store cluster tags against each data object.

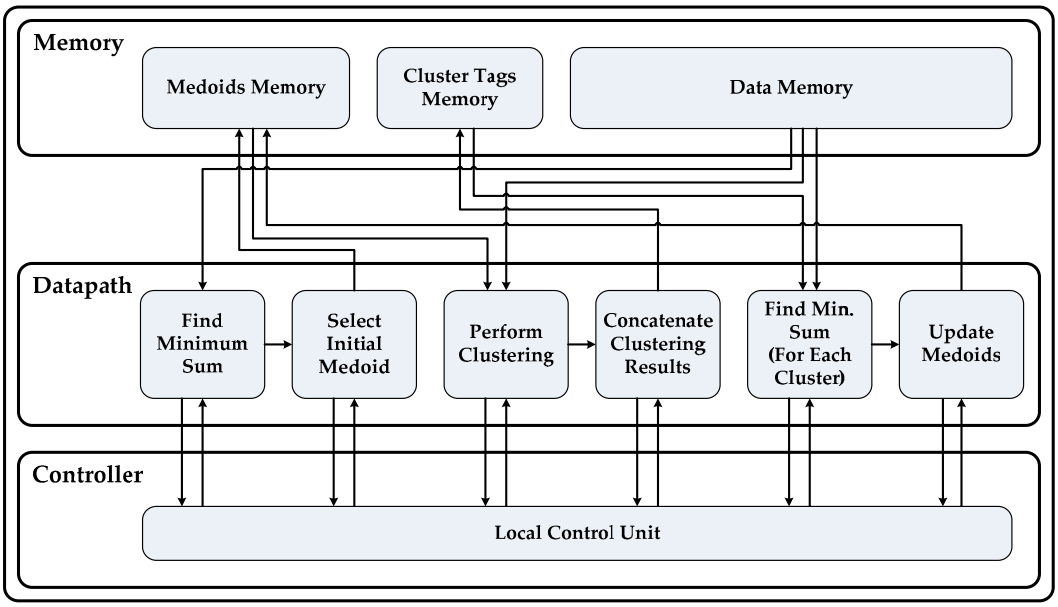



**Figure 4.** Internal structure of a processing element (PE).

## 4. Experimentation and Results

In order to demonstrate the usefulness of our proposed parallel implementation of PAM, first we implemented the sequential PAM algorithm as described in the previous section and recorded the running time of the algorithm in terms of number of computations required by both the build and swap phases. Then the running time in the case of parallel implementation of PAM was computed along similar lines to examine the speedup attained for different numbers of PEs. In the latter case, the time required for communication among different PEs, to share results and data, was also added to get the total running time.

First, this experimentation was performed for randomly generated artificial data points ($N$ = 400 to 1600) each having two attributes ($d$ = 2). At the second stage, color-based segmentation of different RGB images was performed. Sizes were around 7000 pixels to 16,000 pixels for different images and value of $d$ is 3 in this case (three color components). After thorough experimentation, the following results were concluded.

1. The time complexity $n$ of the algorithm reduces exponentially as we increase the number of cores for the same data set or image. Here by time complexity we mean the running time which is taken by all computations required by the build and swap phases. This computation complexity reduces as we increase the number of processing entities and divide the computations among them. For example, the running time of PAM algorithm for $N$ = 800, $d$ = 2, $K$ = 4 and $P$ = 1, is $n \approx 1.41 \times 10^7$. For $P$ = 2, this value is half of the previous value i.e., $n \approx 7.1 \times 10^6$ plus a small communication overhead = 4948. Similarly, for $P$ = 4, $n \approx 3.5 \times 10^6$ plus overhead is 3416 and so on. Figure 5 shows this trend for both the artificial data set of size 800 and an image of 9720 pixels in size. Furthermore, it is observed that when the size of the data is increased this trend becomes more uniform and gets close to $\frac{n_1}{P}$, where $n_1$ is the computational complexity of the sequential algorithm. This is evident from Figure 5a,b where the trend remains the same even if we increase the data points which need to be clustered.
2. The speedup of the algorithm is defined as $S_p = \frac{n_1}{n_p}$, where $n_1$ is running time of the algorithm for a single PE and $n_p$ is the running time for $P$ processing elements. It was observed that the speedup of the parallel PAM algorithm increases with the increase in the number of processing elements, but this increasing trend varies slightly for different scenarios discussed below.

   (1) The speedup graph gets more linear as the data size increases for the same number of PEs. This trend is shown in Figure 6.

(2) If the clusters to be formed are uniform i.e., the number of data objects is equal in each cluster then the speedup attained is slightly better than the case when clusters are non-uniform for same data set size.

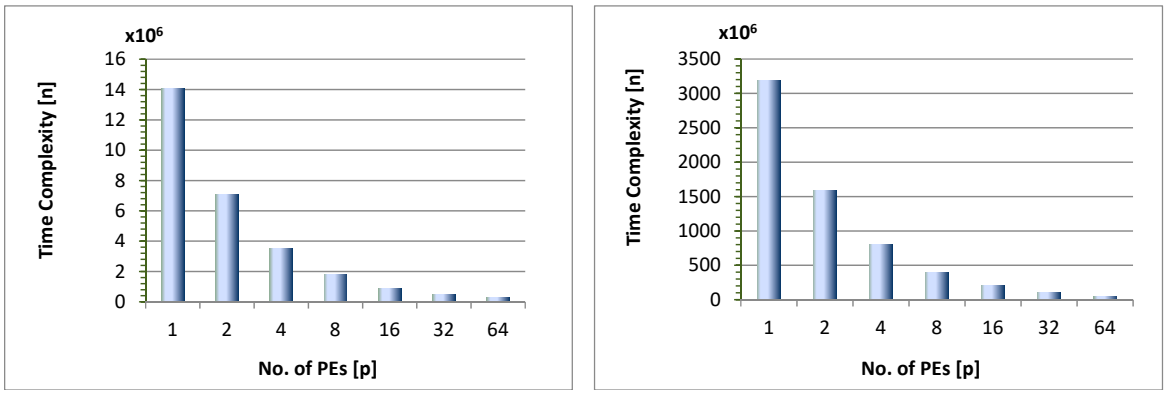



**Figure 5.** Computational complexity of the proposed architecture: (**a**) For artificial random data, $N$ = 800, $d$ = 2 & $K$ = 4; (**b**) For image pixels, $N$ = 9720, $d$ = 3 & $K$ = 4.

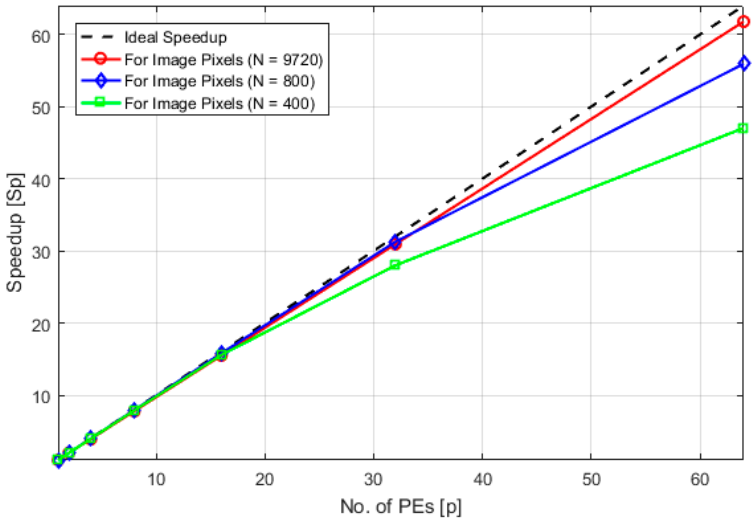



**Figure 6.** Comparison of speedup for different dataset sizes.

As we know PAM consists of two discrete phases, the first one is the Build Phase which is just an initialization method for medoids while the actual algorithm which iteratively runs and tries to minimize the objective function is the Swap Phase. The Build Phase is computationally more expensive than the Swap Phase but on the other hand it significantly increases the probability of convergence of the PAM algorithm. Other different methods of initialization can also be used, the simplest one of which would be random initialization of medoids, but at the cost of a decrease in the probability of convergence.

(1) It was observed that due to the high computation cost of the Build Phase, the total computation cost of the algorithm (order of $N^2$) is much higher than the communication cost (order of $N$), for large values of $N$. Therefore, communication overhead doesn't affect the speedup and it is almost equal to the number of PEs.

(2) If we don't include the effect of build phase or medoids are randomly initialized then communication overhead will affect the speedup achieved by our parallel PAM algorithm, otherwise this trend will be near linear, as shown in Figure 7.

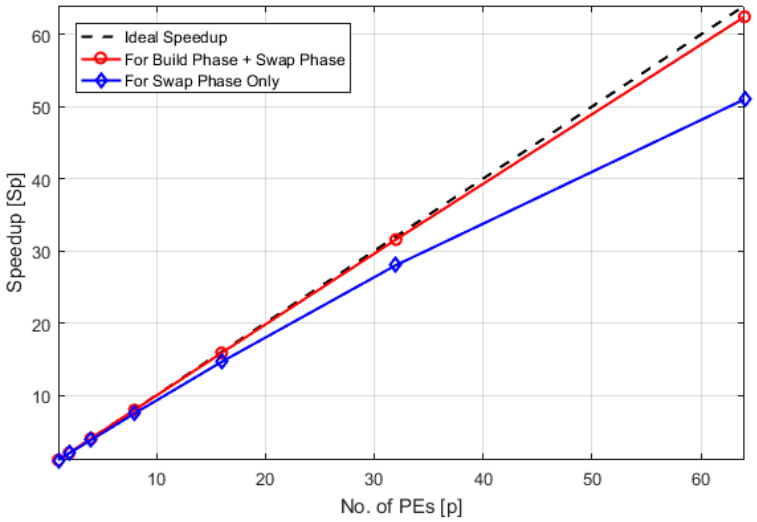



**Figure 7.** Comparison of speedup for the case when medoids are initialized using the build phase with the case when medoids are randomly initialized, data size is same in both cases.

We have also studied the effect of cluster size on speedup. If the clusters to be formed have equal sizes, i.e., the number of data objects is equal in each cluster then the speedup graph is slightly better than the case when all clusters have different sizes. This effect is shown in Figure 8a for clustering of an RGB image of size $N = 9720$ pixels when total number of computations including both Build Phase and Swap Phase is taken into account. When the effect of only Swap Phase is considered then there is a prominent difference between speedup achieved by the algorithm for equal-sized and unequal-sized clusters as opposed to the previous case when the effect of both phases is considered. These results are shown in Figure 8b.

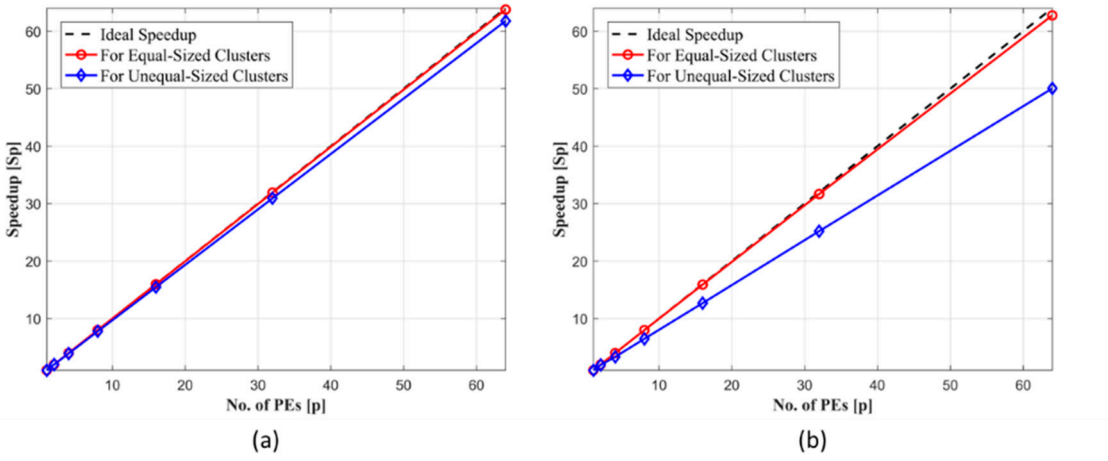
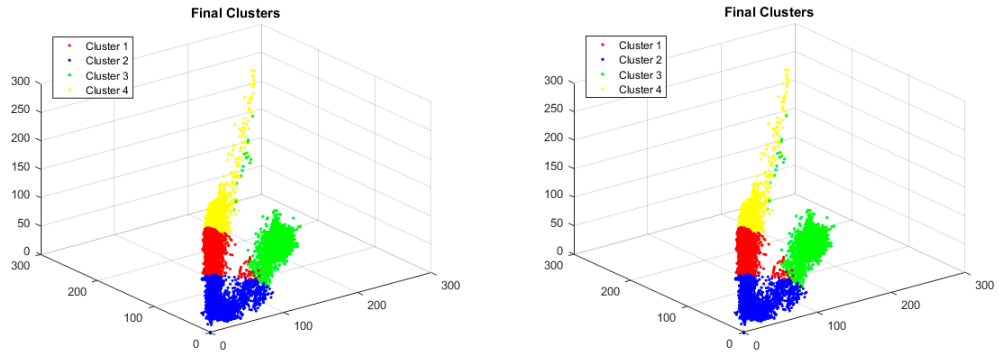



**Figure 8.** (**a**) Comparison of speedup for same size of data but having equal and unequal cluster sizes (**b**) Comparison of speedup for equal and unequal sized clusters but data size is same for both cases. Effect of Build Phase is not included.

Finally, the accuracy of both sequential and parallel implementations of PAM was checked against the results obtained through a well-known but different implementation of the K-Medoids clustering algorithm [17]. This implementation is being used by many researchers in the field of cluster analysis. It was found that the results, such as cost function value, medoids values and cluster tags against each data point, obtained by our implementations and the referenced algorithm for same data input were exactly matched. We have performed color-based segmentation of different images to verify the accuracy of our proposed model. Figure 9 shows a comparison of the final clustering of pixels of an example image in 3-dimensional space.




(**a**) (**b**)

**Figure 9.** Clusters produced by (**a**) our proposed implementation of parallel PAM with $P = 64$ and (**b**) K-Medoids clustering algorithm [17]. Final medoids locations and objective function value were same in both cases.

In order to assess the quality of the output images, the Structural Similarity Index Metric (SSIM) has been calculated. The Structural Similarity Index (SSIM) is an assessment mechanisms which computes the image quality degradation which is caused by processing images [38]. SSIM has been calculated for the output images of this segmentation process and was found to be 1.00 for all images. Some example images and output images are shown in Figure 10. Similarly, SSIM of the proposed architecture was cross-checked on segmentation of various health-related images such as MRI scans of brain tumor, fluorescein angiography of a retina to detect neovascular AMD and MRI scans of legs for detection of fluid collection in calf muscles due to injury which resulted in values of 0.97, 1.0 and 0.93, respectively, as shown in Figure 11. The results indicate that the proposed optimized model can be applied on images for segmentation processes without fear of distortion of the results as is clear from SSIM.

**Figure 10.** Left to right: original image, result of proposed implementation of PAM, result of K-Medoids clustering algorithm [17] (SSIM = 1, for all cases).

In this article, we have designed a generic scalable architecture for K-Medoids clustering which has been tested for randomly generated data sets and color image segmentation. The proposed model is scalable with respect to the number of processing cores depending on the availability of hardware resources. We can increase the parallelism by increasing the number of PEs. The same architecture is also valid for any number of clusters and also for any dimensional data points. The results presented in Figures 6 and 7 use the Euclidean distance metric to find similarity (or dissimilarity) among data objects but this architecture is also tested for other distance metrics such as squared Euclidean, city block etc. it was found that speedup trend is not affected by using different distance metrics.

The proposed architecture is an ideal fit for implementing it on reconfigurable devices such as FPGAs and MPSoCs because of its overall small footprint. Each sub-task is designed in such a way that its internal working can easily be unfolded, while keeping the hardware resources in check, to further reduce the overall processing time of algorithm. The efficiency of the proposed at higher number of PE shows divergence from ideal speedup, this trend can be improved by the use of efficient communication interface such as Network-on-Chip (NoC) for linking of multiple cores. Researchers have provided various platforms for NoC based MPSoC models [39,40]. The use of NoC in multicore models in various application has shown promising results in terms of scalability and efficiency [14,41–45].

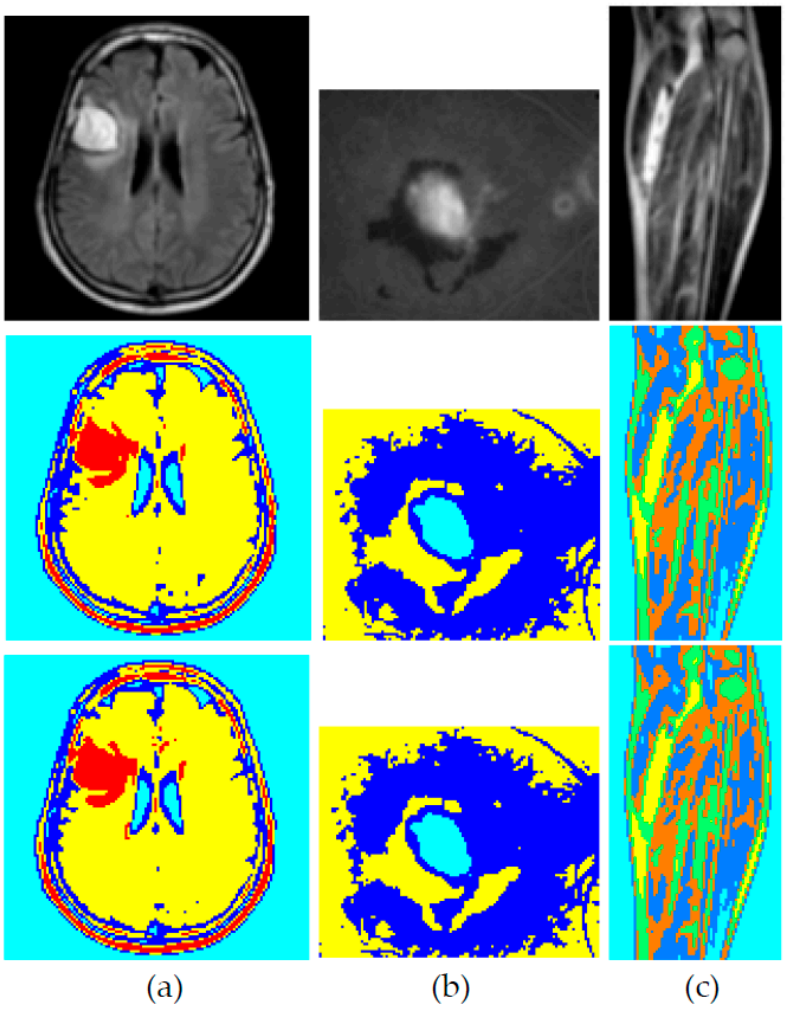



**Figure 11.** Top to bottom: original image, segmentation result of proposed model of PAM algorithm, result of K-Medoids clustering by [17]: (**a**) MRI scan of a brain to detect tumor (T2W image with full brain coverage in axial plane), SSIM = 0.97; (**b**) Fluorescein angiography of a retina to detect neovascular AMD, SSIM = 1.00; (**c**) MRI scan of a leg to detect fluid collection in calf muscles due to injury (T2W image in coronal plane), SSIM = 0.93.

## 5. Conclusions

Clustering is a commonly used platform for labeling of unknown data based on similarity. Among the many clustering scheme variants the Partitioning Around Medoids (PAM) algorithm belongs to the partitioning-based methods of clustering used for numerous applications ranging from object categorization to data compression. In today's world a significant increase in available data for clustering has made clustering algorithms computationally expensive thus they can't be used in their original form for real-time processing. In this paper a scalable multi-core parallel architecture for implementation of PAM is presented. The architecture is aimed towards dividing the algorithm into

sub-tasks and implementing them in parallel in order to reduce the computation time. The proposed architecture has been designed while keeping in view the reconfigurable architectures such as FPGA and MPSoC or for multi-core processor platform. Our model equally divides the data among available processing cores which perform homogeneous tasks in parallel on respective local data points. The intermediate results of each core are shared with other cores for finalizing the ultimate clusters thus forming a collaborative working environment (CWE). Experiments were carried out on randomly generated datasets and colored images. The computational time of our proposed solution was compared against sequential implementation of the PAM algorithm. The results showed an exponential decrease in the computational complexity of the algorithm with the increase in the number of processing cores. Similarly, the speedup trends showed an almost linear increase against the sequential algorithm. It was also observed that speedup of the proposed implementation becomes more linear as the size of the data set increases and also as the clusters become more uniform for the same data set size. In the future, the proposed algorithm can be implemented using an effective communication interface such as NoC in order to reduce the communication overhead causing non-linearity of speedup which occurs at a high number of cores. Furthermore, more experimentation can be done on real-time video feeds to validate the effectiveness of the algorithm and increase its application base to include video processing as well.

**Author Contributions:** Conceptualization, H.M., M.U.A., S.G.K. and S.A.K.; Methodology, H.M.; Software, H.M.; Validation, M.U.A., S.G.K., A.-u.Y. and S.A.K.; Resources, H.M. and M.U.A.; Data Curation, H.M.; Writing–Original Draft Preparation, H.M.; Writing—Review & Editing, M.U.A., S.K., M.M., and S.G.K.; Supervision, S.A.K.; Project Administration, M.U.A.

**Funding:** This research received no external funding.

**Conflicts of Interest:** The authors declare no conflict of interest.

## References

1. Li, F.; Hong, J. Efficient certificateless access control for wireless body area networks. *IEEE Sens. J.* **2016**, *16*, 5389–5396.
2. Pirbhulal, S.; Zhang, H.; Wu, W.; Mukhopadhyay, S.C.; Zhang, Y.T. Heart-beats based biometric random binary sequences generation to secure wireless body sensor networks. *IEEE Trans. Biomed. Eng.* **2018**, doi:10.1109/TBME.2018.2815155.
3. Agrawal, D.P. Personal/body area networks and healthcare applications. In *Embedded Sensor Systems*; Springer: Singapore, 2017; pp. 353–390.
4. Pirbhulal, S.; Zhang, H.; Alahi, M.E.; Ghayvat, H.; Mukhopadhyay, S.C.; Zhang, Y.T.; Wu, W. A novel secure IoT-based smart home automation system using a wireless sensor network. *Sensors* **2016**, *17*, 69.
5. Sodhro, A.H.; Pirbhulal, S.; Sangaiah, A.K. Convergence of IoT and product lifecycle management in medical health care. *Future Gener. Comput. Syst.* **2018**, *86*, 380–391.
6. Wu, W.; Pirbhulal, S.; Sangaiah, A.K.; Mukhopadhyay, S.C.; Li, G. Optimization of signal quality over comfortability of textile electrodes for ECG monitoring in fog computing based medical applications. *Future Gener. Comput. Syst.* **2018**, *86*, 515–526.
7. Pirbhulal, S.; Zhang, H.; Mukhopadhyay, S.C.; Li, C.; Wang, Y.; Li, G.; Wu, W.; Zhang, Y.T. An efficient biometric-based algorithm using heart rate variability for securing body sensor networks. *Sensors* **2015**, *15*, 15067–15089.
8. Rechkalov, T.V.; Zymbler, M. Accelerating Medoids-based Clustering with the Intel Many Integrated Core Architecture. In Proceedings of the 2015 9th International Conference on Application of Information and Communication Technologies (AICT), Rostov on Don, Russia, 14–16 October 2015; pp. 413–417, doi:10.1109/ICAICT.2015.7338591.
9. Tehreem, A.; Khawaja, S.G.; Akram, M.U.; Khan, S.A. A Novel Mean-shift Architecture for Scalable Multiprocessor Implementation. In Proceedings of the 2016 Future Technologies Conference (FTC), San Francisco, CA, USA, 6–7 December 2016; pp. 1107–1111, doi:10.1109/FTC.2016.7821741.
10. Girolami, M.; He, C. Probability density estimation from optimally condensed data samples. *IEEE Trans. Pattern Anal. Mach. Intell.* **2003**, *25*, 1253–1264.

11. Oyelade, O.J.; Oladipupo, O.O.; Obagbuwa, I.C. Application of K-Means Clustering algorithm for prediction of Students Academic Performance. *arXiv* **2010**, arXiv:1002.2425.
12. Akkaya, K.; Senel, F.; McLaughlan, B. Clustering of wireless sensor and actor networks based on sensor distribution and connectivity. *J. Parallel Distrib. Comput.* **2009**, *69*, 573–587.
13. Schaible, T. Method and System to Derive Glycemic Patterns from Clustering of Glucose Data. U.S. Patent No. 9,504,412, 29 November 2016.
14. Khawaja, S.G.; Akram, M.U.; Khan, S.A.; Shaukat, A.; Rehman, S. Network-on-Chip based MPSoC Architecture for K-Mean Clustering Algorithm. *Microprocess. Microsyst.* **2016**, *46*, 1–10, doi:10.1016/j.micpro.2016.08.006.
15. Wu, W.; Zhang, Z.; Pirbhulal, S.; Mukhopadhyay, S.C.; Zhang Y.T. Assessment of biofeedback training for emotion management through wearable textile physiological monitoring system. *IEEE Sens. J.* **2018**, *15*, 7087–7095.
16. Pirbhulal, S.; Shang, P.; Wu, W.; Sangaiah, A.K.; Samuel, O.W.; Li, G. Fuzzy vault-based biometric security method for tele-health monitoring systems. *Comput. Electr. Eng.* **2018**, 71, 546–557.
17. Sodhro, A.H.; Pirbhulal, S.; Sangaiah, A.K.; Lohano, S.; Sodhro, G.H.; Luo, Z. 5G-Based Transmission Power Control Mechanism in Fog Computing for Internet of Things Devices. *Sustainability* **2018**, *10*, 1258.
18. Sodhro, A.H.; Sangaiah, A.K.; Pirphulal, S.; Sekhari, A.; Ouzrout, Y. Green media-aware medical IoT system. *Multimed. Tools Appl.* **2018**, 77, 1–20.
19. Kaufman, L.; Rousseeuw, P.J. Clustering by Means of Medoids. In *Statistical Data Analysis Based on the L1 Norm and Related Methods*; Dodge, Y., Ed.; Birkhäuser: Amsterdam, The Netherlands, 1987; pp. 405–416.
20. Lloyd, S.P. Least Squares Quantization in PCM. *IEEE Trans. Inf. Theory* **1982**, *28*, 129–136, doi:10.1109/TIT.1982.1056489.
21. Huang, Z. Extensions to the k-Means Algorithm for Clustering Large Data Sets with Categorical Values. *Data Min. Knowl. Discov.* **1998**, *2*, 283–304, doi:10.1023/A:1009769707641.
22. Ibrahim, A.; Gastaldo, P.; Chible, H.; Valle, M. Real-time digital signal processing based on FPGAs for electronic skin implementation. *Sensors* **2017**, *17*, 558.
23. Chen, C.A.; Chen, S.L.; Huang, H.Y.; Luo, C.H. An efficient micro control unit with a reconfigurable filter design for wireless body sensor networks (WBSNs). *Sensors* **2012**, *12*, 16211–16227.
24. Rodríguez, A.; Valverde, J.; Portilla, J.; Otero, A.; Riesgo, T.; de la Torre, E. FPGA-Based High-Performance Embedded Systems for Adaptive Edge Computing in Cyber-Physical Systems: The ARTICo3 Framework. *Sensors* **2018**, *18*, 1877.
25. Vishnoi, U.; Noll, T.G. Area-and energy-efficient CORDIC accelerators in deep sub-micron CMOS technologies. *Adv. Radio Sci.* **2012**, *10*, 207–213.
26. Gadea-Gironés, R.; Colom-Palero, R.; Herrero-Bosch, V. Optimization of Deep Neural Networks Using SoCs with OpenCL. *Sensors* **2018**, *18*, 1384.
27. Luo, J.H.; Lin, C.H. Pure FPGA implementation of an HOG based real-time pedestrian detection system. *Sensors* **2018**, *18*, 1174.
28. Mehmood, S.; Cagnoni, S.; Mordonini, M.; Farooq, M. Particle swarm optimisation as a hardware-oriented meta-heuristic for image Analysis. In Proceedings of the Workshops on Applications of Evolutionary Computation, Tübingen, Germany, 15–17 April 2009; Springer: Berlin/Heidelberg, Germany, 2009.
29. Vishnoi, U.; Noll, T.G. Cross-layer optimization of QRD accelerators. In Proceedings of the ESSCIRC (ESSCIRC), Bucharest, Romania, 16–20 September 2013.
30. Aljoby, W.; Alenezi, K. Parallelization of K-Medoid Clustering Algorithm. In Proceedings of the 5th International Conference on Information and Communication Technology for the Muslim World (ICT4M), Rabat, Morocco, 26–27 March 2013; doi:10.1109/ICT4M.2013.6518923.
31. Rechkalov, T.V. Partition Around Medoids Clustering on the Intel Xeon Phi Many-Core Coprocessor. In Proceedings of the 1st Ural Workshop on Parallel, Distributed, and Cloud Computing for Young Scientists (Ural-PDC 2015), Yekaterinburg, Russia, 17 November 2015; pp. 29–41.
32. Velmurugan, T.; Santhanam, T. A Practical Approach of K-Medoids Clustering Algorithm for Artificial data points. In Proceedings of the International Conference on Semantics, E-business and E-Commerce, Tiruchirappalli, India, 4–6 November 2009.
33. Park, H.S.; Jun, C.H. A simple and fast algorithm for K-medoids clustering. *Expert Syst. Appl.* **2009**, *36*, 3336–3341, doi:10.1016/j.eswa.2008.01.039.

34. Tehreem, A.; Khawaja, S.G.; Khan, A.M.; Akram, M.U.; Khan, S.A. Multiprocessor architecture for real-time applications using mean shift clustering. *J. Real-Time Image Process.* **2017**, 1–14, doi:10.1007/s11554-017-0733-0.
35. Saponara, S.; Fanucci, L.; Petri, E. A multi-processor NoC-based architecture for real-time image/video enhancement. *J. Real-Time Image Process.* **2013**, *8*, 111–125, doi:10.1007/s11554-011-0215-8.
36. Mehmood, S.; Cagnoni, S.; Mordonini, M.; Khan, S.A. An embedded architecture for real-time object detection in digital images based on niching particle swarm optimization. *J. Real-Time Image Process.* **2015**, *10*, 75–89, doi:10.1007/s11554-012-0256-7.
37. Li, H.-Y.; Hwang, W.-J.; Chang, C.-Y. Efficient Fuzzy C-Means Architecture for Image Segmentation. *Sensors* **2011**, *11*, 6697–6718, doi:10.3390/s110706697.
38. Monemi, A.; Tang, J.W.; Palesi, M.; Marsono, M.N. ProNoC: A low latency network-on-chip based many-core system-on-chip prototyping platform. *Microprocess. Microsyst.* **2017**, *54*, 60–74.
39. Kaufman, L.; Rousseeuw, P.J. Partitioning of Medoids (Program PAM). In *Finding Groups in Data an Introduction to Cluster Analysis*; John Wiley & Sons, Inc.: Hoboken, NJ, USA, 2005; pp. 68–125.
40. Ruaro, M.; Lazzarotto, F.B.; Marcon, C.A.; Moraes, F.G. DMNI: A specialized network interface for NoC-based MPSoCs. In Proceedings of the 2016 IEEE International Symposium on Circuits and Systems (ISCAS), Montreal, QC, Canada, 22–25 May 2016.
41. Sievers, G.; Hübener, B.; Ax, J.; Flasskamp, M.; Kelly, W.; Jungeblut, T.; Porrmann, M. The CoreVA-MPSoC: A multiprocessor platform for software-defined radio. In *Computing Platforms for Software-Defined Radio*; Springer: Cham, Switzerland, 2017; pp. 29–59.
42. Sepulveda, J.; Flórez, D.; Immler, V.; Gogniat, G.; Sigl, G. Efficient security zones implementation through hierarchical group key management at NoC-based MPSoCs. *Microprocess. Microsyst.* **2017**, *50*, 164–174.
43. Wang, Z.; Liu, W.; Xu, J.; Li, B.; Iyer, R.; Illikkal, R.; Wu, X.; Mow, W.H.; Ye, W. A case study on the communication and computation behaviors of real applications in NoC-based MPSoCs. In Proceedings of the 2014 IEEE Computer Society Annual Symposium on VLSI (ISVLSI), Tampa, FL, USA, 9–11 July 2014.
44. Kiani, V.; Reshadi, M. Mapping multiple applications onto 3D NoC-based MPSoCs supporting wireless links. *J. Supercomput.* **2017**, *73*, 2187–2213.
45. Wang, Z.; Bovik, A.C.; Sheikh, H.R.; Simoncelli, E.P. Image quality assessment: From error visibility to structural similarity. *IEEE Trans. Image Process.* **2004**, *13*, 600–612.