Fast and Area-Efficient Hardware Implementation of the K-means Clustering Algorithm

AWOS KANAN, FAYEZ GEBALI
University of Victoria
Electrical & Computer Engineering
Victoria, BC
Canada
akanan, fayez@uvic.ca

ATEF IBRAHIM
Prince Sattam Bin Abdulaziz University
Alkharj, Saudi Arabia.
Electronics Research Institute, Cairo, Egypt.
University of Victoria, Victoria, BC, Canada.
atef@ece.uvic.ca

Abstract: K-means clustering algorithm aims to partition data elements of an input dataset into K clusters in which each data element belongs to the cluster with the nearest centroid. The algorithm may take long time to process large datasets. In this paper, a fast and area-efficient hardware implementation of the K-means algorithm for clustering one-dimensional data is proposed. In the proposed design, centroids update equations are rewritten to calculate the new centroids recursively. New centroids are calculated using the current centroid value and the change in this value that results from adding or removing one data element to or from the cluster. In the new equations, the division operation appears only in the term that represents this change. The proposed design approximates only the value of this change by replacing the slow and area-consuming division operation with a shift operation. New centroids are also calculated without the need to accumulate the summation of all data elements in each cluster, as in the conventional accumulation-based design of the algorithm. Experimental results show that the approximation adopted in the proposed design results in a more area-efficient hardware implementation without affecting the quality of clustering results. Experimental results also show that the algorithm converges faster using less number of iterations as a result of continuously updating clusters centroids compared to the general update approach used in the conventional design.

Key–Words: K-means, Clustering, Data Mining, Hardware, Reconfigurable Computing, FPGA.

1 Introduction

K-means algorithm [1] is a clustering algorithm [2] that is commonly used for data analysis in many fields like machine learning, pattern recognition, bioinformatics, and image processing. The algorithm aims to partition data elements of an input dataset into K subgroups called clusters such that data elements within a cluster share some degree of similarity while being dissimilar to data in other clusters.

Because of its importance and high computational requirements, various FPGA-based hardware accelerators of the K-means algorithm have already been proposed in the literature [3–8]. In [4], the authors proposed a software/hardware co-design of the K-means algorithm. In the proposed design, distance calculation was implemented in hardware while new centroids were calculated by the host computer. The proposed hardware implementation benefited from truncating the bitwidth of the input data, and achieved a speedup of 50x over the implementation on a 500 MHz Pentium III host processor. The author of [5] proposed a systolic array architecture of the K-means clustering algorithm. The aim was to accelerate the distance calculation unit by calculating the distance between the input data and the centroids of the K clusters in parallel. The cluster index is obtained at the end of the array, and results are sent back to the host computer to calculate the new centroids. In [6], the authors proposed an efficient FPGA implementation of the K-means algorithm by utilizing a floating point divider [7] for the calculation of new centroids within the FPGA. The proposed approach required extra blocks to convert between fixed-point and floating-point data representations. The speedup of the hardware implementation using this floating point divider was compared with a hardware implementation that has the division performed in the host computer. No speedup or advantages, other than freeing the host for other tasks while the new centroids are being calculated in hardware, was gained. The authors of [8] fully implemented the K-means algorithm in hardware to cluster Microarray genomic data. The objective of their work is to have multiple cores of the K-means algorithm operating on the same chip to cluster multiple datasets in a server solution.

Given the complexity of today's data, similarity

measures and classical clustering algorithms are no longer reliable in separating clusters from each other [9]. When considering too many dimensions simultaneously, objects become distinct and dissimilarity does not provide meaningful information to determine clusters. To overcome this problem, a new approach that relies on clustering data elements in individual dimensions first, and then use the results of this clustering to perform full dimension-wide clustering has been proposed [9], [10]. In [10], the authors proposed a clustering technique for high-dimensional gene expression data. In the proposed technique, genes are initially clustered in individual dimensions using K-means algorithm, and clustering results of individual dimensions are then used for further full dimension-wide clustering.

The computation of clustering in individual dimensions, adopted as a preprocessing phase in [9], [10], consists of a set of distinct tasks that can be executed by multiple processing elements in parallel, with each processing element clustering the data in one dimension. An area-efficient hardware implementation of the K-means algorithm for clustering one-dimensional data allows for more processing elements to fit on a single chip to run in parallel.

The contributions of this paper can be summarized as:

1. Faster convergence of the K-means algorithm using less number of iterations compared to the conventional implementation.

2. Approximate centroids update equations that result in a more area-efficient hardware implementation without affecting the quality of clustering results.

This paper is organized as follows. Section 2 presents the standard K-means algorithm. In Section 3, the conventional accumulation-based hardware design of the K-means algorithm is presented. The proposed hardware design is introduced in Section 4. Implementation details of all blocks of the proposed design are presented in Section 5. Experimental results comparing the proposed and conventional designs are presented in Section 6. Finally, Section 7 concludes the paper.

2 K-means Clustering Algorithm

The K-means clustering algorithm aims to partition an input dataset of *m* data elements into *K* subgroups called clusters such that data elements within a cluster share some degree of similarity while being dissimilar to data in other clusters.

Algorithm 1 Pseudo Code of the Standard K-means Algorithm.

Input:
e=[$e_1\ e_2\ \ldots\ e_m$] %Input dataset of m data elements
K: %Number of Clusters
Output:
c=[$c_1\ c_2\ \ldots\ c_K$] % Cluster centroids
l=[$l_1\ l_2\ \ldots\ l_m$] % Cluster labels of ***e***
n=[$n_1\ n_2\ \ldots\ n_K$] % No. of data elements in each cluster
s=[$s_1\ s_2\ \ldots\ s_K$] % Sum of all data elements in each cluster

```
1: Initialize_Centroids(c);
2: done = false;
3: repeat
4:     s=[0 0 ... 0];
5:     n=[0 0 ... 0];
6:     changed = false;
7:     for (i=1 to m) do
8:         min_dist = ∞;
9:         for (j=1 to K) do
10:            if (D_ij<min_dist) then
11:                min_dist= D_ij;
12:                label = j;
13:                changed = true;
14:            end if
15:        end for
16:        l_i = label;
17:        s_label+=e_i;
18:        n_label++;
19:    end for
20:    for (k=1 to K) do
21:        c_k=s_k/n_k;
22:    end for
23: until (changed == false)
24: done = true;
```

The pseudo code of the standard K-means clustering algorithm is shown in Algorithm 1. The algorithm proceeds in iterations, each iteration begins with *K* centroids corresponding to the *K* clusters. For the first iteration, clusters centroids are initialized with random numbers in the range of values in the input dataset. Each data element is then assigned to one of the *K* clusters whose centroid is at minimal distance to the data element based on a distance metric [4] such as the Euclidean distance:

$$D_{ij} = \sqrt{\sum_{k=1}^{f} (e_{ik} - c_{jk})^2} \quad (1)$$

and Manhattan distance:

$$D_{ij} = \sum_{k=1}^{f} |e_{ik} - c_{jk}| \quad (2)$$

where D_{ij} is the distance between data element i and the centroid of cluster j, e_{ik} is the value of feature k of data element i, c_{jk} is the value of feature k of the centroid of the cluster j, and f is the number of features or dimensions. In this work, one-dimensional data is considered. For f=1 in Eqs. (1) and (2), the distance between a data element and a cluster centroid is simply the absolute value of the difference between their values.

Clusters centroids are then updated according to the new partitioning of the dataset. Centroids update may take place either continuously after the assignment of each data element, or once at the end of each iteration after the assignment of all data elements. Continuous updating of centroids is adopted in this paper since the more often the centroids are updated, the faster the algorithm will converge. The algorithm repeats for a specific number of iterations or until cluster centroids remain unchanged as a result of data elements stop moving across clusters [5].

The K-means algorithm can be formally reprepesnted as [11]:

$$X_j(t) = \{i : D_{ij} \leq D_{im} \forall\, m = 1, ..., K\} \quad (3)$$

where $X_j(t)$ is the set of data elements i assigned to cluster j at iteration t, D_{ij} denotes a suitable distance metric, such as the Euclidean or Manhattan distance, between data element i and $c_j(t-1)$, the centroid of cluster j calculated in the previous iteration. Initial centroids $c_j(0)$ for the first iteration are selected randomly. It is also required that the sets X_j are disjoint, i.e.,

$$X_j(t) \cap X_m(t) = \phi, \forall\, m \neq j. \quad (4)$$

New clusters centroids are calculated at the end of each iteration as a simple average by dividing the summation of all data elements assigned to each cluster by the number of these elements:

$$s_j(t) = \sum_{i \in X_j(t)} e_i \quad (5)$$

$$c_j(t) = \frac{s_j(t)}{n_j(t)} \quad (6)$$

where $c_j(t)$ is the updated centroid of cluster j at the end of iteration t, e_i is the value of data element i, and n_j is the number of elements in the set $X_j(t)$.

3 Conventional Implementation of the K-means Algorithm

The conventional implementation of the standard K-means clustering algorithm, described in Algorithm 1, differs from the proposed implementation, that is introduced in Section 4, in two points; the calculation of the new centroids, and the update frequency of clusters centroids. In the conventional implementation, as shown in lines 4 and 5 in Algorithm 1, the summation of all data elements assigned to each cluster along with the number of these data elements are stored in special registers. A general update of the clusters centroids are performed at the end of each iteration. The new centroid of each cluster is calculated by dividing the summation of all data elements assigned to this cluster by the number of these elements, according to line 21 in Algorithm 1 and Eq. (6).

Figure 1 shows the functional blocks of the conventional hardware design of the K-means algorithm as implemented in [6], [8], [12]. The Distance Calculation unit and the Minimum Distance unit are the same in both the conventional and proposed designs. The first unit is responsible for calculating the distances between the data element and all clusters centroids. These disances are passed to the second unit to find the minimum distance among them. The index of the cluster with nearest centroid to the data element is passed to the Accumulation unit. The value of the data element is added to the sum of all data elements assigned to the cluster with the index passed from the Minimum Distance unit. The register that holds the number of elements assigned to this cluster is also inceremented by one. This process repeats to assign all data elements to their nearest clusters. The Centroids Update unit then calculates the new centroid of each cluster by dividing the summation of all data elements assigned to this cluster by the number of these data elements.

4 The proposed Approach

In this section, we introduce our approach to enhance the conventional hardware design of the K-means algorithm in terms of speed and area efficiency. Algo-

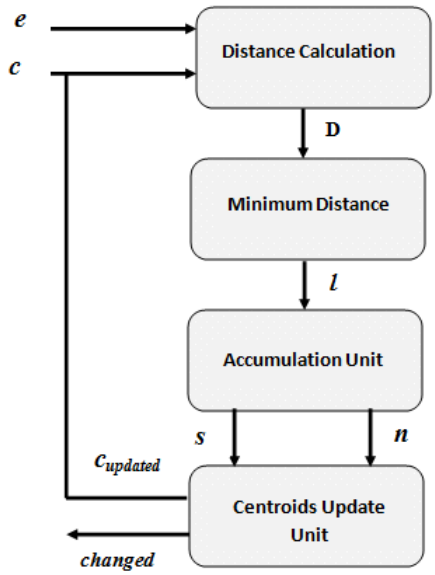


Figure 1: Functional Blocks of the Conventional K-means Algorithm Hardware Design.

rithm 2 shows the pseudo code of the proposed implementation of the K-means algorithm. Initialization step in line 1 is performed by randomly assigning all data elements to the K clusters and initialize ***c***, ***l***, and ***n*** based on this random assignment. To achieve faster convergence of the algorithm, clusters centroids are updated continuously every time a data element is assigned to a cluster, according to lines 14-23 in Algorithm 2. The centroids of two clusters need to be updated; the source cluster, from which the data element is removed, and the destination cluster, to which the data element is assigned. Implementation details of the proposed hardware design, along with the differences between the conventional and proposed implementations are described in Section 5.

New centroid update equations, that are derived in the following subsections, are written in a recursive form. New centroids are calculated as the sum of the current centroid value and the change in this value that results from adding or removing one data element to or from the cluster. In the new centroid update equations, division operation appears only in the term that represents this change. The proposed implementation approximates only the value of this change by replacing the slow and area-consuming division operation with a shift operation. The main advantage of rewriting these equations in this form, as will be shown in Section 6, is to minimize the effect of this approximation on the quality of clustering results. Another advantage of using this recursive form is that new centroids are calculated without the need to accumulate the summation of all data elements in each cluster, as in the conventional accumulation-based implementation of the algorithm. The following subsections introduce the proposed approach to update the centroids of the source and destination clusters.

Algorithm 2 Pseudo Code of the Proposed K-means Algorithm Implementation.

Input:
e=[e_1 e_2 . . . e_m] %Input dataset of m data elements
K: %Number of Clusters
Output:
c=[c_1 c_2 . . . c_K] % Cluster centroids
l=[l_1 l_2 . . . l_m] % Cluster labels of ***e***
n=[n_1 n_2 . . . n_K] % No. of data elements in each cluster

```
1: Initialize(c,l,n);
2: done = false;
3: repeat
4:     changed = false;
5:     for (i=1 to m) do
6:         src = l_i;
7:         min_dist = ∞;
8:         for (j=1 to K) do
9:             if (D_ij < min_dist) then
10:                min_dist=D_ij;
11:                dest = j;
12:            end if
13:        end for
14:        if (src ≠ dest) then
15:            l_i = dest;
16:            changed = true;
17:            n_src−−;
18:            n_dest++;
19:            X=Nearest_Power_of_2(n_src);
20:            Y=Nearest_Power_of_2(n_dest);
21:            c_src+=(c_src−e_i)>>X;
22:            c_dest+=(e_i−c_dest)>>Y;
23:        end if
24:    end for
25: until (changed == false)
26: done = true;
```

4.1 Source Cluster

As a result of removing a data element from its source cluster, the number of elements asigned to the source cluster is decremented by one, and the centroid is updated after subtracting the value of data element from the sum of data elements in the source cluster. We can

write the following iterative equations:

$$n_{src}(t) = n_{src}(t-1) - 1 \quad (7)$$

$$c_{src}(t) = \frac{[n_{src}(t)+1]c_{src}(t-1) - e_i}{n_{src}(t)} \quad (8)$$

After simple algebra, we obtain the simplified equation:

$$c_{src}(t) = c_{src}(t-1) + \frac{c_{src}(t-1) - e_i}{n_{src}(t)} \quad (9)$$

where:
e_i: Value of data element i.
$c_{src}(t-1)$: Current centroid of the source cluster.
$c_{src}(t)$: Updated centroid of the source cluster.
$n_{src}(t-1)$: Current number of elements in the source cluster.
$n_{src}(t)$: Updated number of elements in the source cluster.

4.2 Destination Cluster

After a data element is assigned to its destination cluster, the number of elements in the destination cluster is incremented by one, and the centroid is updated after adding the value of the data element to the sum of data elements in the destination cluster. We can write the following iterative equations:

$$n_{dest}(t) = n_{dest}(t-1) + 1 \quad (10)$$

$$c_{dest}(t) = \frac{[n_{dest}(t)-1]c_{dest}(t-1) + e_i}{n_{dest}(t)} \quad (11)$$

After simple algebra, we obtain the simplified equation:

$$c_{dest}(t) = c_{dest}(t-1) + \frac{e_i - c_{dest}(t-1)}{n_{dest}(t)} \quad (12)$$

where:
e_i: Value of data element i.
$c_{dest}(t-1)$: Current centroid of the destination cluster.
$c_{dest}(t)$: Updated centroid of the destination cluster.
$n_{dest}(t-1)$: Current number of elements in the destination cluster.
$n_{dest}(t)$: Updated number of elements in the destination cluster.

The values of n_{src} and n_{dest} are rounded to their nearest power of 2 integer, and the division is performed as a shift right by the rounded values according to lines 19-22 in Algorithm 2.

5 Hardware Design and Implementation

The proposed hardware design of the K-means clustering algorithm, shown in Figure 2, consists of four fully pipelined units. Both Distance Calculation and Minimum Distance units have the same hardware implementation as in the conventional design of the algorithm. The main differences between the conventional and proposed designs are in the Accumulation/Count, and Centroids Update units. Hardware design and implementation details of these units are presented in the following subsections.

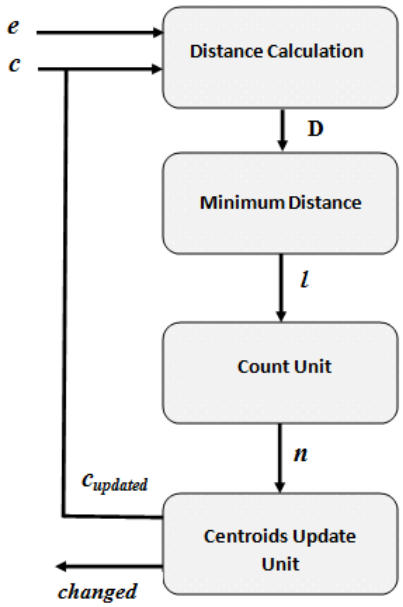


Figure 2: Functional Blocks of the proposed Hardware Design.

5.1 Distance Calculation Unit

The distance calculation unit, shown in Figure 3, reads one data element every clock cycle, and calculates the *K* distances between a data element and the centroids of the *K* clusters simultaneously.

5.2 Minimum Distance Unit

The *K* distances from the previous stage go through a compare tree to find the index of the cluster with the minimum distance to the data element as shown in Figure 4. This unit is pipelined, and the number of stages is equal to the number of levels in the compare tree, which is equal to $\lceil \log_2(K) \rceil$, where *K* is the number of clusters. The output of this unit represents the label of the nearest cluster to the current data element. As shown in Figure 4, the data element *e* is

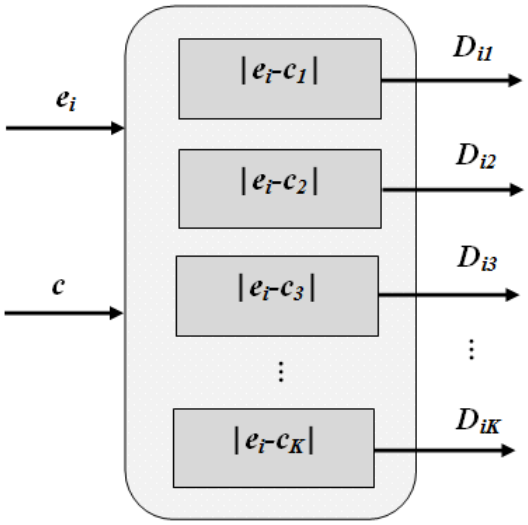


Figure 3: Distance Calculation Unit.

passed through the pipeline stages since it is needed in next stages to update clusters centroids.

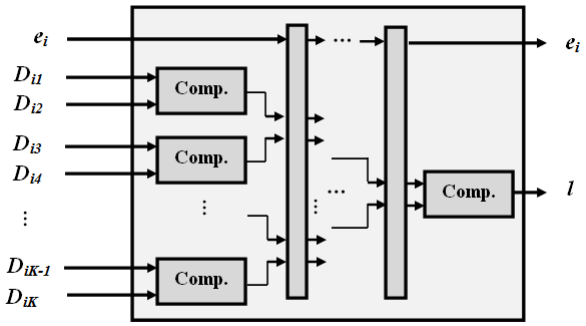


Figure 4: Minimum Distance Unit.

5.3 Count Unit

This unit keeps track of the number of data elements in each cluster. The Accumulation unit of the conventional design requires $2K$ registers. K registers to accumulate the sum of all data elements assigned to each cluster, and another K registers to store the numbers of data elements in each cluster. The proposed design, on the other hand, calculates the new centroids recursively without the need to accumulate the sums of all data elements in each cluster. Hence, only K registers are required by the Count unit of the proposed design. Figure 5 shows the hardware design of the Count unit. The inputs *src* and *dest* represents the labels of the source and destination clusters of the data element e, respectively. A comparator is used to check for the equality of *src* and *dest* to make sure that the data element is assigned to a new cluster other than its current cluster. If *src* and *dest* are not equal, the counter associated with the source cluster is decremented by one and the counter associated with the destination cluster is incremented by one, according to equations (7) and (10) and lines 17 and 18 in Algorithm 2. The values of the two registers n_{src} and n_{dest} are passed to the Centroids Update unit.

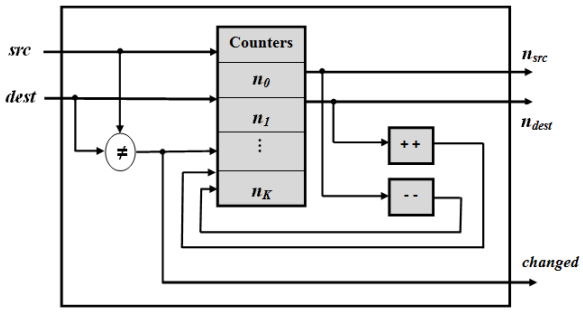


Figure 5: Count Unit.

5.4 Centroids Update Unit

As its name indicates, this unit is responsible for updating the centroids of the source and destination clusters of data elements. In the conventional design, centroids update takes place only after the assignment of all data elements in the input dataset. New centroids are calculated by dividing the sum of all data elements assigned to a cluster by the number of these data elements, which are calculated and stored in the Accumulation unit. In the proposed design, centroids of the source and destination clusters are updated continuously after the assignement of each data element according to equations (9) and (12) and lines 14-23 in Algorithm 2.

As shown in Figure 6, the Centroids Update unit is pipelined with three pipeline stages. In the first stage, the two inputs n_{src} and n_{dest} are rounded either up or down to their nearest power of 2 integer. The value of the inputs are checked against a set of intervals, and the rounded outputs are determined based on the existence of the input in one of these intervals. Shift registers in the second stage are implemented in Verilog using the shift right operator with variable shift amount equals to the rounded value of n_{src} or n_{dest}. This implementation infers a barrel shifter, which is a combinational circuit that performs the shift operation $(x >> sh\_amount)$ in one clock cycle, where sh_amount is the shift amount. Two adders are used to calculate the updated values of the source and destination clusters according to lines 21 and 22 in Algorithm 2. The algorithm proceeds in iterations, and the value of the output $changed$ is used to

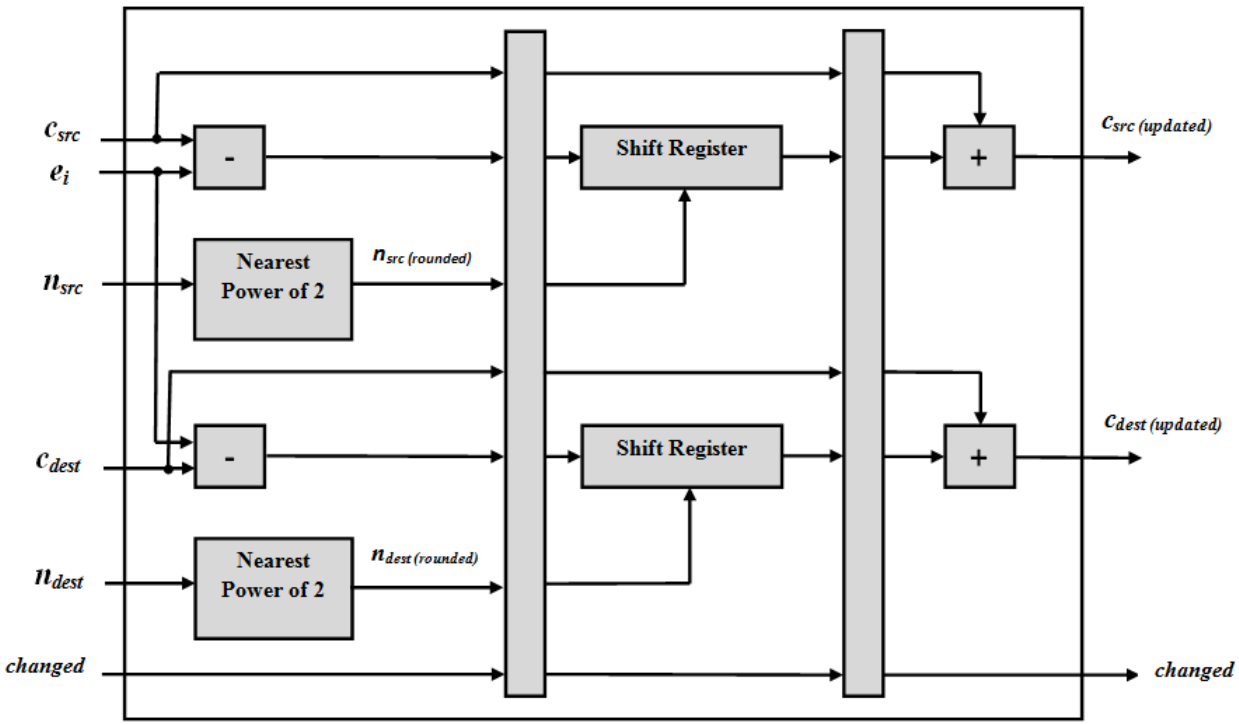


Figure 6: Centroids Update Unit.

check for algorithm convergence at the end of each iteration. A value of zero for this output means that data elements stop moving between clusters, and hence the algorithm converges.

The total number of stages in the pipelined datapath of the proposed design is:

$$N_{Stages} = 1 + \lceil \log_2(K) \rceil + 1 + 3 \quad (13)$$

$$N_{Stages} = \lceil \log_2(K) \rceil + 5 \quad (14)$$

where K is the number of clusters

6 Results and Discussion

The proposed design was implemented in Verilog hardware description language using Xilinx ISE Design Suite 9.2 to target Xilinx Virtex4 XC4VLX25. Table 1 shows implementation results for clustering a one-dimensional image dataset, that was used in previous FPGA [12] and GPU [13] implementations of the algorithm, into 8 Clusters. The implementation occupies 8% of the available slices with a maximum clock frequency of 121 MHz.

Figure 7 shows the effect of adopting the continuous update of clusters centroids on the speed of convergence of the proposed design compared to the general update approach used in the conventional design. The speedup is calculated as the ratio of the number of iterations required by the conventional design to converge, to the number of iterations required by the proposed design. Behavioral simulation results for ten runs of the two implementations, described in Algorithm 1 and Algorithm 2, are shown. Each run starts with a different set of random initial clusters centroids. Both implementations were fed with the same initial centroids in each run. The line on the box plot connects the average speedups of the 10 runs for different number of clusters. Simulation results show that the proposed design converges faster than the conventional design using less number of iterations. It is clear from the variation of speedup values that the convergence of the algorithm is highly dependent on the initial cluster centroids. To avoid biased results when comparing the two implementations, we use the same initial centroids in our expirements.

A comparison between the proposed FPGA implementation of the K-means algorithm and previous FPGA and GPU conventional implementations in terms of speed is shown in Table 2. In [12], a conventional FPGA implementation of the K-means algorithm is compared with a GPU implementation of the algorithm presented in [13] for an image processing application. The results were based on a 0.4 Mega Pixel image dataset clustered into 16, 32, and 64 clusters. GPU results in [13] were based on 2.2 GHz Intel

Table 1: Implementation Results for clustring one-dimensional dataset into 8 clusters on Xilinx XC4VLX25.

Logic Utilization	Available	Used	Utilization
No. of Slices	10752	918	8%
No. of Slice Flip Flops	21504	629	3%
No. of 4-input LUTs	21504	1598	7%
Max. Clock Frequency	121 MHz		

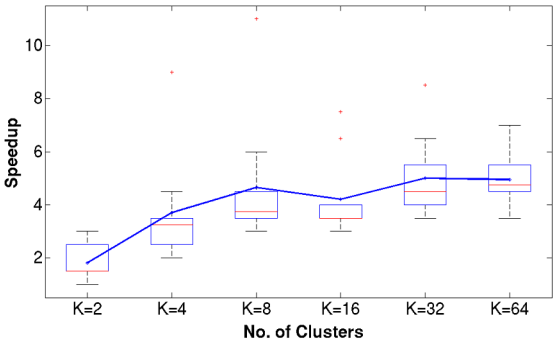


Figure 7: Speedup of the Proposed Design over the Conventional Design Calculated as the Ratio of Number of Iterations Required to Converge.

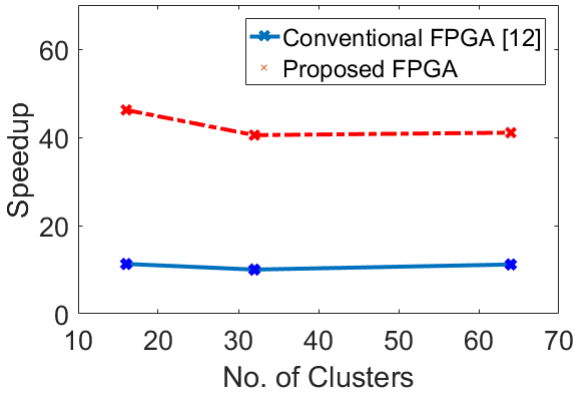


Figure 8: Speedup of Conventional [12] and Proposed FPGA Implementations of the K-means algorithm over GPU Implementation [13] for a One-Dimensional Dataset and Different Number of Clusters.

Core 2 Duo, with 4GB RAM and Nvidia GeForces 9600MGT graphics card. In [12], the targeted FPGA device was Xilinx XC4VSX35 running at a maximum clock frequency of 141 MHz.

For the implementation of the proposed design, time measurements are based on a clock frequency of 121 MHz. The execution time for a single iteration is:

$$T_{Single} = \frac{C_{Single}}{F} \tag{15}$$

where C_{Single} is the number of clock cycles required to complete one iteration of the algorithm, and F is the clock frequency.

The complete execution time of the algorithm is the time required to complete N_{iter} iterations of the algorithm before it converges:

$$T_{Complete} = T_{Single} \times N_{iter} \tag{16}$$

Table 2 shows the time per single iteration, and the complete execution time for the three implementations. In all cases, both FPGA implementations were faster than the GPU implementation. The GPU implementation is used as a reference implementation to compare the speedup of the two FPGA implementations over it. The proposed implementation took more time to complete a single iteration compared to the conventional implementation in [12]. One reason for this is because of the continuous update approach used in the proposed design, that requires more number of updates compared to the general update approach used in the conventional design. The second reason is that the conventional implementation in [12] achieved a higher maximum clock frequency compared to the maximum frequency achieved in the proposed implementation. However, and as shown if Figure 7, the continuous update adopted in the proposed design results in the algorithm to converge after less number of iterations, and hence reducing the complete execution time of the algorithm. The speedups of both FPGA implementations over the GPU implementation is shown in Figure 8.

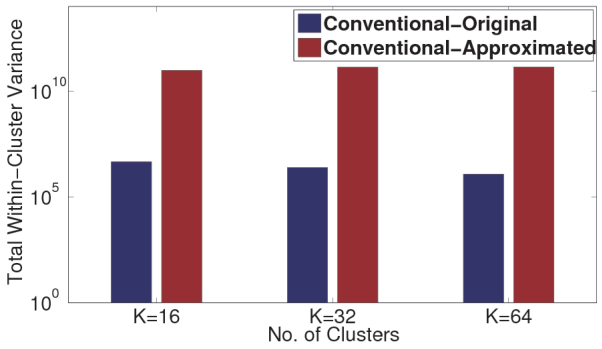
To compare our work with the conventional hardware design in terms of area efficiency, the proposed design is implemented on Xilinx XC4VLX25, the same FPGA device used in [8]. Table 3 Shows the number of slices occupied by the Centroids Update unit of the proposed design, and those occupied by the Divider used to update clusters centroids in the conventional design in [8]. To have a valid comparison, we do not compare the complete implementations since the proposed design targets one-dimensional

Table 2: Execution Times of GPU, Conventional FPGA, and Proposed FPGA Implementations of the K-means algorithm for a One-Dimensional Image Dataset and Different Number of Clusters.

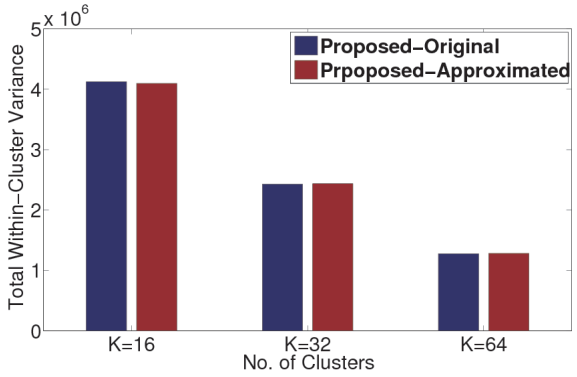
K	GPU [13]		FPGA [12]		Proposed	
	Single Iteration Time (ms)	Avg. Complete Execution Time (ms)	Single Iteration Time (ms)	Avg. Complete Execution Time (ms)	Single Iteration Time (ms)	Avg. Complete Execution Time (ms)
16	21	443	2.8	39.2	3.3	9.57
32	20	421	2.8	42	3.3	10.395
64	23	508	2.8	45.4	3.3	12.37

datasets, while the conventional design in [8] is implemented to cluster a multi-dimensional dataset. As discussed in Section 5, the two designs differ in the Accumulation/Count and Centroids Update units. The proposed design requires half the number of registers in the Count unit compared to Accumulation unit of the conventional design. As shown in Table 3, the divider used in the conventional design occupied 8% of the available slices cmpared to 3% occupied by the Centroids Update unit of the proposed design. The area occupied by divider only is equal to the total area occupied by the proposed design, as shown in Table 1.

To determine the effect of the approximation adopted in the proposed design on the quality of clustering results, we calculated the total within-cluster variance, a commonly used quality measure, for both the approximated and original implementations of the proposed and conventional designs. In the approximated implementations, the division operation in centroid update equations (6), for the conventional design, and (9),(12), for the proposed design, is implemented as a shift right by the nearest power of 2 integer to n_j. As shown in Figure 9, the proposed design is less sensitive to the error results from approximating n_j compared to the conventional design. For the proposed design, the quality of clustering results using the approximated implementation is very close to that of the original division-based implementation.



(a) Conventional Design.



(b) Proposed Design.

Figure 9: Total Within-Cluster Variance for the Approximated, Shift-based, and Original, Division-based, Centroids Update Units.

7 Conclusion and Future Work

In this paper, we have proposed a fast and area-efficient FPGA implementation of the K-means algorithm for clustering one-dimensional data. In the proposed implementation, centroids update equations are rewritten to calculate the new centroids recursively. The division operation appears in these equations is replaced with a shift operation. Experimental results show that this approximation results in a more area-efficient hardware implementation without affecting the quality of clustering results. Experimental results also show that the continuous update of clusters centroids adopted in the proposed design results in faster convergence of the algorithm using less number of iterations compared to the general update approach used in the conventional design.

The work presented in this paper shows the early results of a project that aims to accelerate clustering algorithms for high-dimensional gene expression data. As a future work, we intend to use more accurate apprximation techniques that is suitable for floating-point data. Also, the proposed implementation will be used in the clustering of high-dimensional gene ex-

Table 3: Area Occupied by the Divider and Centroids Update Units in the Conventional and Proposed Designs, Respectively.

Area	**Divider- Conventional Design [8]**	**Proposed Centroids Update Unit**
No. of Slices	967	332
Utilization	8%	3%

pression data in individual dimensions as a preprocessing to improve the quality of clustering results.

References:

[1] J. B. MacQueen. Some methods for classification and analysis of multivariate observations. In L. M. Le Cam and J. Neyman, editors, *Proc. of the fifth Berkeley Symposium on Mathematical Statistics and Probability*, volume 1, pages 281–297. University of California Press, 1967.

[2] Rui Xu, Donald Wunsch, et al. Survey of clustering algorithms. *Neural Networks, IEEE Transactions on*, 16(3):645–678, 2005.

[3] Yuk-Ming Choi and Hayden Kwok-Hay So. Map-reduce processing of k-means algorithm with FPGA-accelerated computer cluster. In *2014 IEEE 25th International Conference on Application-specific Systems, Architectures and Processors (ASAP)*, pages 9–16. IEEE, 2014.

[4] Mike Estlick, Miriam Leeser, James Theiler, and John J Szymanski. Algorithmic transformations in the implementation of k-means clustering on reconfigurable hardware. In *Proceedings of the 2001 ACM/SIGDA ninth international symposium on Field programmable gate arrays*, pages 103–110. ACM, 2001.

[5] Dominique Lavenier. FPGA implementation of the k-means clustering algorithm for hyperspectral images. In *Los Alamos National Laboratory LAUR 00-3079*, 2000.

[6] Xiaojun Wang and M. Leeser. K-means clustering for multispectral images using floating-point divide. In *15th Annual IEEE Symposium on Field-Programmable Custom Computing Machines FCCM*, pages 151–162, April 2007.

[7] Xiaojun Wang and B.E. Nelson. Tradeoffs of designing floating-point division and square root on virtex FPGAs. In *11th Annual IEEE Symposium on Field-Programmable Custom Computing Machines, 2003. FCCM 2003.*, pages 195–203, April 2003.

[8] H.M. Hussain, K. Benkrid, H. Seker, and A.T. Erdogan. FPGA implementation of k-means algorithm for bioinformatics application: An accelerated approach to clustering Microarray data. In *NASA/ESA Conference on Adaptive Hardware and Systems (AHS)*, pages 248–255, June 2011.

[9] Emin Aksehirli, Bart Goethals, and Emmanuel Müller. Efficient cluster detection by ordered neighborhoods. In *Big Data Analytics and Knowledge Discovery*, pages 15–27. Springer, 2015.

[10] Taegyun Yun, Taeho Hwang, Kihoon Cha, and Gwan-Su Yi. CLIC: clustering analysis of large microarray datasets with individual dimension-based clustering. *Nucleic Acids Research*, 38(Web-Server-Issue):246–253, 2010.

[11] Kai J Kohlhoff, Vijay S Pande, and Russ B Altman. K-means for parallel architectures using all-prefix-sum sorting and updating steps. *Parallel and Distributed Systems, IEEE Transactions on*, 24(8):1602–1612, 2013.

[12] Hanaa M Hussain, Khaled Benkrid, Ali Ebrahim, Ahmet T Erdogan, and Huseyin Seker. Novel dynamic partial reconfiguration implementation of k-means clustering on FPGAs: comparative results with GPPs and GPUs. *International Journal of Reconfigurable Computing*, 2012:1, 2012.

[13] Grzegorz Karch. GPU-based acceleration of selected clustering techniques. *Master's thesis, Silesian*, 2010.