Fast Fuzzy $c$-Means Clustering Based on Low-Cost High-Performance VLSI Architecture in Reconfigurable Hardware

Yao-Jung Yeh, Hui-Ya Li, Cheng-Yen Yang, and Wen-Jyi Hwang
Department of Computer Science and Information Engineering,
National Taiwan Normal University, Taipei, 116, Taiwan
{spmark.tw, royalfay}@gmail.com, {u93126, whwang}@csie.ntnu.edu.tw

Abstract—This paper presents a novel low-cost and high-performance VLSI architecture for fuzzy $c$-means clustering. In the architecture, the operations at both the centroid and data levels are pipelined to attain high computational speed while consuming low hardware resources. In addition, the usual iterative operations for updating the membership matrix and cluster centroid are merged into one single updating process to evade the large storage requirement. Experimental results show that the proposed solution is an effective alternative for clustering analysis with low computational cost and high performance.

Keywords—fuzzy $c$-means; fuzzy system; FPGA; reconfigurable computing; data clustering; system on programmable chip

I. INTRODUCTION

The objective of clustering methods is to provide useful information by grouping data in clusters. The $k$-means algorithm is the most well-known clustering approach which restricts each point in the data set to exactly one cluster. However, since data points can have features belonging to different clusters, the disjoint partition obtained from $k$-means technique may not be able to acquire good clustering results.

A more suitable technique for clustering applications is the fuzzy clustering. Rather than assigning each data point one and only one cluster, the fuzzy clustering assigns each data point a set of membership coefficients. Each coefficient in the set corresponds to one cluster. The cluster boundaries therefore are not crisp but fuzzy. The most popular algorithm for fuzzy clustering is the fuzzy $c$-means (FCM) algorithm, which has been applied to document clustering, pattern recognition, medical applications and data compression.

One drawback of the FCM algorithm is its high computational complexity. A number of algorithms [1], [2], [3], [9] have been proposed for accelerating the computational speed and/or reducing memory requirement of FCM. Most of these algorithms are implemented by software, and only moderate acceleration can be achieved. In [4], [10], hardware implementations of FCM are proposed. However, the design in [4] is based on analog circuits. The clustering results therefore are difficult to be directly used for digital applications. Although the architecture shown in [10] adopts digital circuits, the architecture aims for applications with only two classes. The architecture may then not be useful for applications demanding the clustering of larger number of classes.

In our earlier work [8], a digital FCM architecture capable of processing more than two classes is presented. Although the architecture is effective, the area cost is very high. The large hardware resource consumption arises from the employment of broadcasting scheme for membership coefficients and centroid computation at centroid level. As a result, the area cost grows with the number of classes. The FCM architecture may then only be used for clustering applications with small number of classes.

The goal of this paper is to present a novel high performance FCM architecture consuming low hardware resources for large number of classes. Pipelined operations are performed at both the data level and centroid level in the architecture. Instead of using the broadcasting scheme at centroid level, the proposed architecture computes membership coefficients associated with each input training vector one at a time in a pipelined fashion. The hardware resource consumption can then be significantly reduced. To accelerate the computation speed, a pipeline architecture is also employed at the data level so that FCM clustering for different training vectors can be operated in parallel.

To eliminate the large storage size for membership matrix, our implementation combine the usual iterative updating processes of membership matrix and cluster centroid into a single updating process. In our design, the updating process is divided into three steps: pre-computation, membership coefficients updating, and centroid updating. The pre-computing step is used to compute and store information common to the updating of different membership coefficients. This step is beneficial for reducing the computational complexity for the updating of membership coefficients.

The membership updating step computes new membership coefficients based on a fixed set of centroids and the results of the pre-computation step. All the membership coefficients associated with a data point will be computed one at a time in this step for reducing the hardware resource consumption. The weighted sum of data points and the
The sum of membership coefficients are also updated incrementally here for the subsequent centroid computation. This incremental updating scheme eliminates the requirement for storing the entire membership coefficients.

The centroid updating step computes the centroid of clusters using the current results obtained from the membership updating step. Because of the employment of incremental updating scheme, the centroids obtained in this step are actually the centroids of the clusters of the data points processed by the circuit. The major advantage of this scheme is that the circuit can be halted at any time. The current clustering results can then be obtained from the circuit for subsequent processing. This is particularly beneficial for applications having large data volume and requiring only single pass clustering [6].

The proposed architecture has been implemented on field programmable gate array (FPGA) devices [5] so that it can operate in conjunction with a softcore CPU [12]. Using the reconfigurable hardware, we are then able to construct a system on programmable chip (SOPC) system for the FCM clustering.

II. PRELIMINARIES

We first give a brief review of the FCM algorithm. Let $X = \{x_1, ..., x_t\}$ be a data set to be clustered by the FCM algorithm into $c$ classes, where $t$ is the number of data points in the design set. Each class $i, 1 \leq i \leq c$, is identified by its centroid $v_i$. The goal of FCM is to minimize the following cost function:

$$J = \sum_{i=1}^{c} \sum_{k=1}^{t} u_{i,k}^m \|x_k - v_i\|^2,$$

\hspace{1cm} (1)

where $u_{i,k}$ is the membership of $x_k$ in class $i$, and $m > 1$ indicates the degree of fuzziness. The cost function $J$ is minimized by a two-step iteration in the FCM. In the first step, the centroids $v_1, ..., v_c$ are fixed, and the optimal membership matrix $\{u_{i,k}, i = 1, ..., c; k = 1, ..., t\}$ is computed by

$$u_{i,k} = \left( \sum_{j=1}^{c} (\|x_k - v_j\|/\|x_k - v_j\|)^{2/(m-1)} \right)^{-1}.$$

\hspace{1cm} (2)

After the first step, the membership matrix is then fixed, and the new centroid of each class $i$ is obtained by

$$v_i = \left( \sum_{k=1}^{t} u_{i,k}^m x_k \right) / \left( \sum_{k=1}^{t} u_{i,k}^m \right).$$

\hspace{1cm} (3)

The FCM algorithm requires large number of floating point operations. Moreover, from eqs.(1) and (3), it follows that the membership matrix needs to be stored for the computation of cost function and centroids. As the size of the membership matrix grows with the product of $t$ and $c$, the storage size required for the FCM may be impractically large when the data set size and/or the number of classes becomes high.

III. THE PROPOSED ARCHITECTURE

As shown in Figure 1, the proposed FCM architecture can be decomposed into six units: the pre-computation unit, the membership coefficients updating unit, the centroid updating unit, the cost function computation unit, the on-chip centroid RAM, and the control unit.

A. Pre-computation unit

The pre-computation unit is used for reducing the computational complexity of the calculation for membership coefficients. Observe that $u_{i,k}$ in eq.(2) can be rewritten as

$$u_{i,k} = \|x_k - v_i\|^{2/(m-1)} P_k^{-1},$$

\hspace{1cm} (4)

where

$$P_k = \sum_{j=1}^{c} (1/\|x_k - v_j\|^2)^{1/(m-1)}.$$

\hspace{1cm} (5)

Given $x_k$ and centroids $v_1, ..., v_c$, membership coefficients $u_{1,k}, ..., u_{c,k}$ have the same $P_k$. Therefore, the complexity for computing membership coefficients can be reduced by calculating $P_k$ in the pre-computation unit. For the sake of simplicity, we set $m = 2$ for our design. Consequently, $P_k$ can be viewed as the sum of $1/\|x_k - v_j\|^2$.

Figure 2 shows the architecture for computing $P_k$, where the $x_k$ is obtained from the on-chip memory of the SOPC system, and $v_j$ is obtained from the on-chip centroid RAM of the FCM architecture. As depicted in the Figure 2.(a), the circuit in its simplest form can be divided into two stages. The first stage evaluates $\|x_k - v_i\|^2$. The second stage finds the inverse of $\|x_k - v_i\|^2$, and then accumulates this value with $\sum_{j=1}^{c} (1/\|x_k - v_j\|^2)^{1/(m-1)}$. This circuit can be extended for a 2-stage pipelined operation, as revealed in Figure 2.(b). It can be observed from the figure that two centroids $v_i$ and $v_{i-1}$ are operated concurrently in the pipeline, where the first stage and the second stage are used.
Figure 2. The architecture of pre-computation unit: (a) Basic circuit, (b) 2-stage pipeline architecture.

Figure 3. The architecture of membership coefficients updating unit: (a) The basic circuit, (b) 2-stage pipeline architecture.
and multiplication operations so that the inverse operation can be completed in one clock. In addition, only one small table is required for obtaining high precision results. Our design therefore does not need high area cost for division implementation.

**B. Membership coefficients updating unit**

Figure 3 depicts the architecture of the membership coefficients updating unit based on eq.(4). It can be observed from Figure 3 that, given a training data \(x_k\), the membership coefficients updating unit computes \(u_{i,k}^2\) for \(i = 1, ..., c\), one at a time. Similar to the pre-computation unit, the \(x_k\) will remain as the input until all the centroids \(v_i, i = 1, ..., c\), have been fetched from the on-chip centroid RAM for the computation of \(u_{i,k}^2\). The basic circuit for computing \(u_{i,k}^2\) is shown in Figure 3.(a). Based on eq.(4) with \(m = 2\), it follows that the circuit contains 3 multipliers and 1 divider. From the figure, we observe that \(\|x_k - v_i\|^{2}P_k\) is first computed. This requires two multipliers. Following that, a divider is used for obtaining \(u_{i,k}\). Finally, another multiplier is employed for computing \(u_{i,k}^2\). The circuit can be separated into two stages for pipeline operation as shown in Figure 3.(b). The first stage and the second stage of the pipeline are used for computing \(\|x_k - v_i\|^{2}P_k\) and \(u_{i-1,k}\), respectively.

**C. Centroid updating unit**

The centroid updating unit incrementally computes the centroid of each cluster. The major advantage for the incremental computation is that it is not necessary to store the entire membership coefficients matrix for the centroid computation. Define the incremental centroid for the \(i\)-th cluster up to data point \(x_k\) as

\[
v_i(k) = \frac{\sum_{n=1}^{k} u_{i,n}^m x_n}{\sum_{n=1}^{k} u_{i,n}^m},
\]

where \(m\) is the number of clusters.
When \( k = t \), \( v_i(k) \) is then identical to the actual centroid \( v_i \) given in eq.(3).

Figure 4 shows the architecture of the centroid updating unit, which contains a multiplier, an intermediate on-chip RAM and a divider. The unit has three inputs: centroid index \( i \), training vector \( x_k \) and membership coefficient \( \mu_{2,i,k} \). As shown in Figure 4, both \( \mu_{2,i,k}x_k \) and \( \mu_{2,i,k} \) are used as the inputs to the intermediate on-chip RAM, whose architecture is depicted in Figure 5. There are two groups of cells in the RAM. The cell \( i \) in the first group contains the accumulated sum \( \sum_{j=1}^{k-1} x_j \mu_{2,j}^i \). Moreover, the cell \( i \) in the second group contains the accumulated sum \( \sum_{j=1}^{k-1} \mu_{2,j}^i \). As shown in Figure 4, the outputs of the both groups of memory cells are used for computing \( v_i(k) \) using a divider. The result will then be delivered to the on-chip centroid RAM for centroid updating.

D. Cost function computation unit

Similar to the centroid updating unit, the cost function unit incrementally computes the cost function \( J \). Define the incremental cost function \( J(i,k) \) as

\[
J(i,k) = \sum_{z=1}^{k} \sum_{j=1}^{i} u_{i,j,z}^2 \| x_z - v_j \|^2. \tag{7}
\]

As shown in Figure 6, the cost function computation circuit receives \( u_{i,k}^2 \) and \( \| x_k - v_i \|^2 \) from the membership coefficients updating unit. The product \( u_{i,k}^2 \| x_k - v_i \|^2 \) is then accumulated for computing \( J(i,k) \) in eq.(7).

When \( i = c \) and \( k = t \), \( J(i,k) \) is then identical to the actual cost function \( J \) given in eq.(1). Therefore, the output of the circuit becomes \( J \) as the cost function computations for all training vectors are completed.

E. On-Chip centroid RAM

This unit is used for storing the centroids for FCM clustering. There are two memory banks (Memory Bank 1 and Memory Bank 2) in the on-chip centroid RAM unit. The Memory Bank 1 stores the current centroids \( v_1, ..., v_c \). The Memory Bank 2 contains the new \( v_1, ..., v_c \) obtained from the centroid updating unit. Only the centroids stored in the Memory Bank 1 are delivered to the pre-computation unit and membership updating unit for the membership coefficients computation. The updated centroids obtained from the centroid updating unit are stored in the Memory Bank 2. Note that, the centroids in the Memory Bank 2 will not replace the centroids in the Memory Bank 1 until all the input training data points \( x_k, k = 1, ..., t \), are processed.
**F. Control Unit**

The goal of the control unit is to coordinate the operations of different units in the proposed architecture. In particular, by the employment of the control unit, a novel two-level pipeline operation can be conducted in the proposed architecture for expediting the fuzzy clustering.

It can be observed from Figures 2.(b) and 3.(b) that the computation of the membership coefficients are performed in the pipelined fashion. The pipeline operations are in the centroid level because different centroids \(v_k\) and \(v_{k-1}\) operate concurrently in the pipeline.

Note that the input vectors to figures 2.(b) and 3.(b) can be different, as shown in Figure 7. That is, two input vectors \(x_k\) and \(x_{k-1}\) can be operated in parallel to pre-computation unit and membership coefficients updating unit, respectively. Therefore, a higher level pipeline operation in the input vector level is also performed in this architecture. The exploitation of parallelism at the input vector level (i.e., training data level) is able to further enhance the computational efficiency of the proposed architecture.

**G. The SOPC system based on the proposed architecture**

The proposed architecture is used as a custom user logic in a SOPC system consisting of softcore NIOS CPU, DMA controller, and SDRAM. The set of training vectors is stored in the SDRAM. The training vectors are first delivered to the on-chip memory by the DMA controller, and then are transferred to the proposed circuit one at a time by the on-chip memory. The softcore NIOS CPU running on a simple software for receiving the value of cost function \(J\) of FCM. It does not participate in the partitioning and centroid computation processes. The operation for delivering the training data from the on-chip memory to the proposed circuit will be repeated until the cost function \(J\) converges. The CPU then collects the centroid of each cluster from the proposed circuit as the clustering results.

**IV. EXPERIMENTAL RESULTS**

This section presents some experimental results of the proposed architecture. The design platform is Altera Quartus II with SOPC Builder. Training vectors are obtained from image “Lena”. The vector dimension of codewords and training vectors is \(2 \times 2\). The target FPGA for the experiment is Altera Cyclone III EP3C120 [11].

Table I shows the area complexities of the proposed architecture. There are two area complexities considered in the table: logic elements (LEs) and embedded memory bits. It can be observed from the table that only the area complexities of centroid updating unit and on-chip memory are independent of \(c\). The area costs of the other components are independent of \(c\).

Table II gives comparisons of the area cost of the proposed FCM architecture with architecture presented in [8] for various \(c\) values. Because the architecture presented in [8] uses broadcasting scheme for membership coefficients and centroid computation at centroid level, we can see from Table II that the proposed architecture consumes significantly less hardware resources. As the number of classes reaches 32, the architecture presented in [8] consumes 114117 LEs, which is 97% of the LE capacity of the target FPGA device. By contrast, when the proposed architecture contains 128 classes, the architecture presented in [8] only consumes 92295 LEs, which is 78% of the LE capacity of the target FPGA device.

Table III contrasts the average distortion of the proposed FCM architecture with that of its software counterpart for various \(c\) values. The comparison is based on the same training set with 25600 training vectors. The average distortion is computed by

<table>
<thead>
<tr>
<th>(c)</th>
<th>LEs of Proposed Hardware</th>
<th>LEs of Architecture in [8]</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>16553/119088 (14%)</td>
<td>21084/119088 (18%)</td>
</tr>
<tr>
<td>8</td>
<td>18504/119088 (16%)</td>
<td>35423/119088 (30%)</td>
</tr>
<tr>
<td>16</td>
<td>22568/119088 (19%)</td>
<td>59868/119088 (50%)</td>
</tr>
<tr>
<td>32</td>
<td>30827/119088 (26%)</td>
<td>114117/119088 (97%)</td>
</tr>
<tr>
<td>64</td>
<td>47412/119088 (40%)</td>
<td>NA</td>
</tr>
<tr>
<td>96</td>
<td>69810/119088 (59%)</td>
<td>NA</td>
</tr>
<tr>
<td>128</td>
<td>92295/119088 (78%)</td>
<td>NA</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>(c)</th>
<th>Proposed Hardware Distortion</th>
<th>Software Distortion</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>231.1218</td>
<td>290.2778</td>
</tr>
<tr>
<td>8</td>
<td>110.5756</td>
<td>109.5644</td>
</tr>
<tr>
<td>16</td>
<td>78.7715</td>
<td>78.5624</td>
</tr>
<tr>
<td>32</td>
<td>64.3645</td>
<td>63.9327</td>
</tr>
<tr>
<td>64</td>
<td>52.7055</td>
<td>52.2210</td>
</tr>
</tbody>
</table>
\[ D = \frac{1}{wL} \sum_{j=1}^{t} ||x_j - v_{\alpha(x_j)}||^2, \]  
\[ \alpha(x) = \arg \min_{1 \leq j \leq c} ||x - v_j||^2. \]

where \( w \) is the vector dimension, and \( \alpha() \) is the source encoder given by

\[
\alpha(x) = \arg \min_{1 \leq j \leq c} ||x - v_j||^2. \tag{9}
\]

It can be observed from the table that both implementations have similar average distortions, and the employment of the finite precision calculation only causes a slight degradation in performance.

Table IV reveals the execution time of the proposed architecture, the hardware architecture in [8], the basic FCM software, and the fast FCM software [9]. All implementations share the same training set with 25600 training vectors. Although the fully-pipelined architecture proposed in [8] has lower CPU time than our design, its number of classes is restricted to no more than 32. It also can be observed from Table IV that the execution time of our system is significantly lower than other software implementations. In addition, the speedup enlarges as \( c \) increases. This is because the training set is often stored in the main memory, thus requires long memory access time. When \( c = 128 \), the speedup of our system over the basic FCM software realization exceeds 2718. The speedup over the fast FCM proposed in [9] also attains 26.

V. CONCLUDING REMARKS

The proposed architecture is a high performance realization for fuzzy \( c \)-means clustering which remains offering the advantage of low area cost while the number of classes is high. By replacing the employment of broadcasting scheme for membership coefficients and centroid computation with a pipelined fashion, the hardware resource consumption is dramatically reduced. To enhance the computational efficiency, both the centroid and data levels are pipelined without consuming substantial area. The requirement of large storage size for membership matrix is also excluded by combining the usual iterative updating processes of membership matrix and cluster centroid into a single updating process. Physical performance measurements show that the proposed NIOS-based SOPC system has the ability to handle the clustering with high number of classes, and has lower execution time than other software methods.

REFERENCES


