Hindawi
Scientific Programming
Volume 2019, Article ID 3185137, 11 pages
https://doi.org/10.1155/2019/3185137

Research Article

Low-Complexity Scalable Architectures for Parallel Computation of Similarity Measures

Awos Kanan,[1] **Fayez Gebali**,[2] **Atef Ibrahim**,[3,4] **and Kin Fun Li**[2]

[1]*Department of Computer Engineering, Princess Sumaya University for Technology, Amman, Jordan*
[2]*Department of Electrical and Computer Engineering, University of Victoria, Victoria, BC, Canada*
[3]*Department of Computer Engineering, College of Computer Engineering and Sciences, Prince Sattam Bin Abdulaziz University, Al-Kharj 11942, Saudi Arabia*
[4]*Department of Microelectronics, Electronics Research Institute, Cairo 11622, Egypt*

Correspondence should be addressed to Awos Kanan; aws_kanan@yahoo.com

Received 1 March 2019; Accepted 11 April 2019; Published 26 May 2019

Guest Editor: Mohamed Zahran

Copyright © 2019 Awos Kanan et al. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

Processor array architectures have been employed, as an accelerator, to compute similarity distance found in a variety of data mining algorithms. However, most of the proposed architectures in the existing literature are designed in an ad hoc manner without taking into consideration the size and dimensionality of the datasets. Furthermore, data dependencies have not been analyzed, and often, only one design choice is considered for the scheduling and mapping of computational tasks. In this work, we present a systematic methodology to design scalable and area-efficient linear (1-D) processor arrays for the computation of similarity distance matrices. Six possible design options are obtained and analyzed in terms of area and time complexities. The obtained architectures provide us with the flexibility to choose the one that meets hardware constraints for a specific problem size. Comparisons with the previously reported architectures demonstrate that one of the proposed architectures achieves less area and area-delay product besides its scalability to high-dimensional data.

1. Introduction

The computational complexity of machine learning and data mining algorithms, that are frequently used in today's embedded applications such as handwritten analysis, fingerprint/iris/signature verification, and face recognition, makes the design of efficient hardware architectures for these algorithms a challenge. The computation of similarity distance matrices is one of the computational kernels that is required by several machine learning and data mining algorithms to measure the degree of similarity between data samples [1]. For several algorithms such as K-means [2], SVM [3], and KNN [4], distance calculation is a computationally intensive task that accounts for a significant portion of the overall processing time [5].

Given the complexity of today's applications, machine learning and data mining algorithms are expected to handle big and high-dimensional data. In [6], an optimized FPGA implementation of the K-means clustering algorithm has been presented. The authors reported that the maximum number of features that could fit on Stratix V A7 FPGA is around 160. Even partitioning the computation and caching partial results in local memory to accommodate larger sizes was not efficient due to excessive global memory transactions. Most of the existing hardware architectures for similarity distance computation have not taken into consideration the size and/or dimensionality of the datasets. Theoretical time and area complexities for some architectures, including the ones presented in [7–9], have not been validated experimentally. Implementation results reported for other architectures, including [10, 11], are for low-dimensional datasets of dimensions 4 and 9, respectively. Despite the fact that these architectures have low theoretical time complexities, their poor scalability to high-dimensional data makes them not suitable for hardware implementation or implementable with poor performance, as discussed in [6].

In our recent work [12], we have systematically explored the design space of 2-D processor array architectures for similarity distance computation. Using the employed methodology, we were able to obtain the same architectures, as proposed in [7, 8] and also to identify an additional four architectures with improved area and time complexities. Furthermore, the obtained architectures have been classified into two groups based on the size and dimensionality of input datasets. 2-D processor arrays are generally faster than 1-D (linear) processor arrays as more processing elements (PEs) are used to perform the computation in parallel. On the contrary, linear arrays are more suitable for resource-constrained applications with limited area and I/O bandwidth, typically found in embedded applications. In this work, we present a systematic technique to explore the design space of linear processor arrays for the computation of similarity distance matrices in order to obtain additional design options for area and bandwidth efficiency optimization, which is desirable in the embedded system design.

In summary, the key contributions of this paper are as follows:

(i) We present an algebraic technique to design scalable low-complexity linear processor arrays for the computation of similarity distance matrices based on an algebraic analysis of data dependencies. Compared to the classical approach of analyzing data dependencies that relies on studying how output variables depend on inputs, the employed technique relies on defining a computational domain using algorithm indices and studying how input and output variables depend on these indices.

(ii) We propose six scheduling functions using computational geometry and matrix algebra. In addition to the minimum restrictions we used in [12] to obtain valid scheduling vectors, more time restrictions are introduced in this work to meet area and bandwidth constraints. Associated projection matrices for the obtained scheduling vectors are also introduced, to map points in the 3-D computation domain to PEs in the projected 1-D processor arrays.

(iii) We perform full design space exploration using the proposed scheduling vectors and their associated projection matrices. Six design options are obtained, analyzed in terms of area, speed, and bandwidth efficiency, and compared analytically and experimentally with existing architectures in the literature.

The rest of this paper is organized as follows: related work is presented in the next section. The similarity distance computation problem is formulated in Section 3. In Section 4, the systematic technique used to parallelize distance computation is introduced. In Section 5, a systematic design space exploration is performed to obtain the proposed architectures. Design comparison and implementation results are presented in Section 6 and Section 7, respectively. Finally, Section 8 concludes the paper.

2. Related Work

Several processor array architectures have been proposed for accelerating the computation of similarity distance. In [7], a distance calculation unit for a VLSI cluster analysis architecture has been proposed as a $K \times N$ 2-D processor array to calculate similarity distances between N samples of an input dataset and K cluster centroids. For datasets with large number of samples N, the proposed architecture is not feasible for hardware implementation as it consists of a large number of processing elements (PEs) with numerous input features being fed simultaneously. The authors of [8] proposed a $K \times M$ 2-D processor array for the calculation of similarity distances between samples of an M-dimensional dataset and K cluster centroids. For high-dimensional datasets with large number of features M per sample, the proposed architecture is not feasible for hardware implementation due to chip constraints in I/O bandwidth and number of pins.

Compared to 2-D processor arrays, linear arrays are generally more area-efficient with less bandwidth and energy demands. In [9], a linear processor array for the computation of similarity distance has been proposed. The proposed architecture is used to calculate similarity distances between data samples of an input dataset and clusters centroids in a VLSI clustering analyzer. Input data samples are fed in a feature-serial format. The proposed linear array has higher time complexity than 2-D processor arrays. However, both area complexity and number of I/O pins have been reduced. Another linear array for the computation of similarity measures has been proposed in [13]. The proposed architecture is used to calculate a special case of the similarity distance matrix that is required by some machine learning algorithms, in which pairwise distances among all samples of a dataset are calculated.

In [10], a distance calculation unit has been proposed to calculate similarity distances between data samples and cluster centroids in a hardware implementation of the K-means clustering algorithm. The proposed design calculates K distances between a data sample of M features and K cluster centroids concurrently using K adder trees of $M - 1$ adders each. A similar architecture with pipelined adder trees has been presented in [11] to minimize the critical path delay and improve the throughput.

3. Similarity Distance Computation

Given dataset $\mathbf{X}$ of N samples and dataset $\mathbf{Y}$ of K samples with each sample in the two datasets having M features. A similarity measure such as Manhattan, Euclidean, or Cosine distance [1, 13] can be used to generate a distance matrix $\mathbf{D}$ of $K \times N$ elements. The distance between the n^{th} sample of dataset $\mathbf{X}$ and the k^{th} sample of dataset $\mathbf{Y}$ is represented by the value of element $D(k, n)$ of matrix $\mathbf{D}$. In this work, the calculation of the similarity distance matrix, using Manhattan distance between data samples of the two datasets $\mathbf{X}$ and $\mathbf{Y}$, is used to illustrate the introduced concepts and methodologies. Manhattan distance can be expressed as follows:

$$D(k, n) = \sum_{m=0}^{M-1} |X(m, n) - Y(k, m)|, \quad 0 \le k < K, \; 0 \le n < N, \tag{1}$$

where N and K are the number of samples of datasets $\mathbf{X}$ and $\mathbf{Y}$, respectively, and M is the dimensionality (number of features) of the two datasets. The emphasis of this paper is on the parallelization of similarity distance computation rather than the similarity measure used. Hence, the work presented in this paper can be generalized to other similarity measures.

Similarity distance computation in the K-means clustering algorithm [2], for instance, is performed in the same way as described in this section. Distances between N samples of dataset $\mathbf{X}$ and the set of centroids of K clusters $\mathbf{Y}$ are calculated in order to identify the closest cluster for each data sample.

4. Parallelizing the Computation of Similarity Distance

In our recent work [12], we have systematically explored the design space of 2-D processor array architectures for similarity distance computation using the methodology proposed by Gebali for designing digital filters systolic arrays [14]. In this work, we focus on extending the methodology to explore the design space of area-efficient linear processor arrays for the computation of similarity distance matrices. For more details on the employed methodology, refer [15].

4.1. Computation Domain. As shown in Figure 1, the computation domain of Manhattan distance (1) is defined by the algorithm indices k, m, and n. Every point in the computation domain has three coordinates, represented as follows:

$$\mathbf{p} = \begin{bmatrix} k & m & n \end{bmatrix}^t. \tag{2}$$

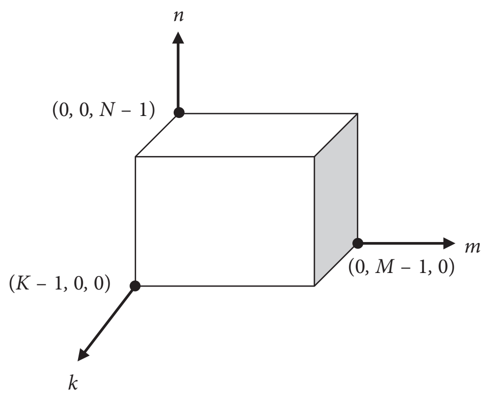


Figure 1: Computation domain.

4.2. Data Dependencies. In the traditional approach, data dependencies are analyzed in dependence graphs, by showing how output variables depend on input variables. In this work, however, data dependencies are analyzed using dependence matrices that show how input and output variables depend on indices k, m, and n, as discussed in our work on 2-D processor array architectures [12]. From (1), output variable $\mathbf{D}$ depends on indices k and n of the algorithm. Hence, its dependence matrix is given by the 2×3 integer matrix:

$$\mathbf{A_D} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}. \tag{3}$$

The three elements in each row of the dependence matrix refer to the ordered algorithm indices k, m, and n. The first row shows that variable $\mathbf{D}$ depends on index k, and the second row shows that the variable depends on index n. From (1) also, the dependence matrices of input variables $X(m,n)$ and $Y(k,m)$ are given by the following equation:

$$\mathbf{A_X} = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \quad \mathbf{A_Y} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}. \tag{4}$$

The associated null space basis vectors of these dependence matrices could be given by the following equation [12]:

$$\mathbf{e_X} = \begin{bmatrix} 1 & 0 & 0 \end{bmatrix}^t, \tag{5}$$

$$\mathbf{e_Y} = \begin{bmatrix} 0 & 0 & 1 \end{bmatrix}^t, \tag{6}$$

$$\mathbf{e_D} = \begin{bmatrix} 0 & 1 & 0 \end{bmatrix}^t. \tag{7}$$

The employed methodology, described in the following subsections, relies on these vectors to obtain valid scheduling and projection vectors to explore the design space of linear processor arrays for the computation of similarity distance matrices.

4.3. Data Scheduling. A scheduling function determines the computation load to be executed at each time step by assigning each point in the computation domain a time value. All tasks assigned the same time value will be executed in parallel. Broadcasting an algorithm variable results in assigning all points in its broadcast subdomain the same time value. Points in the subdomain of a pipelined variable, on the other hand, are assigned different time values. When an input variable is broadcast, a copy of each data element is available to all PEs through a global broadcast bus, while pipelined input variables are stored by each PE and passed to its neighbor through a local link in the next clock cycle. Broadcasting an output variable results in performing all computations on partial results from all PEs in the same clock cycle. For a pipelined output variable, the partial result that is generated by each PE is accumulated and passed to the next PE until the final result is accumulated by the last PE. One simple scheduling function that is used to schedule computation tasks is the linear scheduling function [15]:

$$t(\mathbf{p}) = \mathbf{sp}, \tag{8}$$

where $t(\mathbf{p})$ is the time value assigned to a point $\mathbf{p}$ in the computation domain and $\mathbf{s} = \begin{bmatrix} s_1 & s_2 & s_3 \end{bmatrix}$ is the scheduling vector. To broadcast an algorithm variable whose null vector is $\mathbf{e}$, we must have

$$\mathbf{se} = 0, \tag{9}$$

and to pipeline this variable, we must have

$$\mathbf{se} \neq 0. \tag{10}$$

Conditions in (9) and (10) are the minimum constraints that can be used to get a valid scheduling function.

Our strategy for arriving at suitable scheduling functions combines pipelining and broadcast restrictions in (9) and (10). We start by choosing to pipeline the evaluation of all points that lie in a plane perpendicular to one of the three k-, m-, and n-axes. Next, we pipeline the evaluation of all points that lie on lines in the chosen plane. These lines are parallel to one of the remaining two axes in that plane. Finally, we broadcast the evaluation of all points in the chosen line. In total, we have three axes to choose the planes and two directions to choose the lines in the planes. This gives rise to six possible scheduling functions. Subsection 4.3.1 illustrates how this technique is used to derive our first scheduling vector $\mathbf{s}_1$.

4.3.1. Calculation of the First Scheduling Vector $\mathbf{s}_1$. Let us choose to broadcast input variable $\mathbf{X}$. From (5) and (9), we have

$$\begin{bmatrix} s_1 & s_2 & s_3 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} = 0, \tag{11}$$

which implies $s_1 = 0$.

To avoid feeding large number of features simultaneously, we choose to supply input variable $\mathbf{X}$ in a feature-serial format (i.e., along the m-axis). This implies that, for any data sample, the time between the calculations for feature m and feature $m+1$ is a one-time step:

$$\begin{gathered} t(\mathbf{X}(k, m+1, n)) - t(\mathbf{X}(k, m, n)) = 1, \\ \begin{bmatrix} 0 & s_2 & s_3 \end{bmatrix} \begin{bmatrix} k \\ m+1 \\ n \end{bmatrix} - \begin{bmatrix} 0 & s_2 & s_3 \end{bmatrix} \begin{bmatrix} k \\ m \\ n \end{bmatrix} = 1, \end{gathered} \tag{12}$$

which implies $s_2 = 1$.

We choose to start the first calculation for sample $n+1$ after the last calculation for sample n. The time between these two calculations is also a one-time step:

$$\begin{gathered} t(\mathbf{X}(k, 0, n+1)) - t(\mathbf{X}(k, M-1, n)) = 1, \\ \begin{bmatrix} 0 & 1 & s_3 \end{bmatrix} \begin{bmatrix} k \\ 0 \\ n+1 \end{bmatrix} - \begin{bmatrix} 0 & 1 & s_3 \end{bmatrix} \begin{bmatrix} k \\ M-1 \\ n \end{bmatrix} = 1, \end{gathered} \tag{13}$$

which implies $s_3 = M$. Hence, the first valid scheduling vector is given by the following equation:

$$\mathbf{s}_1 = \begin{bmatrix} 0 & 1 & M \end{bmatrix}. \tag{14}$$

In accordance with Figure 2, the calculated scheduling vector $\mathbf{s}_1$ results in assigning all points on each of the continuous lines the same time value. These lines are called equitemporal zones since the computations for all points on each line are performed simultaneously [15]. From the geometric perspective, scheduling vector $\mathbf{s}_1$ results in executing all points in a plane with a fixed value of coordinate n before points in the plane with the next value of n. Within each plane, all points on a line with a fixed value of coordinate m are executed before points on the line with the next value of m. Points on each of these lines are executed in parallel.

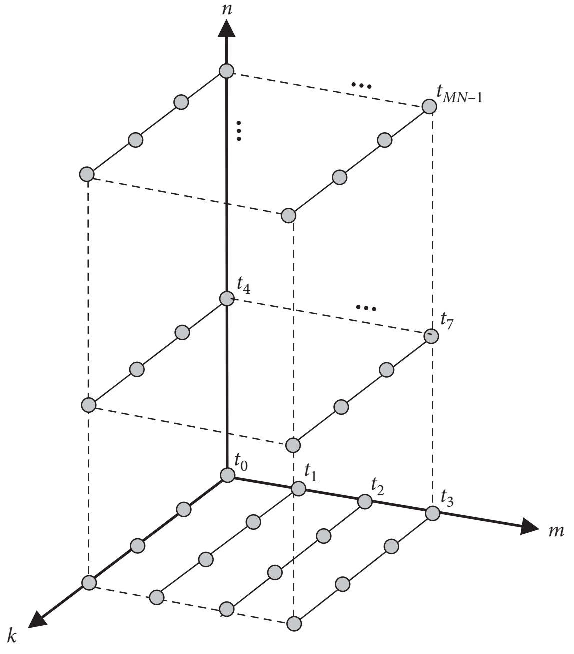


FIGURE 2: Equitemporal zones for scheduling vector $\mathbf{s}_1$.

4.3.2. Calculation of the Remaining Scheduling Vectors. The remaining five scheduling vectors can be calculated using the same procedure employed to calculate $\mathbf{s}_1$ with different orders of execution along the three axes. In Figure 3, the equitemporal zones are also lines along the k-axis with another order of execution. The associated scheduling vector is given by the following equation:

$$\mathbf{s}_2 = \begin{bmatrix} 0 & N & 1 \end{bmatrix}. \tag{15}$$

Other two timing alternatives with equitemporal zones along the m-axis are shown in Figures 4 and 5. The associated scheduling vectors for these timing alternatives are given by the following equation:

$$\begin{gathered} \mathbf{s}_3 = \begin{bmatrix} 1 & 0 & K \end{bmatrix}, \\ \mathbf{s}_4 = \begin{bmatrix} N & 0 & 1 \end{bmatrix}. \end{gathered} \tag{16}$$

Figures 6 and 7 show two timing alternatives with equitemporal zones along the n-axis. The associated scheduling vectors are given as follows:

$$\begin{gathered} \mathbf{s}_5 = \begin{bmatrix} M & 1 & 0 \end{bmatrix}, \\ \mathbf{s}_6 = \begin{bmatrix} 1 & K & 0 \end{bmatrix}. \end{gathered} \tag{17}$$

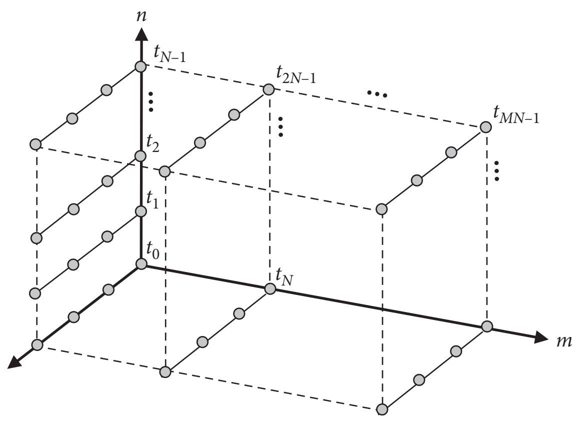


Figure 3: Equitemporal zones for scheduling vector $\mathbf{s}_2$.

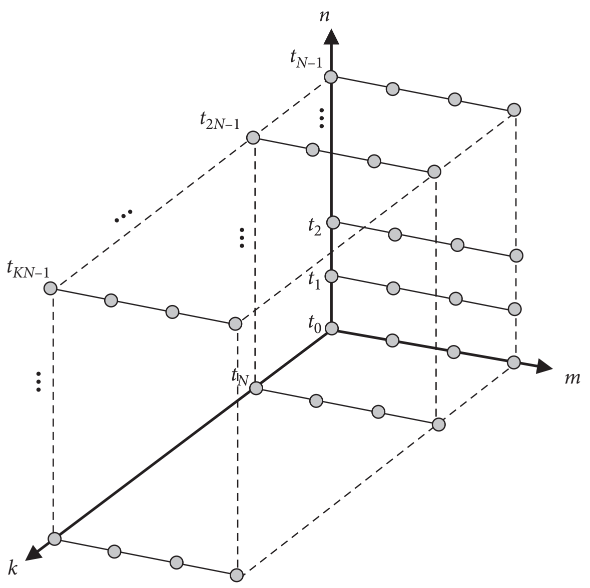


Figure 5: Equitemporal zones for scheduling vector $\mathbf{s}_4$.

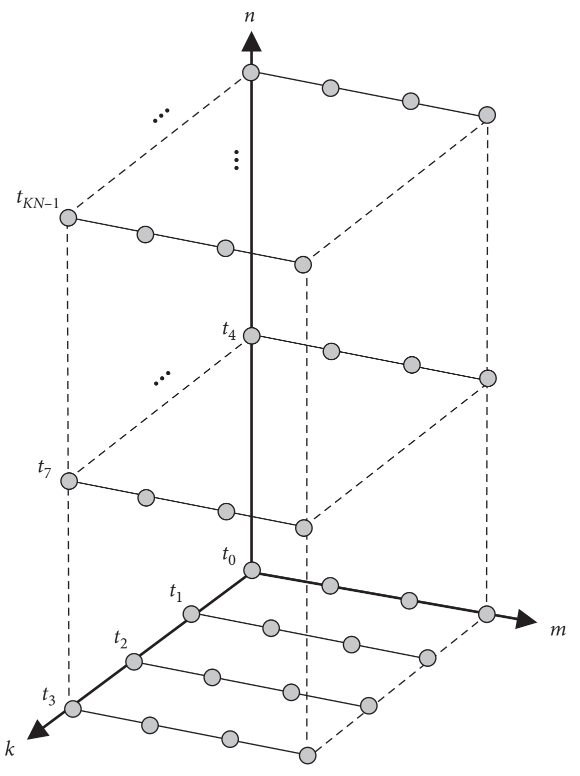


Figure 4: Equitemporal zones for scheduling vector $\mathbf{s}_3$.

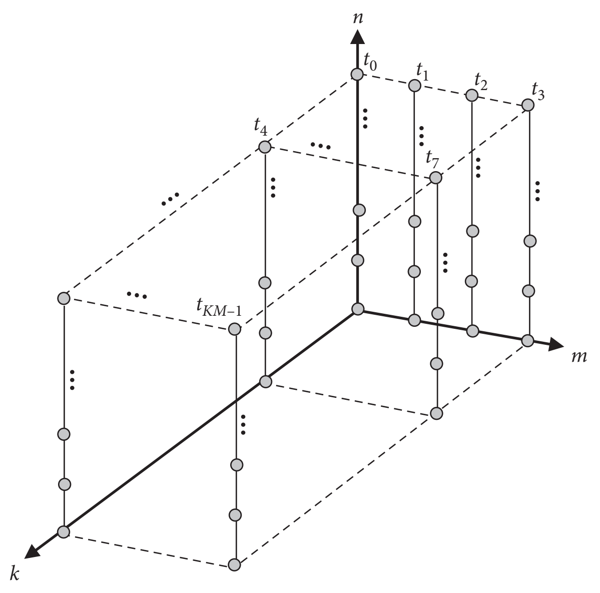


Figure 6: Equitemporal zones for scheduling vector $\mathbf{s}_5$.

4.4. Projection Operation. The computations associated with different points in the computation domain are executed in different time steps. Mapping each point in the computation domain to a single PE results in poor hardware utilization since a PE is active only for one-time instant and idle the rest of the time. Linear projection is defined as the mapping of several points in the n-dimensional computation domain $\mathscr{D}$ to a single point in a k-dimensional domain $\bar{\mathscr{D}}$, where $k \le n$. A projection matrix $\mathbf{P}$ that can be used to perform the projection operation can be obtained using a set of $l = (n-k)$ projection direction vectors $\mathbf{d_i}$ that belong to the null space of the projection matrix and satisfy the following condition as illustrated in the following equation [14]:

$$\mathbf{s}\mathbf{d_i} \neq 0, \tag{18}$$

where $\mathbf{s}$ is the chosen scheduling vector. In this work, our goal is to map the points in the 3-D computation domain shown in Figure 1 to a 1-D domain. Hence, two projection direction vectors have to be specified for each of the six scheduling vectors presented in the previous section. For the scheduling vector $\mathbf{s}_1 = [\,0 \;\; 1 \;\; M\,]$ and according to (18), two possible projection directions could be given as follows:

$$\begin{aligned}\mathbf{d}_{11} &= [\,0 \;\; 1 \;\; 0\,]^t,\\ \mathbf{d}_{12} &= [\,0 \;\; 0 \;\; 1\,]^t.\end{aligned} \tag{19}$$

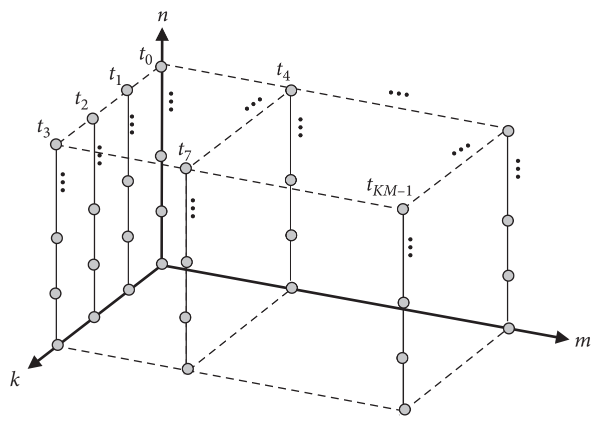


Figure 7: Equitemporal zones for scheduling vector $\mathbf{s}_6$.

Table 1: Possible projection directions and associated projection matrices.

Scheduling vector	Chosen projection directions	Associated projection matrix
$\mathbf{s}_1 = [0\ 1\ M]$	$\mathbf{d}_{11} = [0\ 1\ 0]^t$ $\mathbf{d}_{12} = [0\ 0\ 1]^t$	$\mathbf{P}_1 = [1\ 0\ 0]$
$\mathbf{s}_2 = [0\ N\ 1]$	$\mathbf{d}_{21} = [0\ 1\ 0]^t$ $\mathbf{d}_{22} = [0\ 0\ 1]^t$	$\mathbf{P}_2 = [1\ 0\ 0]$
$\mathbf{s}_3 = [1\ 0\ K]$	$\mathbf{d}_{31} = [1\ 0\ 0]^t$ $\mathbf{d}_{32} = [0\ 0\ 1]^t$	$\mathbf{P}_3 = [0\ 1\ 0]$
$\mathbf{s}_4 = [N\ 0\ 1]$	$\mathbf{d}_{41} = [1\ 0\ 0]^t$ $\mathbf{d}_{42} = [0\ 0\ 1]^t$	$\mathbf{P}_4 = [0\ 1\ 0]$
$\mathbf{s}_5 = [M\ 1\ 0]$	$\mathbf{d}_{51} = [1\ 0\ 0]^t$ $\mathbf{d}_{52} = [0\ 1\ 0]^t$	$\mathbf{P}_5 = [0\ 0\ 1]$
$\mathbf{s}_6 = [1\ K\ 0]$	$\mathbf{d}_{61} = [1\ 0\ 0]^t$ $\mathbf{d}_{62} = [0\ 1\ 0]^t$	$\mathbf{P}_6 = [0\ 0\ 1]$

These projection directions are then used to calculate the associated projection matrix according to the procedure described in [14]:

$$\mathbf{P}_1 = [1\ 0\ 0]^t. \tag{20}$$

Using projection directions $\mathbf{d}_{11}$ and $\mathbf{d}_{12}$ results in eliminating the m- and n-axes, respectively, and ensures that the projected processor array will be a linear array of K PEs along the k-axis. Table 1 shows the chosen projection directions and the associated project matrices for the six obtained scheduling vectors.

5. Design Space Exploration

In this section, we explore the design space of linear processor arrays for similarity distance computation using the derived scheduling vectors and projection matrices in Table 1.

5.1. Design #1: Using $\mathbf{s}_1 = [0\ 1\ M]$ and $\mathbf{P}_1 = [1\ 0\ 0]$.

In this design option, each point $\mathbf{p} = [k\ m\ n]^t \in \mathcal{D}$ is assigned a time value using scheduling function (8):

$$t(\mathbf{p}) = [0\ 1\ M]\begin{bmatrix} k \\ m \\ n \end{bmatrix} = m + Mn. \tag{21}$$

The projection matrix $\mathbf{P}_1$ maps any point $\mathbf{p}$ in the computation domain to the point:

$$\tilde{\mathbf{p}} = \mathbf{P}_1\mathbf{p} = [1\ 0\ 0]\begin{bmatrix} k \\ m \\ n \end{bmatrix} = k, \tag{22}$$

which implies that the resulting processor array is a linear array along the k-axis with K PEs. All points in the computation domain with the same k coordinate are mapped to the same point, or PE, in the projected computation domain. The input variable $\mathbf{X}$ is broadcast, and the broadcast direction is mapped to the vector:

$$\tilde{\mathbf{e}}_\mathbf{X} = \mathbf{P}_1\mathbf{e}_\mathbf{X} = [1\ 0\ 0]\begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} = 1, \tag{23}$$

which implies that input data are fed using broadcast lines along the k-axis in the projected architecture. Input variable $\mathbf{Y}$ is pipelined since the pipeline condition in (10) is satisfied. The pipeline direction is mapped to the vector:

$$\tilde{\mathbf{e}}_\mathbf{Y} = \mathbf{P}_1\mathbf{e}_\mathbf{Y} = [1\ 0\ 0]\begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} = 0, \tag{24}$$

which implies that input $\mathbf{Y}$ is localized in the projected architecture. Hence, the k^{th} PE uses only the M features of the k^{th} sample of input matrix $\mathbf{Y}$. Output variable $\mathbf{D}$ is also pipelined, and the pipeline direction is mapped to the vector:

$$\tilde{\mathbf{e}}_\mathbf{D} = \mathbf{P}_1\mathbf{e}_\mathbf{D} = [1\ 0\ 0]\begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} = 0, \tag{25}$$

which implies that output $\mathbf{D}$ is also localized. For every M cycles, each PE generates the distance $\mathbf{D}(k,n)$ between the n^{th} sample of dataset $\mathbf{X}$ and the k^{th} sample of dataset $\mathbf{Y}$. K distances are calculated in parallel by the K PEs. Hence, the total number of time steps is MN steps or clock cycles. The time complexity of the proposed architecture can also be determined by calculating the time value assigned by the scheduling function in (21) to the point with maximum values of coordinates k, m, and n:

$$t(\mathbf{p}_{\max}) = [0\ 1\ M]\begin{bmatrix} K-1 \\ M-1 \\ N-1 \end{bmatrix} = MN - 1. \tag{26}$$

Since the first time value is zero, the total number of time steps is $t(\mathbf{p}_{\max}) + 1 = MN$ steps. The resulting processor array and the structure of each PE are shown in Figures 8 and 9, respectively. The remaining five processor array

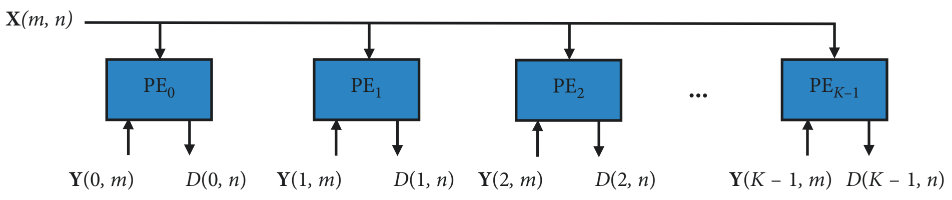


Figure 8: Processor array architecture for Design #1.

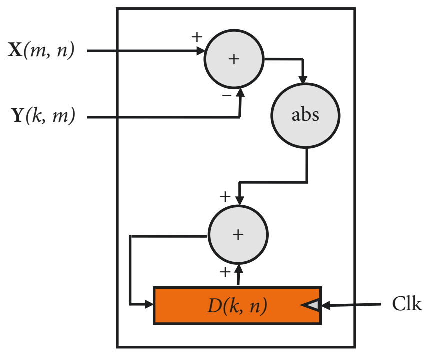


Figure 9: Processing element for Design #1 in Figure 8.

architectures shown in the following subsections are obtained using the same procedure as for Design #1.

5.2. *Design #2: Using* $\mathbf{s}_2 = [\,0 \;\; N \;\; 1\,]$ *and* $\mathbf{P}_2 = [\,1 \;\; 0 \;\; 0\,]$.

The projection matrix is the same as that of Design #1. Hence, all points in the computation domain are mapped to a linear processor array of K PEs similar to that in Figure 8 with variable $\mathbf{X}$ being broadcast while variables $\mathbf{Y}$ and $\mathbf{D}$ are localized. The chosen scheduling vector results in assigning each point in the computation domain the time value:

$$t(\mathbf{p}) = [\,0 \;\; N \;\; 1\,] \begin{bmatrix} k \\ m \\ n \end{bmatrix} = Nm + n. \tag{27}$$

The total number of time steps is also MN steps. However, the scheduling vector imposes a different order of execution. As shown in Figure 3, N computations for feature m of all data samples are performed before the N computations for feature $m+1$. Hence, N registers are required by each PE to store the intermediate results compared to only one register in Design #1.

5.3. *Design #3: Using* $\mathbf{s}_3 = [\,1 \;\; 0 \;\; K\,]$ *and* $\mathbf{P}_3 = [\,0 \;\; 1 \;\; 0\,]$.

The scheduling function for this design alternative is

$$t(\mathbf{p}) = [\,1 \;\; 0 \;\; K\,] \begin{bmatrix} k \\ m \\ n \end{bmatrix} = k + Kn. \tag{28}$$

Accordingly, the total number of time steps is KN steps. The projection matrix $\mathbf{P}_3$ maps any point $\mathbf{p}$ in the computation domain to the point:

$$\tilde{\mathbf{p}} = \mathbf{P}_3\mathbf{p} = [\,0 \;\; 1 \;\; 0\,] \begin{bmatrix} k \\ m \\ n \end{bmatrix} = m, \tag{29}$$

which implies that the resulting processor array is a linear array along the m-axis with M PEs. Both input variables $\mathbf{X}$ and $\mathbf{Y}$ are localized, and output $\mathbf{D}$ is broadcast with its broadcast direction mapped to a line along the m-axis. Broadcasting an output variable requires that partial results from all PEs are used concurrently to generate one data element of the output matrix in every clock cycle, as shown in Figure 10.

The PE structure is shown in Figure 11. Compared to the PE of Design #1 in Figure 9, no registers are required to store partial results since distance calculation is performed within a single clock cycle.

5.4. *Design #4: Using* $\mathbf{s}_4 = [\,N \;\; 0 \;\; 1\,]$ *and* $\mathbf{P}_4 = [\,0 \;\; 1 \;\; 0\,]$.

The scheduling function for this design choice is

$$t(\mathbf{p}) = [\,N \;\; 0 \;\; 1\,] \begin{bmatrix} k \\ m \\ n \end{bmatrix} = Nk + n. \tag{30}$$

The time complexity for this design is equivalent to that of Design #3 that is KN time steps. The processor array architecture and the PE structure are the same as in Figures 10 and 11, respectively. The main difference between the two designs is in the order of execution that results in generating elements of the output matrix $\mathbf{D}$ in a different order.

5.5. *Design #5: Using* $\mathbf{s}_5 = [\,M \;\; 1 \;\; 0\,]$ *and* $\mathbf{P}_5 = [\,0 \;\; 0 \;\; 1\,]$.

The scheduling function for this design option is

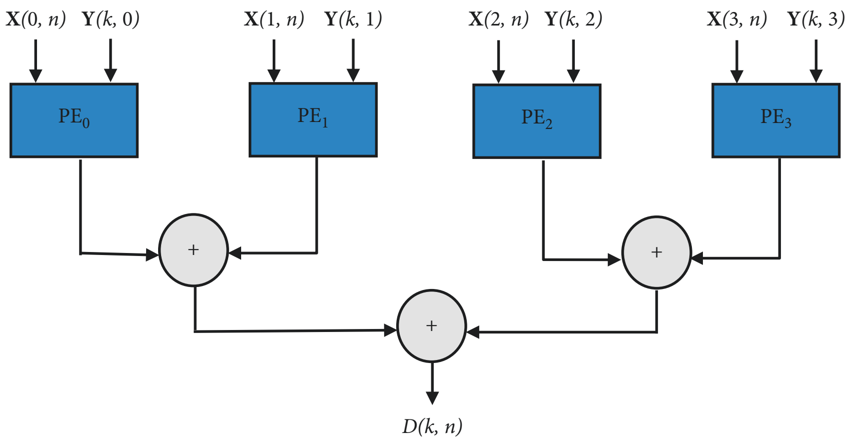


FIGURE 10: Processor array architecture for Design #3 with $M = 4$.

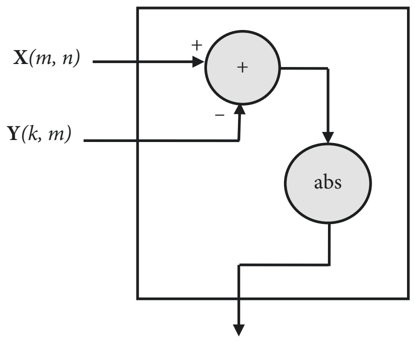


FIGURE 11: Processing element for Design #3.

$$t(\mathbf{p}) = \begin{bmatrix} M & 1 & 0 \end{bmatrix} \begin{bmatrix} k \\ m \\ n \end{bmatrix} = Mk + m. \quad (31)$$

The total number of time steps is KM steps. The projection matrix $\mathbf{P}_5$ maps any point $\mathbf{p}$ in the computation domain to the point:

$$\tilde{\mathbf{p}} = \mathbf{P}_5\mathbf{p} = \begin{bmatrix} 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} k \\ m \\ n \end{bmatrix} = n, \quad (32)$$

which implies that the resulting processor array is a linear array along the n-axis with N PEs. Both input variable $\mathbf{X}$ and output variable $\mathbf{D}$ are localized. The input variable $\mathbf{Y}$ is broadcast with its broadcast direction mapped to a line along the n-axis. For every M cycles, each PE generates the distance $\mathbf{D}(k, n)$ between the n^{th} sample of dataset $\mathbf{X}$ and the k^{th} sample of dataset $\mathbf{Y}$. N distances are calculated in parallel by the N PEs. The processor array architecture is shown in Figure 12 with the PE structure being the same as that of Design #1 in Figure 9.

5.6. Design #6: Using $\mathbf{s}_6 = \begin{bmatrix} 1 & K & 0 \end{bmatrix}$ and $\mathbf{P}_6 = \begin{bmatrix} 0 & 0 & 1 \end{bmatrix}$.

The scheduling function for this design option is

$$t(\mathbf{p}) = \begin{bmatrix} 1 & K & 0 \end{bmatrix} \begin{bmatrix} k \\ m \\ n \end{bmatrix} = k + Km. \quad (33)$$

The processor array architecture and its time complexity are the same as of Design #5. However, the order of execution that is imposed by the chosen scheduling vector, shown in Figure 7, dictates that K registers are required by each PE to store the intermediate results compared to only one register in Design #5.

6. Design Comparison

The systematic approach adopted in this paper facilitates design space exploration of linear processor arrays for the computation of similarity distance matrices. The obtained architectures provide us with the flexibility to choose the one that meets hardware constraints for specific values of system parameters K, M, and N.

Design #1 and Design #2 are suitable for parallelizing distance calculation tasks involved in processing dataset $\mathbf{X}$ of large size N compared to K that represents the size of dataset $\mathbf{Y}$. Distance calculation that is required for clustering data samples of a large-scale dataset $\mathbf{X}$ using the K-means clustering algorithm, for example, fits these design options since the size of dataset N and the number of features M are generally much larger than the number of clusters K. Compared to Design #1, Design #2 is not practical since it has the same time complexity but with a large number of additional registers to store intermediate results.

The systematic methodology adopted in this work can be used to obtain the previously devised architecture in [9]. This architecture is similar to Design #1 with input $\mathbf{X}$ being pipelined rather than broadcast. Scheduling vector $\mathbf{s}_1$ can be modified to reflect this change by applying the pipeline restriction in (10) instead of the broadcast restriction in (9). The modified scheduling vector is as follows:

$$\mathbf{s}_7 = \begin{bmatrix} 1 & 1 & M \end{bmatrix}. \quad (34)$$

The total number of time steps is $K + MN - 1$ as compared to MN steps for Design #1. The resulting

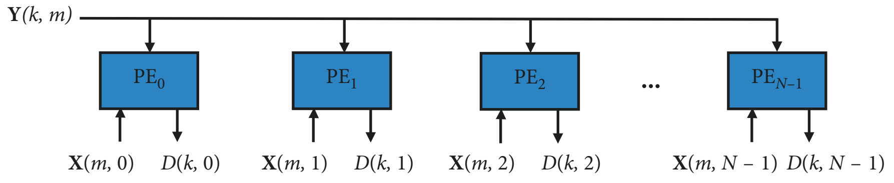

$\mathbf{Y}(k, m)$
PE_0
PE_1
PE_2
...
PE_{N-1}
$\mathbf{X}(m, 0)$ $D(k, 0)$
$\mathbf{X}(m, 1)$ $D(k, 1)$
$\mathbf{X}(m, 2)$ $D(k, 2)$
$\mathbf{X}(m, N-1)$ $D(k, N-1)$

FIGURE 12: Processor array architecture for Design #5.

processor array architecture is shown in Figure 13. A total of $K(K-1)/2$ delay registers are required to feed features of the input dataset $\mathbf{Y}$. The structure of PE is the same as of Design #1 shown in Figure 9 with one more register for the pipelined input $\mathbf{X}$.

Design #3 and Design #4 are similar to the distance calculation unit proposed in [10]. The main drawback of these three architectures is their slow clock rate when used for processing high-dimensional data since M partial results have to be added within a single clock cycle. Clock speed of these architectures can be improved by pipelining the adder trees, with each level representing a pipeline stage and a total of extra $K(M-1)$ pipeline registers, as proposed in [11]. Large-scale datasets are usually stored in off-chip or on-chip RAMs with limited bandwidth and number of data ports. Hence, architectures in [10, 11] are not practical for high-dimensional data with large number of features M as they require the feeding of a large number of features simultaneously. These two architectures are implemented as K instances of Design #3. However, the proposed architecture is more scalable to high-dimensional data, even if multiple instances of it are used, since only one feature is fed at a time. Another drawback of architectures in [10, 11] is that all features of each cluster centroid are read simultaneously. Hence, cluster centroids cannot be stored in on-chip RAMs with limited data ports and have to be stored in register banks. Our proposed architectures, on the contrary, are more flexible as the designer can choose to store cluster centroids in either on-chip RAMs or register banks based on data dimensionality and hardware constraints. This flexibility is critical for implementing hardware architectures for the K-means clustering algorithm since cluster centroids have to be updated at the end of each iteration of the algorithm.

Design #5 is another option that is similar to Design #1. The main differences between the two architectures are in the choice of broadcasting or localizing input variables $\mathbf{X}$ and $\mathbf{Y}$ and the number of PEs. Design #5 is not amenable for hardware implementation when the value of N is very large since it results in a huge number of PEs. However, this design option is suitable for processing high-dimensional, low sample size (HDLSS) datasets [16]. One example of these datasets is the gene expression microarray datasets. These datasets typically have a small number of samples N and a large number of genes that represent the features [17]. The time complexity for Design #6 is the same as that of Design #5 with extra $K \times N$ registers to store intermediate results.

Table 2 summarizes circuit and time complexities of the six proposed processor array architectures obtained in this work and the three previous architectures in [9–11]. Critical path delays are also presented, with T_a referring to the delay of a w-bit adder to model the delay of subtraction, absolute value, and addition operations, where w is the data width.

7. Implementation Results

As discussed in the previous section, Design #1 is the best design for calculating similarity distances in the K-means clustering algorithm for large-scale datasets with size N being much larger than the number of clusters K. For a fair comparison, Design #1 and previous architectures in [9–11] are implemented on the same FPGA device to accelerate distance computation involved in clustering 4,096 samples of an image dataset (Bridge) [18], with each sample consisting of 16 numerical features of the integer data type. The four architectures are implemented in Verilog hardware description language with Xilinx ISE Design Suite 13.4 targeting Xilinx Virtex7 XC7VX330T. Table 3 shows implementation results for distance calculation involved in one iteration of the K-means clustering algorithm with $N = 4{,}096$ samples, $M = 16$ features, and number of clusters $K = 64$.

Implementation results show that Design #1 outperforms Design of [9] in terms of area and speed with 25% decrease in area and 51% decrease in the area-delay product (ADP). Design of [9] occupies more slices due to the delay registers used to feed features of input dataset $\mathbf{Y}$. Execution time is determined by the number of clock cycles required to calculate all elements of distance matrix $\mathbf{D}$ and the clock rate. Design of [9] requires $K-1$ additional clock cycles and has a slower clock speed. Although the clock speed for Design #1 is affected by the delay of long broadcast bus used to feed features of dataset $\mathbf{X}$, it still attains a higher clock rate due to the higher clock skew for Design of [9], as inspected by the Xilinx tool. The effect of clock skew and long broadcast buses can be minimized by using clock distribution networks and buffer insertion for Design of [9] and Design #1, respectively, at the cost of more area and power consumption.

Based on time complexities of the four designs in Table 2, as expected, Designs of [10, 11] require less time to complete the computation of all elements of distance matrix $\mathbf{D}$. On the contrary, Design #1 achieves 58% and 14% decrease in ADP compared to Designs of [10, 11], respectively, using only 6% of their area.

8. Conclusion

The systematic technique presented in this work is used to explore the design space of scalable and area-efficient processor arrays for the computation of similarity distance matrices. Six new processor arrays, in addition to a

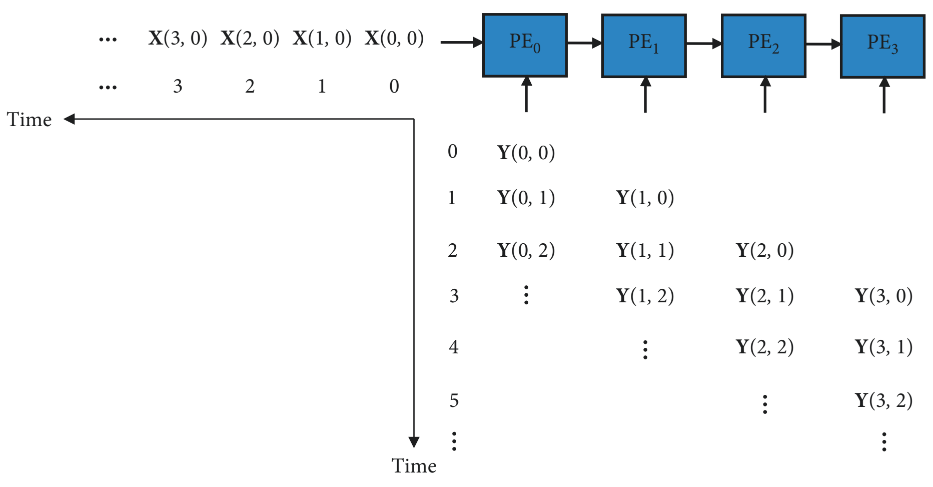


FIGURE 13: Processor array architecture for Design of [9] with $K = 4$.

TABLE 2: Design comparison.

Design	Circuit complexity			Time complexity	Critical path delay
	Absolute difference	Adder	Register		
Design #1	K	K	K	MN	$3T_a$
Design #2	K	K	KN	MN	$3T_a$
Design #3	M	$M-1$	—	KN	$(2+\log_2 M)T_a$
Design #4	M	$M-1$	—	KN	$(2+\log_2 M)T_a$
Design #5	N	N	N	KM	$3T_a$
Design #6	N	N	KN	KM	$3T_a$
[9]	K	K	$(1/2)(K^2+3K)$	$K+MN-1$	$3T_a$
[10]	KM	$K(M-1)$	KM	N	$(2+\log_2 M)T_a$
[11]	KM	$K(M-1)$	$K(2M-1)$	$\log_2 M + N$	$2T_a$

TABLE 3: Implementation results for image dataset [18] with $K = 64$, $M = 16$, and $N = 4,096$.

Design	Area (no. of slices)	Max. frequency (MHz)	Delay (μs)	Area-delay product
Design #1	768	353.1	185.6	142,540
[9]	1,020	230.5	284.5	290,190
[10]	13,132	157.9	25.9	340,118
[11]	13,087	320.7	12.7	166,204

previously devised one, are obtained systematically. Implementation results for previous architectures and one of our proposed architectures show that the proposed architecture achieves the best compromise between area and speed with an average decrease of 71% in area and 41% in ADP. The proposed architecture is more scalable to high-dimensional data as it requires the feeding of only one feature at time.

Scheduling and projection operations introduced in this work allow for control on time and area complexities of the proposed architectures. We intend to analyze the proposed architectures in terms of power efficiency and extend the proposed methodology to design power-efficient architectures that are critical for embedded applications.

Data Availability

Data sources in this work are from the public clustering datasets available at http://cs.uef.fi/sipu/datasets.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

References

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

[2] A. K. Jain, M. N. Murty, and P. J. Flynn, "Data clustering: a review," *ACM Computing Surveys (CSUR)*, vol. 31, no. 3, pp. 264–323, 1999.

[3] T.-M. Huang, V. Kecman, and I. Kopriva, *Kernel-based Algorithms for Mining Huge Datasets*, Springer, Berlin, Germany, 2006.

[4] S. Kotsiantis, "Supervised machine learning: a review of classification techniques," *Informatica*, vol. 31, no. 3, pp. 249–269, 2007.

[5] A. Choudhary, R. Narayanan, B. Ö. Ikyılmaz, G. Memik, J. Zambreno, and J. Pisharath, "Optimizing data mining

workloads using hardware accelerators," in *Proceedings of the Workshop on Computer Architecture Evaluation using Commercial Workloads (CAECW)*, Beijing China, May 2007.

[6] Q. Y. Tang and M. A. S. Khalid, "Acceleration of K-means algorithm using altera SDK for OpenCL," *ACM Transactions on Reconfigurable Technology and Systems*, vol. 10, no. 1, pp. 1–19, 2016.

[7] H. Cheng and C. Tong, "Clustering analyzer," *IEEE Transactions on Circuits and Systems*, vol. 38, no. 1, pp. 124–128, 1991.

[8] M. F. Lai, M. Nakano, Y. P. Wu, and C. H. Hsieh, "VLSI design of clustering analyser using systolic arrays," *IEE Proceedings-Computers and Digital Techniques*, vol. 142, no. 3, pp. 185–192, 1995.

[9] M. F. Hsieh and C. H. Lai, "A serial input VLSI systolic architecture for a clustering analyser," *International Journal of Electronics*, vol. 84, no. 3, pp. 269–284, 1998.

[10] R. Ratnakumar and S. J. Nanda, "A FSM based approach for efficient implementation of K-Means algorithm," in *Proceedings of the 20th International Symposium on VLSI Design and Test (VDAT)*, pp. 1–6, Guwahati, India, May 2016.

[11] H. M. Hussain, K. Benkrid, A. Ebrahim, A. T. Erdogan, and H. Seker, "Novel dynamic partial reconfiguration implementation of K-means clustering on FPGAs: comparative results with GPPs and GPUs," *International Journal of Reconfigurable Computing*, vol. 2012, Article ID 135926, 15 pages, 2012.

[12] A. Kanan, F. Gebali, and A. Ibrahim, "Design space exploration of 2-D processor array architectures for similarity distance computation," *IEEE Transactions on Parallel and Distributed Systems*, vol. 28, no. 8, pp. 2218–2228, 2017.

[13] D. G. Perera and K. F. Li, "Parallel computation of similarity measures using an FPGA-based processor array," in *Proceedings of the 22nd International Conference on Advanced Information Networking and Applications (AINA)*, pp. 955–962, Okinawa, Japan, March 2008.

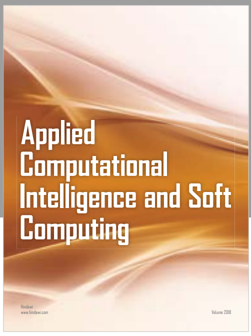
[14] F. Gebali and A. Tawfik, "Mapping 3-d IIR digital filter onto systolic arrays," *Multidimensional Systems and Signal Processing*, vol. 7, no. 1, pp. 7–26, 1996.

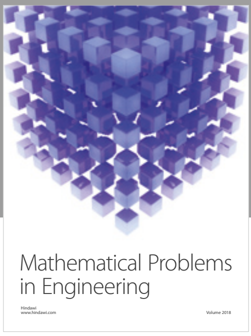
[15] F. Gebali, *Algorithms and Parallel Computing*, John Wiley & Sons, Hoboken, NJ, USA, 2011.

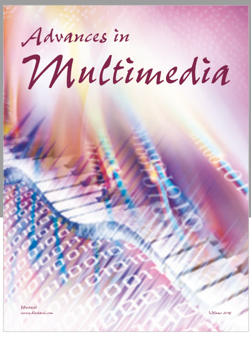
[16] J. Ahn, M. H. Lee, and Y. J. Yoon, "Clustering high dimension, low sample size data using the maximal data piling distance," *Statistica Sinica*, vol. 22, no. 2, pp. 443–464, 2012.

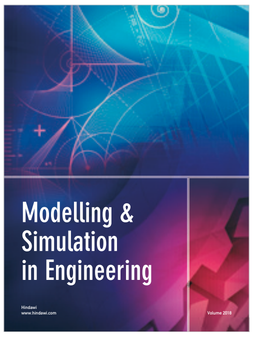
[17] A. Hochstein, H. I. Ahn, Y. T. Leung, and M. Denesuk, "Survival analysis for HDLSS data with time dependent variables: lessons from predictive maintenance at a mining service provider," in *Proceedings of the IEEE International Conference on Service Operations and Logistics, and Informatics (SOLI)*, pp. 372–381, Dongguan, China, July 2013.

[18] P. Franti: Bridge Dataset, Clustering Datasets, 2015, http://cs.uef.fi/sipu/datasets/.









Hindawi

Submit your manuscripts at
www.hindawi.com