Linear Processor Array Architectures for Similarity Distance Computation

Awos Kanan
Dept. of Computer Engineering
Princess Sumaya University for Technology
Amman, Jordan
a.kanan@psut.edu.jo

Fayez Gebali
Dept. of Electrical & Computer Engineering
University of Victoria
Victoria BC, Canada
fayez@uvic.ca

Atef Ibrahim
Dept. of Computer Engineering
Prince Sattam Bin AbdulAziz University
Al-Kharj, Saudi Arabia
aa.mohamed@psau.edu.sa

Kin Fun Li
Dept. of Electrical & Computer Engineering
University of Victoria
Victoria BC, Canada
kinli@uvic.ca

***Abstract*—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 existing literature are designed in an ad hoc manner. 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 technique to design linear processor arrays for the computation of similarity distance matrices. The technique employed is used to define the computation domain of the algorithm, with time restrictions on input and output variables. Six scheduling vectors and their associated projection matrices are generated to illustrate our systematic technique. The six possible design options obtained are analyzed in terms of area and time complexities. We are also able to derive a previously existing processor array in the literature by modifying the scheduling vector for one of the proposed architectures. Field Programmable Gate Array (FPGA) Implementations show that our proposed architecture achieves better performance in both speed and area.**

***Index Terms*—Similarity measures, data mining, parallel architectures, design methodology, processor arrays.**

I. Introduction

With the advances in computing and data storage, huge volumes of data are being collected by governments, businesses, universities, and other organizations. As the data collection rate increases, the gap between our understanding of such data and the knowledge hidden in it becomes larger and larger. To bridge this gap, novel data mining techniques and algorithms started to emerge in order to discover meaningful patterns and knowledge from such large volumes of data. Existing serial implementations of data mining algorithms are not sufficient to analyze and process this enormous amount of data in an effective manner. To satisfy performance constraints and requirements associated with data mining applications, computation must be accelerated. In [1], different acceleration platforms available for big data analysis have been surveyed. High Performance Clusters (HPC), Multicore Processors, Graphics Processing Units (GPU), and Field Programmable Gate Arrays (FPGA) platforms have been assessed and compared based on various metrics such as data size, scalability, data I/O rate, and real-time processing. According to the analysis provided in [1], FPGA-based hardware accelerators are more suitable for applications that require high I/O data rates and real-time processing.

The computation of similarity distance matrices is one of the computation kernels that is generally required by several machine learning and data mining algorithms to measure the degree of similarity between data samples [2]. For several algorithms such as K-Means [3], SVM [4], and KNN [5], distance calculation is a computationally intensive task that accounts for a significant portion of the processing time, especially when dealing with large and high-dimensional datasets [6].

Several processor array architectures have been proposed for accelerating similarity distance computation. 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. In [8], a $K \times M$ 2-D processor array has been proposed to calculate 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 constraints in I/O bandwidth and number of pins. In our recent work [9], we have systematically explored the design space of 2-D processor array architectures for similarity distance computation. The employed methodology was able to obtain the architectures proposed in [7] and [8] and also to identify an additional four architectures with improved area and time complexities. 2-D

978-1-5386-7942-5/19/$31.00 ©2019 IEEE

processor arrays are generally faster than 1-D (linear) processor arrays as more PEs are used to perform the computation in parallel. On the other hand, linear arrays are more suitable for area, power, and bandwidth-constrained applications. In [10], 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 architecture has higher time complexity than other 2-D processor arrays. However, both area complexity and number of I/O pins have been reduced. In [11], the authors proposed an FPGA-based linear processor array for similarity measures computation. The proposed architecture is used to calculate three similarity measures among all samples of an input dataset. Most of the existing processor arrays in the literature have been designed in an ad hoc manner. Data dependencies have not been analyzed, and only one design alternative is considered for tasks scheduling and mapping.

In this paper, we present a systematic technique to design linear processor arrays for the computation of similarity distance matrices. The employed technique is used to define the computation domain of the algorithm. Time restrictions on input and output variables are introduced in order to meet area and bandwidth constraints. Six scheduling vectors are calculated, and six possible design alternatives are obtained and analyzed in terms of area and time.

The rest of this paper is organized as follows: the distance computation problem is formulated in Section II. In Section III, the systematic technique used to parallelize distance computation is introduced. Scheduling and projection operations are presented in Section IV and Section V, respectively. In Section VI, the proposed architectures are obtained systematically using the calculated scheduling vectors and projection matrices. Comparison between the proposed architectures and implementation results are presented in Section VII. Finally, Section VIII concludes the paper.

II. 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 [2] [11] 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 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:

$$D(k,n) = \sum_{m=0}^{M-1} |X(m,n) - Y(k,m)| \qquad (1)$$

$$0 \leq k < K, \quad 0 \leq n < N,$$

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 [3], 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.

III. Parallelizing the Computation of Similarity Distance

In our recent work [9], we have systematically explored the design space of 2-D processor array architectures for similarity distance computation using the methodology proposed by the Gebali in [12]. In this work, the same methodology is employed, by using different scheduling and projection operations, to design linear processor arrays for the computation of similarity distance matrices.

A. Computation Domain

The computation domain $\mathcal{D}$ is defined by the three indices of the algorithm [13], as shown in Fig. 1. Every point in the computation domain has three coordinates, represented as:

$$\mathbf{p} = \begin{bmatrix} k & m & n \end{bmatrix}^t \qquad (2)$$

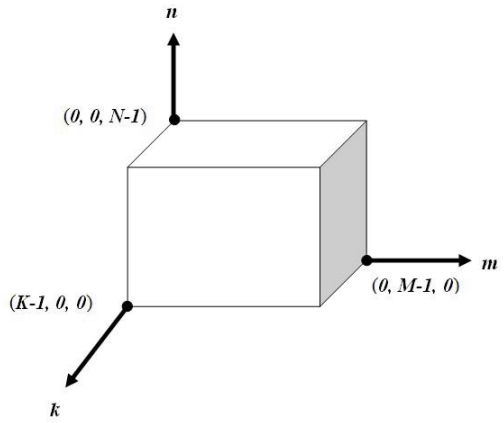

Fig. 1: Computation domain.

B. Data Dependencies

In 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 [9]. The dependence matrices of the

algorithm variables $X(m,n)$, $Y(k,m)$, and $D(k,n)$ are given by [9]:

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

$$\mathbf{A_Y} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \tag{4}$$

$$\mathbf{A_D} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{5}$$

and the associated nullvectors could be given by [9]:

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

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

$$\mathbf{e_D} = \begin{bmatrix} 0 & 1 & 0 \end{bmatrix}^t \tag{8}$$

IV. 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, 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 [13]:

$$t(\mathbf{p}) = \mathbf{sp} \tag{9}$$

where $t(\mathbf{p})$ is the time value assigned to a point $\mathbf{p}$ in the computation domain $\mathcal{D}$, and $\mathbf{s} = \begin{bmatrix} s_1 & s_2 & s_3 \end{bmatrix}$ is the scheduling vector. To broadcast an algorithm variable whose nullvector is $\mathbf{e}$, we must have [13]:

$$\mathbf{se} = 0 \tag{10}$$

and to pipeline this variable, we must have:

$$\mathbf{se} \neq 0 \tag{11}$$

Conditions in (10) and (11) 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 (10) and (11). We start by choosing to pipeline the evaluation of all points that lie in a plane perpendicular to one of the three k-, m-, or 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 IV-A illustrates how this technique is used to derive our first scheduling vector $\mathbf{s_1}$.

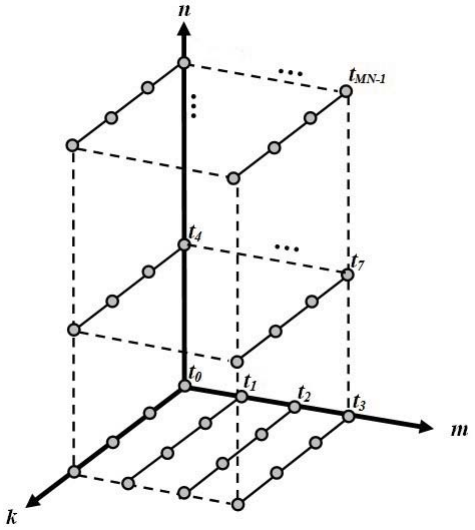

Fig. 2: Equitemporal zones for scheduling vector $\mathbf{s_1}$

A. Calculation of the first scheduling vector $\mathbf{s_1}$

Let us choose to broadcast input variable $\mathbf{X}$. From (6) and (10), we have:

$$\begin{bmatrix} s_1 & s_2 & s_3 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} = 0 \tag{12}$$

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 n, the time between the calculations for feature m and feature $m+1$ is one time step:

$$\begin{bmatrix} 0 & s_2 & s_3 \end{bmatrix} \begin{bmatrix} k \\ m+1 \\ n \end{bmatrix} - \begin{bmatrix} 0 & s_2 & s_3 \end{bmatrix} \begin{bmatrix} k \\ m \\ n \end{bmatrix} = 1 \tag{13}$$

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 one time step:

$$\begin{bmatrix}0 & 1 & s_3\end{bmatrix}\begin{bmatrix}k\\0\\n+1\end{bmatrix}-\begin{bmatrix}0 & 1 & s_3\end{bmatrix}\begin{bmatrix}k\\M-1\\n\end{bmatrix}=1 \quad (14)$$

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

$$\mathbf{s_1}=\begin{bmatrix}0 & 1 & M\end{bmatrix} \quad (15)$$

Referring to Fig.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 equitemoporal zones since the computations for all points on each line are performed simultaneously [13]. 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 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 next value of m. Points on each of these lines are executed in parallel.

B. Calculation of the remaining scheduling vectors

V. Projection Operation

Linear projection is defined as the mapping of several points in the n-dimensional computation domain $\mathcal{D}$ to a single point in a k-dimensional domain $\tilde{\mathcal{D}}$, where $k \leq n$. 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 condition [12]:

$$\mathbf{sd_i} \neq 0 \quad (16)$$

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 Fig. 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 $\mathbf{s_1}=\begin{bmatrix}0 & 1 & M\end{bmatrix}$ and according to (16), two possible projection directions could be given by:

$$\mathbf{d_{11}}=\begin{bmatrix}0 & 1 & 0\end{bmatrix}^t \quad (17)$$

and

$$\mathbf{d_{12}}=\begin{bmatrix}0 & 0 & 1\end{bmatrix}^t \quad (18)$$

These projection directions are then used to calculate the associated projection matrix according to the procedure described in [12]:

$$\mathbf{P_1}=\begin{bmatrix}1 & 0 & 0\end{bmatrix}^t \quad (19)$$

Table I shows the chosen projection directions and the associated project matrices for the six obtained scheduling vectors.

TABLE I: 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]

VI. Design Space Exploration

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

A. Design #1: using $\mathbf{s_1}$ = [0 1 M] and $\mathbf{P_1}$ = [1 0 0]

In this design option, each point $\mathbf{p}=\begin{bmatrix}k & m & n\end{bmatrix}^t \in \mathcal{D}$ is assigned a time value using the scheduling function:

$$t(\mathbf{p})=\begin{bmatrix}0 & 1 & M\end{bmatrix}\begin{bmatrix}k\\m\\n\end{bmatrix}=m+Mn \quad (20)$$

The projection matrix $\mathbf{P_1}=\begin{bmatrix}1 & 0 & 0\end{bmatrix}$ maps any point $\mathbf{p}$ in the computation domain to the point:

$$\tilde{\mathbf{p}}=\mathbf{P_1 p}=\begin{bmatrix}1 & 0 & 0\end{bmatrix}\begin{bmatrix}k\\m\\n\end{bmatrix}=k \quad (21)$$

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 will map 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:

$$\tilde{\mathbf{e}}_{\mathbf{X}}=\mathbf{P_1 e_X}=\begin{bmatrix}1 & 0 & 0\end{bmatrix}\begin{bmatrix}1\\0\\0\end{bmatrix}=1 \quad (22)$$

which implies that input data is fed using broadcast lines along the k-axis in the projected architecture. Input variable $\mathbf{Y}$ is pipelined since the pipeline condition in (11) is satisfied. The pipeline direction is mapped to the vector:

$$\tilde{\mathbf{e}}_{\mathbf{Y}} = \mathbf{P_1 e_Y} = \begin{bmatrix}1 & 0 & 0\end{bmatrix}\begin{bmatrix}0\\0\\1\end{bmatrix} = 0 \tag{23}$$

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

$$\tilde{\mathbf{e}}_{\mathbf{D}} = \mathbf{P_1 e_D} = \begin{bmatrix}1 & 0 & 0\end{bmatrix}\begin{bmatrix}0\\1\\0\end{bmatrix} = 0 \tag{24}$$

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 (20) to the point with upper limits coordinates:

$$t(\mathbf{p}_{max}) = \begin{bmatrix}0 & 1 & M\end{bmatrix}\begin{bmatrix}K-1\\M-1\\N-1\end{bmatrix} = MN - 1 \tag{25}$$

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 Fig. 3 and Fig. 4, respectively. The remaining processor array architectures are obtained in the following subsections using the same procedure utilized in this subsection to obtain Design #1.

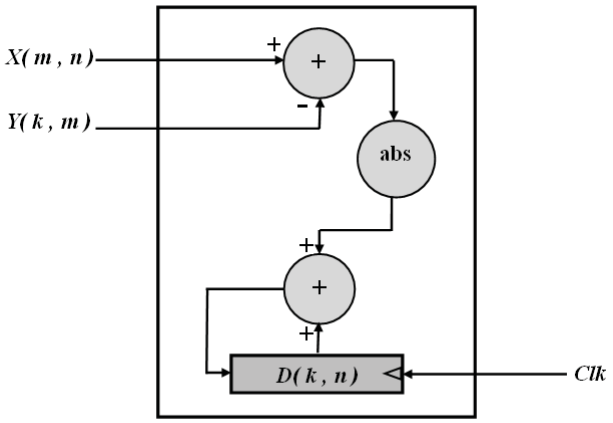

Fig. 4: Processing element for Design #1 in Fig. 3

B. 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 will be mapped to a linear processor array of K PEs similar to that in Fig. 3 with variable $\mathbf{X}$ being broadcast and variables $\mathbf{Y}$ and $\mathbf{D}$ are localized. The chosen scheduling vector results in assigning each point in the computation domain the time value:

$$t(\mathbf{p}) = \begin{bmatrix}0 & N & 1\end{bmatrix}\begin{bmatrix}k\\m\\n\end{bmatrix} = Nm + n \tag{26}$$

The total number of time steps is also MN steps. However, the scheduling vector imposes a different order of execution. 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.

C. Design #3: using $\mathbf{s_3}$ *=[1 0 K] and* $\mathbf{P_3}$ *=[0 1 0]*

The scheduling function for this design alternative is:

$$t(\mathbf{p}) = \begin{bmatrix}1 & 0 & K\end{bmatrix}\begin{bmatrix}k\\m\\n\end{bmatrix} = k + Kn \tag{27}$$

Accordingly, the total number of time steps is KN steps. 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. The PE structure is shown in Fig. 5.

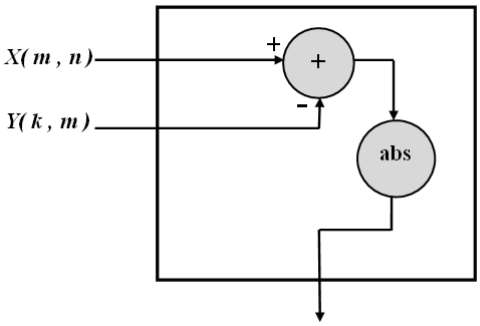

Fig. 5: Processing element for Design #3

Broadcasting an output variable requires that partial results from all PEs are used concurrently to generate one data element of the output matrix every clock cycle as shown in Fig. 6.

D. Design #4: using $\mathbf{s_4}$ *=[N 0 1] and* $\mathbf{P_4}$ *=[0 1 0]*

The scheduling function for this design choice is:

$$t(\mathbf{p}) = \begin{bmatrix}N & 0 & 1\end{bmatrix}\begin{bmatrix}k\\m\\n\end{bmatrix} = Nk + n \tag{28}$$

The time complexity for this design is equivalent to that of Design #3 which is KN time steps. The PE structure and the processor array architecture are the same in Fig. 5 and Fig. 6, 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.

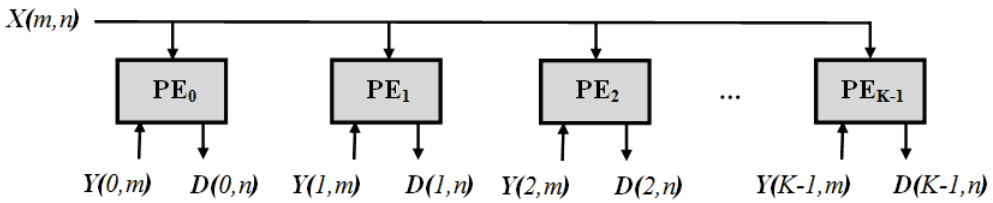

Fig. 3: Processor array architecture for Design #1

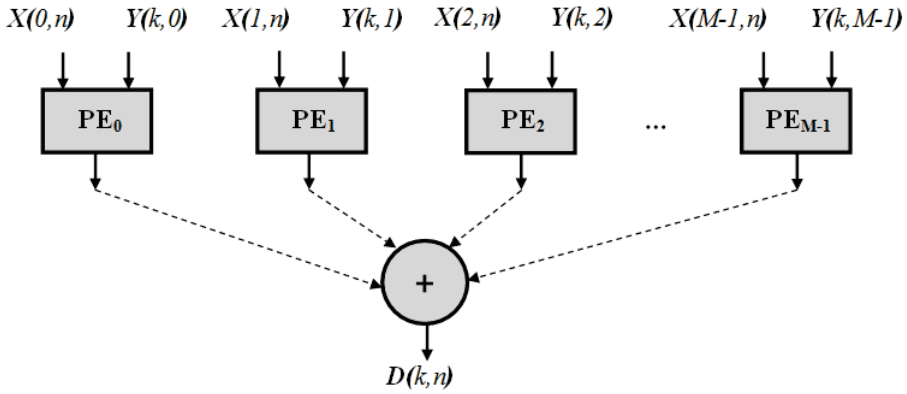

Fig. 6: Processor array architecture for Design #3

E. Design #5: using $\mathbf{s_5}$ *= [M 1 0] and* $\mathbf{P_5}$ *=[0 0 1]*

The scheduling function for this design option is:

$$t(\mathbf{p}) = \begin{bmatrix} M & 1 & 0 \end{bmatrix} \begin{bmatrix} k \\ m \\ n \end{bmatrix} = Mk + m \tag{29}$$

The total number of time steps is KM steps. Both input variable $\mathbf{X}$ and output variable $\mathbf{D}$ are localized. 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 Fig. 7 with the PE structure being the same as that of Design #1 in Fig. 4.

F. Design #6: using $\mathbf{s_6}$ *= [1 K 0] and* $\mathbf{P_6}$ *= [0 0 1]*

The scheduling function for this design alternative is:

$$t(\mathbf{p}) = \begin{bmatrix} 1 & K & 0 \end{bmatrix} \begin{bmatrix} k \\ m \\ n \end{bmatrix} = k + Km \tag{30}$$

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 dictates that K registers are required by each PE to store the intermediate results compared to only one register in Design #5.

VII. Design Comparison and Results

The systematic approach adopted in this work facilitates design space exploration of linear processor arrays for the similarity distance computation problem. 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.

A. Design Comparison

Design #1 and Design #2 are suitable for parallelizing distance calculation tasks where $N \gg K, M$. Distance calculation that is required for clustering data samples of a large dataset $\mathbf{X}$ using K-Means algorithm, for example, fits these design options since the size of dataset $\mathbf{X}$ is much larger than the number of cluster centroids K and the number of features M. Compared to Design #1, Design #2 is not practical since it has the same time complexity as a design with a large number of additional registers to store intermediate results.

In Design #1, elements of input matrix $\mathbf{X}$ are globally used by all PEs. On the other hand, each PE in Design #3 and Design #4 only consumes a single feature or dimension of data samples. In an FPGA implementation of these design alternatives, input dataset $\mathbf{X}$ can be distributed among fast and small on-chip RAMs instead of storing it in a large single, but slower off-chip RAM. The main drawback of these two architectures is their slow clock rate compared to other

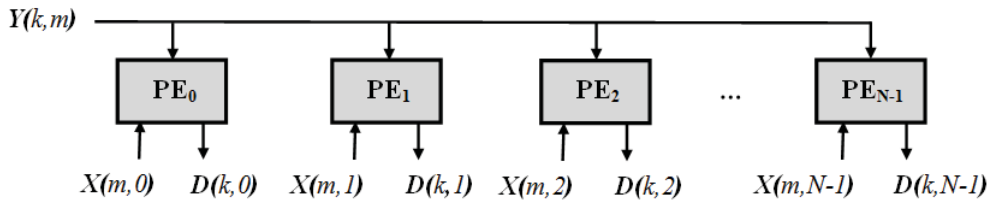

Fig. 7: Processor array architecture for Design #5

architectures since M partial results have to be added within a single clock cycle.

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 [14]. 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 [15]. The time complexity for Design #6 is the same as of Design #5 with an extra $K \times N$ registers to store intermediate results.

In addition to the six obtained architectures, the systematic methodology adopted in this work can be used to obtain a previously devised architecture in [10]. 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 pipeline restriction in (11) instead of broadcast restriction in (10). The modified scheduling vector is:

$$\mathbf{s_7} = \begin{bmatrix} 1 & 1 & M \end{bmatrix} \tag{31}$$

The total number of time steps is $(K+MN-1)$ as compared to (MN) steps for Design #1. The resulting processor array architecture is shown in Fig. 8. A total of $\frac{1}{2}K(K-1)$ delay registers are used to feed features of input dataset $\mathbf{Y}$. As shown in Fig. 9, no delay registers are used to feed any of the inputs as in Design #1. The structure of PE is the same as of Design #1 shown in Fig. 4 with one more register for the pipelined input $\mathbf{X}$.

B. Implementation Results

Both Design #1 and Design of [10] are implemented on FPGA to accelerate distance computation involved in clustering pixels of 512×512 RGB color images. Each pixel is represented using three 8-bit color components (red, green, and blue). Both architectures are implemented in Verilog hardware description language with Xilinx ISE Design Suite 13.4 targeting Xilinx Virtex7 XC7VX330T. Table II and Fig. 10 show implementation results for distance calculation involved in one iteration of the K-Means clustering algorithm with N=262,144 pixels, M=3 features, and different number of clusters K.

Implementation results show that Design #1 outperforms Design of [10] in terms of area and speed for all values of K. Design of [10] 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. For all values of K, Design of [10] has a slower clock speed and requires $(K-1)$ more clock cycles than Design #1. As shown in Table II, as the number of PEs increases, the maximum clock rate for Design #1 decreases due to the higher delay of longer broadcast buses. However, Design #1 still attains higher clock rate than Design of [10] due to higher clock skew 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 [10] and Design #1, respectively at the cost of more area and power consumption.

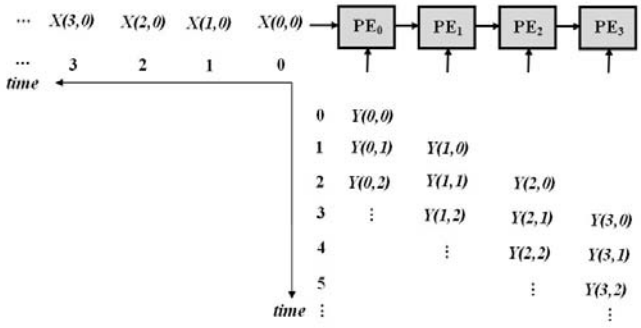

Fig. 8: Inputs timing for Design of [10] with K=4

VIII. Conclusion

The systematic technique presented in this work is used to explore the design space of linear processor arrays for the computation of similarity distance matrices. Six new processor arrays, in addition to a previously devised one, are obtained systematically. Time and area complexities of these seven architectures are compared and analyzed. Implementation results for the previously obtained architecture and one of our

TABLE II: Implementation results

	Design #1			Design of [10]		
#PEs	Area	Max. Frequency	Execution Time	Area	Max. Frequency	Execution Time
K	(# Slices)	(MHz)	(ms)	(# Slices)	(MHz)	(ms)
4	**44**	**238.72**	**3.29**	**99**	**167.44**	**4.69**
8	**93**	**234.85**	**3.34**	**168**	**175.68**	**4.44**
16	**184**	**224.41**	**3.5**	**318**	**184.09**	**4.27**
32	**373**	**212.72**	**3.69**	**644**	**185.04**	**4.25**

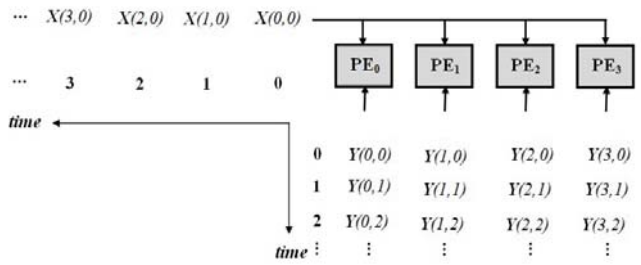

Fig. 9: Inputs timing for Design #1 with K=4

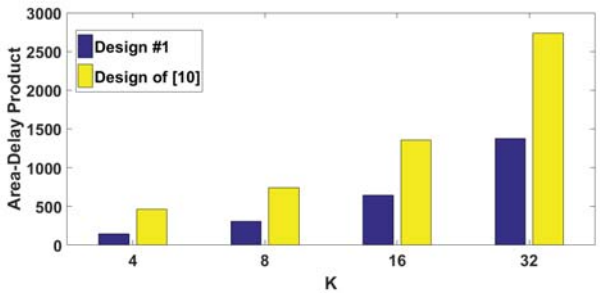

Fig. 10: Area-delay product of Design #1 and Design of [10] for different values of **K**

proposed architectures show that the proposed architecture achieves better performance in terms of speed and area.

Currently, we are investigating nonlinear scheduling and projection operations that allow for more control on the workload assigned to the processing elements, and on the number of processing elements to explore more design alternatives.

REFERENCES

[1] D. Singh and C. K. Reddy, "A survey on platforms for big data analytics," *Journal of Big Data*, vol. 2, no. 1, p. 1, 2014.

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

[3] 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.

[4] T.-M. Huang, V. Kecman, and I. Kopriva, *Kernel based algorithms for mining huge data sets*. Springer, 2006, vol. 1.

[5] S. B. Kotsiantis, I. Zaharakis, and P. Pintelas, "Supervised machine learning: A review of classification techniques," 2007.

[6] A. Choudhary, R. Narayanan, B. Ö. Ikyılmaz, G. Memik, J. Zambreno, and J. Pisharath, "Optimizing data mining workloads using hardware accelerators," in *Proc. of the Workshop on Computer Architecture Evaluation using Commercial Workloads (CAECW)*, 2007.

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

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

[9] 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, Aug 2017.

[10] 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.

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

[12] F. Gebali [El-Guibaly] 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.

[13] F. Gebali, *Algorithms and Parallel Computing*. John Wiley & Sons, 2011.

[14] Y. Terada, "Clustering for high-dimension, low-sample size data using distance vectors," *arXiv preprint arXiv:1312.3386*, 2013.

[15] 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 *Service Operations and Logistics, and Informatics (SOLI), 2013 IEEE International Conference on*, July 2013, pp. 372–381.