JOURNAL OF PARALLEL AND DISTRIBUTED COMPUTING 5, 1–25 (1988)

Systematic Designs of Buffers in Macropipelines of Systolic Arrays*

BENJAMIN W. WAH AND MOKHTAR ABOELAZE

Department of Electrical and Computer Engineering and the Coordinated Science Laboratory, University of Illinois at Urbana–Champaign, Urbana, Illinois 61801

AND

WEIJIA SHANG

School of Electrical Engineering, Purdue University, West Lafayette, Indiana 47907

Received March 1, 1986

In a macropipeline of systolic arrays, outputs of one systolic array in a given format have to be fed as inputs to another systolic array in a possibly different format. A common memory becomes a bottleneck and limits the number of systolic arrays that can be connected together. In this paper, we study designs of buffers to convert data from one format to another. The minimum number of buffers is determined by a dynamic-programming algorithm with $\Theta(n^2)$ computational complexity, where n is the problem size. A general-purpose converter to convert data from any distribution to any other in a subset of the possible data distributions is also proposed. Last, buffer designs for a macropipeline to perform feature extraction and pattern classification are used to exemplify the design process.

1. INTRODUCTION

The evolution in very-large-scale-integration (VLSI) technology has had a great impact on both computer architecture and digital signal processing. An important architectural approach resulting from the availability of inexpensive special-purpose VLSI circuits is the systolic array, which consists of multiple

* Research was supported by National Science Foundation Grant DMC-85 19649, Joint Services Electronics Contract N000014-84-C-0149, and a scholarship from the Egyptian Educational and Cultural Bureau.

regularly connected processing elements to exploit the potential of pipelining and multiprocessing [11, 10]. Several data items flowing along different pipes with the same or different rates may meet and interact. The processing elements operate synchronously; that is, each data item must stay in a processing element for one and only one clock cycle, and all necessary operands to be processed by a processing element in each computational step must arrive at this processing element simultaneously. The major advantage of systolic processing is that each data item, once accessed, will be used a number of times, and thus a high computational throughput can be achieved with a modest input/output bandwidth. Other advantages include modular expandability, extensive concurrency, simple and regular data and control flow, and simplicity and uniformity of processing cells.

In a large system, especially in real-time applications, a pool of systolic arrays of different types can be configured into a macropipeline to solve a given problem. A *macropipeline* is a pipeline of systolic arrays with the outputs of one array acting as inputs to another array in the pipe. Each stage of the pipe is a systolic array that performs one operation, such as matrix addition or multiplication. Such structure of macropipelines characterizes most image-processing algorithms [16, 17]. Examples include real-time vision system [15], analysis of motion [1], image reconstruction from projections [4], radar signal processing [2], air traffic control [7], pattern analysis and image database management [6], recursive filtering [18], and pattern recognition [8]. The Programmable Systolic Chip [5] and the Warp array processor [12] are examples of reconfigurable systolic arrays dedicated to handling compute-bound problems in image and digital signal processing. A number of these arrays can be used in a pipelined fashion to perform the various tasks in image and signal processing.

A *data distribution* of a systolic array is either the format of inputs fed into the systolic array or the format of outputs exiting the systolic array. The input data distribution of one systolic array may be different from the output data distribution of another; hence, when two systolic arrays are connected together, it may be necessary to convert the outputs of the systolic array that feeds data to the other into its required input data distribution. A conventional approach is to use a common memory to buffer the outputs of the systolic arrays, which becomes a bottleneck when many systolic arrays are sharing the common memory. Another approach is to design the systolic arrays such that the output format of one array is the same as the input format of the next array in the macropipeline and to connect the systolic arrays directly. This may not always be possible, especially when the macropipeline is reconfigurable. A third approach is to design a converter between two stages of the macropipeline, which consists of multiple buffers and a control unit to select the appropriate buffers for inputs and outputs [3]. This approach is exemplified by MOSAIC [14], a project carried out at ESL. The system consists of a statically scheduled

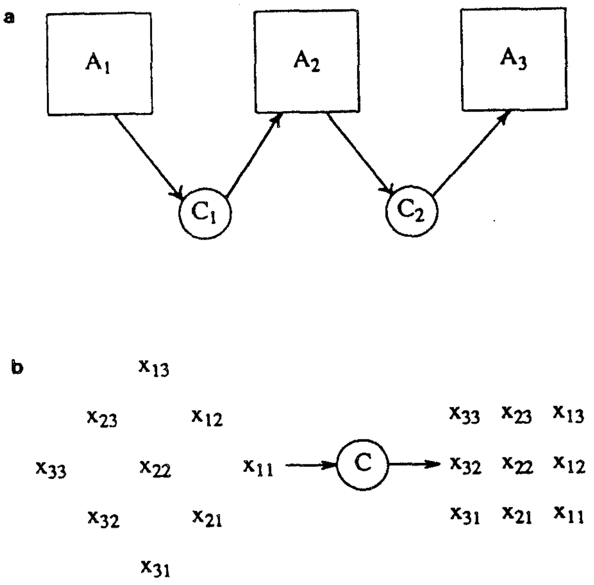


FIG. 1. Macropipelining of systolic arrays. (a) A macropipeline of systolic arrays. (b) Conversion of data from one distribution to another.

crossbar switch that connects multiple Warp processors, each with local memory modules, into a macropipeline. The local memory modules are used to store input data and restructure them into the required input format. Since Warp processors are reconfigurable general-purpose systolic arrays, the memory requirement for each module is not defined at design time; hence, all memory modules are of the same size. The interfacing of multiple systolic arrays was also addressed in an ad hoc fashion in the design of a 450-MFLOP systolic processor for adaptive beamforming in acoustic and signal processing applications at ESL [9].

The concept of using buffers to perform data conversion is illustrated in Fig. 1a. C_1 and C_2 are converters to convert the output data into the required input formats. Figure 1b shows this conversion. To convert data from distribution D_i to D_o, at least six buffers are needed. The first column of D_o cannot be output until the third column of D_i has arrived. Six buffers are needed to store the data in D_i such that elements in the first column of D_o are available in the buffers. Likewise, five buffers are needed for the second column, and three buffers are needed for the last. Note that the input and output rates may not be the same when the minimum number of buffers are used.

In this paper, we study designs of converters to interface systolic arrays in a macropipeline. The design depends on the type of macropipeline. A *static macropipeline* consists of a fixed pipeline of systolic arrays with a fixed function in each array. Hence, the conversion of data distributions between adjacent stages is fixed as well, and special-purpose converters are needed. In contrast, in a *dynamic macropipeline*, a subset of systolic arrays is selected from the pool and configured into a pipeline depending on the application. As the configuration of a dynamic macropipeline may not be fixed and data of different formats may be fed into a given array, general-purpose converters are needed here.

The objective of this paper is to provide a methodology to design an efficient converter for given input and output distributions. It is assumed that both inputs and outputs are two-dimensional arrays in which the elements are equally spaced along the rows and columns in the data distributions, that there are no duplicated data in the distribution, and that data can be described by two vectors to be discussed in Section 2.1. The macropipeline is asynchronous, and the interarrival times of data may be different for different systolic arrays. In the remaining sections, we study the minimum number of buffers for a given conversion [19], propose design procedures for general-purpose and special-purpose converters, and exemplify the design process.

2. Minimum Number of Buffers

A converter is made up of buffers, the interconnections among the buffers, and the necessary control hardware that issues signals to buffers to accept or send data at the proper times. $B_{\min}$ is defined as the minimum number of buffers in a converter to buffer incoming data before they are output. In this section, an algorithm is presented to find $B_{\min}$ for given input and output distributions.

2.1. *Data Distributions*

To describe different data distributions, two vectors are introduced here [13]. Suppose that the row and column indices of X are i and j, respectively. The row vector of X is defined as the directional distance between $x_{i,j}$ and $x_{i+1,j}$ and is denoted by $\vec{I}$. Similarly, the column vector of X, denoted by $\vec{J}$, is defined as the directional distance between $x_{i,j}$ and $x_{i,j+1}$. A data distribution with vectors $\vec{I}$ and $\vec{J}$ is denoted by $D(\vec{I}, \vec{J})$. Two data distributions are illustrated in Fig. 2.

The geometric layout of a data distribution can be described in the Cartesian plane. Without loss of generality, it is assumed that $x_{1,1}$ for both the input and output distributions is placed at the origin, and that data are moving in the direction of the positive x-axis. Vectors $\vec{I}$ and $\vec{J}$ used to define a data distribution determine the locations of its elements uniquely. $C_x(i, j)$ and $C_y(i, j)$ denote the x- and y-coordinates of element $x_{i,j}$. I_x and J_x are the projections of vectors $\vec{I}$ and $\vec{J}$ on the x-axis. Likewise, I_y and J_y are the corresponding projections on the y-axis. Then,

$$C_x(i,j) = (i-1)I_x + (j-1)J_x \tag{1}$$

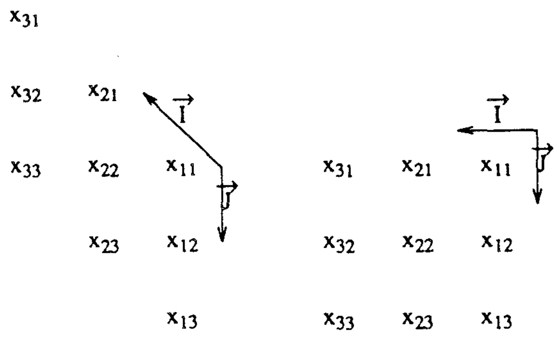


FIG. 2. Two data distributions and their corresponding vectors. (The first data distribution has five streams of dataflow, and the second one has three streams, assuming that data are moving from left to right.)

$$C_y(i,j) = (i-1)I_y + (j-1)J_y. \tag{2}$$

Note that if I_x, J_x, I_y, and J_y are integers, then the coordinates will be integers.

In the Cartesian-coordinate representation we have adopted, the x-coordinate indicates timing; that is, elements with the same x-coordinate arrive at (or depart from) the converter at the same time. Data with the smallest x-coordinate arrive at (or depart from) the converter first, while data with the largest x-coordinate arrive (or depart) last. The *ith* (*input or output*) *step* is defined as the set of elements in the (input or output) distribution with the x-coordinate equal to i.

2.2. *Finding the Minimum Number of Buffers*

A dynamic-programming formulation is developed to find $B_{\min}$, the minimum number of buffers to convert the data distribution from D_i to D_o. Let b_i be the number of buffers needed after the $(i-1)$st output step has been carried out and before the ith step of D_o can be output, while the necessary data to output in the ith output step are being received. In deriving b_i, it is assumed that all input data items are buffered before they can be output. Further, let B_i be the maximum number of buffers needed when the ith step of D_o is output, and let the boundary conditions be $B_0 = 0$. For the example in Fig. 1, before $x_{1,1}$, $x_{1,2}$, and $x_{1,3}$ can be output, b_1 (=6) buffers are needed to buffer the first three columns of D_i. Before the second output step of D_o can be carried out, data in the first four input steps with eight data items must have been received. Hence, b_2 is 5, assuming that the first output step has been completed. Similarly, $b_3 = 3$. As a result,

$$B_i = \begin{cases} 0, & i = 0 \\ \max\{b_i, B_{i-1}\}, & i = 1, 2, 3. \end{cases} \tag{3}$$

$$B_{\min} = B_3 = \max\{b_3, \max\{b_2, \max\{b_1, B_0\}\}\} = b_1 = 6. \tag{4}$$

Note that in deriving the minimum number of buffers, the input and output clocks may be running at different rates.

To allow a more precise formulation, two partitions on the data set $X = \{x_{i,j}: 1 \leqslant i,j \leqslant n\}$ and a partial ordering of these partitions are introduced.

An *input partition* partitions the input array X into N_i disjoint subsets, $\mathbf{I}_p$, $1 \leqslant p \leqslant N_i$, where

$$\mathbf{I}_p = \{x_{i,j} \mid C_x(i,j) = (i-1)I_x^i + (j-1)J_x^i = a_p\}, \qquad p = 1, \ldots, N_i. \tag{5}$$

and N_i is the number of input steps. $\vec{I}^i$ and $\vec{J}^i$ are the row and column vectors of the input distribution, respectively, and I_x^i and J_x^i are the corresponding projections on the x-axis. $\mathbf{I}_p$ represents the set of input elements with the same x-coordinate a_p. $\mathbf{I}_1$ is the set of input data that arrive at the converter first, and $\mathbf{I}_p$ is the pth arrival set.

An *output partition* partitions the output array X into N_o disjoint subsets $\mathbf{O}_k$, $1 \leqslant k \leqslant N_o$, where

$$\mathbf{O}_k = \{x_{i,j} \mid C_x(i,j) = (i-1)I_x^o + (j-1)J_x^o = a_k\}, \qquad k = 1, \ldots, N_o, \tag{6}$$

and N_o is the number of output steps for the output distribution. $\vec{I}^o$ and $\vec{J}^o$ are the vectors of the output distribution, respectively, and I_x^o and J_x^o are the corresponding projections on the x-axis. $\mathbf{O}_k$ represents output elements with the same x-coordinate a_k. $\mathbf{O}_1$ represents the set of data that depart from the converter first, and $\mathbf{O}_k$ is the kth departure set.

Figure 3a shows D_i and D_o of a 3 × 3 array X. Since X arrives in three steps, there are three input partitions (i.e., $N_i = 3$). Similarly, there are seven output partitions (i.e., $N_o = 7$) because the outputs depart in seven steps. Let $\mathbf{S}$ and $\mathbf{O}$ be the sets of input and output partitions and $\mathbf{\Pi}$ be their union.

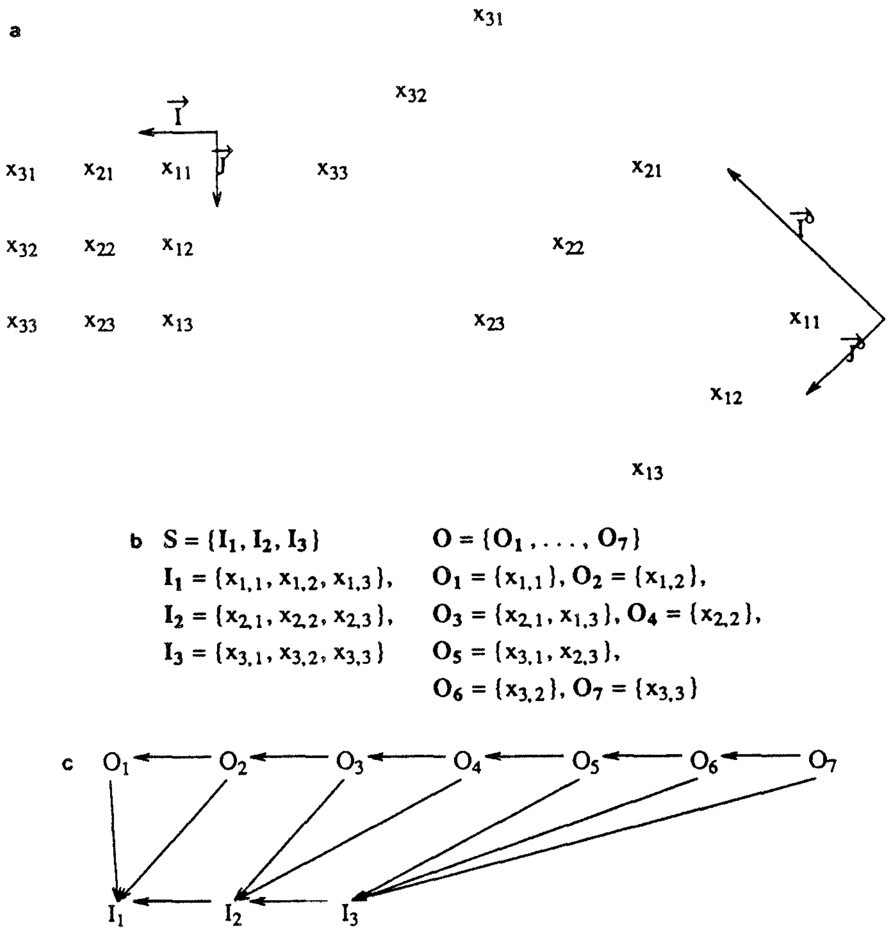
$$\mathbf{S} = \{\mathbf{I}_p \mid 1 \leqslant p \leqslant N_i\} \tag{7}$$

$$\mathbf{O} = \{\mathbf{O}_k \mid 1 \leqslant k \leqslant N_o\} \tag{8}$$

$$\mathbf{\Pi} = \mathbf{S} \cup \mathbf{O}. \tag{9}$$

For $\pi_i \in \mathbf{\Pi}$, $|\pi_i|$ represents the number of elements in π_i. The corresponding input and output partitions in Fig. 3a are shown in Fig. 3b.

The example in Fig. 3 shows that there exists a relationship between the $\mathbf{O}_k$'s and $\mathbf{I}_p$'s. A partial ordering "$\rightarrow$" can be defined on $\mathbf{\Pi}$ as follows. If $\mathbf{I}_k \rightarrow \mathbf{I}_p$, then data in $\mathbf{I}_p$ will arrive earlier than those of $\mathbf{I}_k$. If $\mathbf{O}_k \rightarrow \mathbf{O}_p$, then data in $\mathbf{O}_p$ will leave earlier than those of $\mathbf{O}_k$. Further, if $\mathbf{O}_k \rightarrow \mathbf{I}_p$, then data in $\mathbf{I}_p$ must arrive before data in $\mathbf{O}_k$ can depart. To output the elements in $\mathbf{O}_k$, all elements in $\mathbf{I}_p$'s such that $\mathbf{O}_k \cap \mathbf{I}_p \neq \varnothing$ must have arrived at the converter. In short,


a
$\vec{I}$
$\vec{J}$
x_{31} x_{21} x_{11}
x_{32} x_{22} x_{12}
x_{33} x_{23} x_{13}
x_{31}
x_{32}
x_{33}
x_{21}
x_{22}
x_{23}
x_{11}
x_{12}
x_{13}
b
c
O_1 O_2 O_3 O_4 O_5 O_6 O_7
I_1 I_2 I_3

$\mathbf{S} = \{\mathbf{I}_1, \mathbf{I}_2, \mathbf{I}_3\}$ $\mathbf{O} = \{\mathbf{O}_1, \ldots, \mathbf{O}_7\}$

$\mathbf{I}_1 = \{x_{1,1}, x_{1,2}, x_{1,3}\}$, $\mathbf{O}_1 = \{x_{1,1}\}$, $\mathbf{O}_2 = \{x_{1,2}\}$,

$\mathbf{I}_2 = \{x_{2,1}, x_{2,2}, x_{2,3}\}$, $\mathbf{O}_3 = \{x_{2,1}, x_{1,3}\}$, $\mathbf{O}_4 = \{x_{2,2}\}$,

$\mathbf{I}_3 = \{x_{3,1}, x_{3,2}, x_{3,3}\}$ $\mathbf{O}_5 = \{x_{3,1}, x_{2,3}\}$,

$\mathbf{O}_6 = \{x_{3,2}\}$, $\mathbf{O}_7 = \{x_{3,3}\}$

FIG. 3. Partitioning and partial ordering. (a) Input distribution D_i and output distribution D_o. (b) Input and output partitions. (c) Lattice for the partial ordering (transitive arcs are not shown).

(1) $\mathbf{I}_k \to \mathbf{I}_p$, if $k > p$;

(2) $\mathbf{O}_k \to \mathbf{O}_p$, if $k > p$;

(3) $\mathbf{O}_k \to \mathbf{I}_p$, if either $\mathbf{O}_k \cap \mathbf{I}_p \neq \emptyset$ or there exists an integer q such that $\mathbf{I}_q \cap \mathbf{O}_k \neq \emptyset$ and that $\mathbf{I}_q \to \mathbf{I}_p$.

The above definitions imply that if $\mathbf{O}_k \to \mathbf{I}_p$, then $\mathbf{O}_k \to \mathbf{I}_{p-1} \to \mathbf{I}_{p-2} \to \cdots \to \mathbf{I}_1$. The integer $q_k = p$ such that $\mathbf{O}_k \to \mathbf{I}_p$ and that $\mathbf{O}_k \not\to \mathbf{I}_{p+1}$ is defined as the *key number* for $\mathbf{O}_k$. Note that all the relationships among the $\mathbf{I}_p$'s and $\mathbf{O}_k$ will be known once q_k is found. The partial ordering of the partitions can be represented in a lattice. Fig. 3c shows the lattice of the partial ordering for the example in Fig. 3a. For instance, $q_3 = 2$ is the key number for $\mathbf{O}_3$ since $\mathbf{O}_3 \cap \mathbf{I}_2 = \{x_{2,1}\}$, and $\mathbf{O}_3$ can be output once elements in $\mathbf{I}_2$ have arrived.

To use dynamic programming to find $B_{\min}$, $\mathbf{O}_1, \mathbf{O}_2, \ldots, \mathbf{O}_k$ are examined sequentially. It is assumed that the inputs and outputs may be driven by different clocks; that is, the minimum amount of data are input to generate the necessary outputs. If q_k is the key number for $\mathbf{O}_k$, then $\mathbf{I}_1, \mathbf{I}_2, \ldots, \mathbf{I}_{q_k}$ must have arrived at the converter before elements in $\mathbf{O}_k$ can depart. The reason is that either $\mathbf{I}_p$, $1 \leq p \leq q_k$, contains data in $\mathbf{O}_k$, or $\mathbf{I}_p$ does not contain data in $\mathbf{O}_k$ but $\mathbf{I}_{q_k} \rightarrow \mathbf{I}_p$. Therefore, elements in the set $\mathbf{I}_1 \cup \mathbf{I}_2 \cup \cdots \cup \mathbf{I}_{q_k}$ that remain after $\mathbf{I}_{q_k}$ has arrived and $\mathbf{O}_{k-1}$ has left must be buffered. In other words, b_k, the number of buffers needed, is

$$b_k = \sum_{i=1}^{q_k} |I_i| - \sum_{i=1}^{k-1} |\mathbf{O}_i|, \qquad k = 1, \ldots, N_o. \tag{10}$$

By the principle of optimality, which states that an optimal sequence of decisions has the property that whatever the initial state and decision are, the remaining decisions must constitute an optimal decision sequence with regard to the state resulting from the first decision, we can formulate the problem in dynamic programming as follows.

$$|\mathbf{O}_0| = 0; \qquad q_0 = 0; \tag{11a}$$

$$b_k = \begin{cases} 0, & k = 0 \\ b_{k-1} - |\mathbf{O}_{k-1}| + \sum_{j=q_{k-1}+1}^{q_k} |\mathbf{I}_j|, & k = 1, \ldots, N_o \end{cases} \tag{11b}$$

$$B_k = \begin{cases} 0, & k = 0 \\ \max(b_k, B_{k-1}), & k = 1, \ldots, N_o. \end{cases} \tag{12}$$

Note that the summation in Eq. (11b) is zero if the lower limit is greater than the upper limit.

To establish the partial ordering of partitions, a counter is used to count the number of elements in each partition, and the key number q_k is kept for each $\mathbf{O}_k$. $C_x^i(i, j)$ and $C_x^o(i, j)$ are computed for every element $x_{i,j}$ in the input and output distributions. If $x_{i,j}$ is in $\mathbf{O}_k$ and $\mathbf{I}_p$, then $\mathbf{O}_k \rightarrow \mathbf{I}_p$, and the counters for $\mathbf{O}_k$ and $\mathbf{I}_p$ are incremented. q_k is updated to p if p is larger than the previous value of q_k. The algorithm to compute the partial ordering is shown in Fig. 4. The computational complexity of the algorithm is $O(n^2)$, since all elements in D_i and D_o must be considered. A better algorithm with a computational complexity of $O(\max\{N_i, N_o\} \cdot \min\{I_x^o, J_x^o\})$ can be devised but will not be presented here.

The example in Fig. 3a is used to illustrate the algorithm. Initially, all key numbers are initialized to zeros, $N_i = 3$, and $N_o = 7$. Since $C_x^o(1, 1) = 0$, $C_x^i(1, 1) = 0$, hence, $x_{1,1} \in \mathbf{O}_1$, and $x_{1,1} \in \mathbf{I}_1$. q_1 and the counters for $\mathbf{O}_1$ and $\mathbf{I}_1$ are updated to ones. Similarly, it is found that $x_{1,3} \in \mathbf{O}_3$ and $x_{1,3} \in \mathbf{I}_1$. The counters for $\mathbf{O}_3$ and $\mathbf{I}_1$ are incremented, and q_3 is set to one. For $x_{2,1}$, it is

```
procedure compute_partial_ordering;

/* Inputs:   $\vec{I}^i$ and $\vec{J}^i$: vectors for input distribution;
             $C_x^i(1,1) = C_x^o(1,1) = 0$, and all $C_x^i(i,j)$ and $C_x^o(i,j)$ are integers;
             $\vec{I}^o$ and $\vec{J}^o$: vectors for output distribution;
             $N_i$: number of input steps;
             $N_o$: number of output steps;
   Outputs:  Three arrays $\Phi(N_0)$, $Q(N_0)$, $S(N_i)$, where
             $\Phi(k) = |O_k|$, the number of elements in $O_k$;
             $Q(k) = q_k$, the key number for $O_k$;
             $S(p) = |I_p|$, the number of elements in $I_p$.  */
(1)  Initialize $\Phi$, Q, and S to zeroes.
(2)  for i=1 to n do [
         for j=1 to n do [
             $k := C_x^o(i,j) + 1$;   $\Phi(k) := \Phi(k) + 1$;
             $p := C_x^i(i,j) + 1$;   $S(p) := S(p) + 1$;
             $Q(k) := \max\{Q(k), p\}$
         ]
     ]
```

FIG. 4. Algorithm to compute the partial ordering of partitions.

found that $x_{2,1} \in \mathbf{I}_2$ and that $x_{2,1} \in \mathbf{O}_3$. The counters for $\mathbf{O}_3$ and $\mathbf{I}_2$ are incremented, and q_3 is set to $\max\{q_3, 2\} = 2$. Likewise, the remaining elements in X can be examined. The results of applying the algorithm in Fig. 4 are

$$\Phi = [1, 1, 2, 1, 2, 1, 1]; \qquad Q = [1, 1, 2, 2, 3, 3, 3]; \qquad S = [3, 3, 3].$$

Applying Eq. (11) results in $[b_k] = [3, 2, 4, 2, 4, 2, 1]$. From Eq. (12), we obtain $B_7 = 4$.

3. COMBINATIONS OF DATA DISTRIBUTIONS

In this section, we discuss some properties of data distributions that are useful for designing the converters. As mentioned before, a data distribution is characterized by two nonparallel vectors. Two data distributions $D(\vec{I}^1, \vec{J}^1)$ and $D(\vec{I}^2, \vec{J}^2)$ are said to be *equivalent* (or belong to the same *equivalence class*) if

$$I_x^1 = I_x^2 \qquad \text{and} \qquad J_x^1 = J_x^2 \tag{13a}$$

and

$$(I_x^1 J_y^1 - I_y^1 J_x^1) \cdot (I_x^2 J_y^2 - I_y^2 J_x^2) > 0, \tag{13b}$$

where I^i_x (resp. J^i_x) is the projection of $\vec{I}^{\,i}$ (resp. $\vec{J}^{\,i}$) on the x-axis. The first condition (Eq. (13a)) ensures that the data distributions have the same projections on the x-axis. Consequently, the orders in which data arrive at the systolic array for the two data distributions are identical. The second condition (Eq. (13b)) ensures that the data arriving at the systolic array at the same time have the same permutations. Note that for the two vectors $\vec{I}$ and $\vec{J}$, if θ is the angle in a clockwise direction from $\vec{I}$ to $\vec{J}$, then $\sin(\theta) = (I_xJ_y - I_yJ_x)/(|\vec{I}|\cdot|\vec{J}|)$. The number of streams of data flowing into the systolic array for two equivalent data distributions may not be equal and can range from n to $(2n - 1)$. As an example, the two data distributions in Fig. 2 are equivalent, but have different numbers of streams of dataflow.

The following theorem shows the number of possible equivalence classes of data distributions.

THEOREM. *There are $\Omega(2^n)$ equivalence classes of data distributions for an $n \times n$ array of data.*

Proof. In proving the number of equivalence classes, only the projections of the vectors on the x-axis have to be considered. Without loss of generality, assume that $\vec{J}$ is not orthogonal to the x-axis. From the x-projections of the first row of data, $C_x(1, 1), C_x(1, 2), \ldots, C_x(1, n)$, it is necessary to determine the number of possible x-projections for the remaining rows. Consider the x-projection of $x_{2,1}$. Assuming that $C_x(2, 1) \geqslant C_x(1, 1)$, there are $2n$ possible positions for $C_x(2, 1)$, namely, $C_x(2, 1) = C_x(1, i)$, $i = 1, \ldots, n$; $C_x(1, i) < C_x(2, 1) < c_x(1, i + 1)$, $i = 1, \ldots, n - 1$; and $C_x(2, 1) > C_x(1, n)$. Suppose that $C_x(1, 1) < C_x(2, 1) < C_x(1, 2)$; then there are three possibilities for $C_x(3, 1)$, namely, $C_x(3, 1) = C_x(1, 2)$, $C_x(2, 1) < C_x(3, 1) < C_x(1, 2)$, and $C_x(1, 2) < C_x(3, 1) < C_x(1, 3)$ (see Fig. 5). When $C_x(3, 1) = C_x(1, 2)$, the positions of the remaining elements are determined. However, when either $C_x(2, 1) < C_x(3, 1) < C_x(1, 2)$ or $C_x(1, 2) < C_x(3, 1) < C_x(1, 3)$, then $C_x(4, 1)$ can fall into three possible ranges, as shown on the second level of the tree in Fig. 5. The same argument can be applied to the remaining levels of the tree for $x_{5,1}, \ldots, x_{n,1}$. In level Q, $1 < Q < n - 3$, there are 2^{Q-1} terminals, while in level $n - 2$, there are $3 \cdot 2^{n-3}$ terminals. The total number of terminals is

$$\sum_{Q=1}^{n-3} 2^{Q-1} + 3 \cdot 2^{n-3} = 2^{n-3} - 1 + 3 \cdot 2^{n-3} = 4 \cdot 2^{n-3} - 1. \tag{14}$$

A similar argument can be made when $C_x(1, 1) > C_x(2, 1)$ or $C_x(2, 1) > C_x(1, 2)$. Since each of the above data distributions belongs to a distinct equivalence class, the total number of possible data distributions is $\Omega(2^n)$. ■

It is practically impossible to design a general-purpose converter to perform all the possible data transformations. Some restrictions are necessary to reduce the space of data distributions.

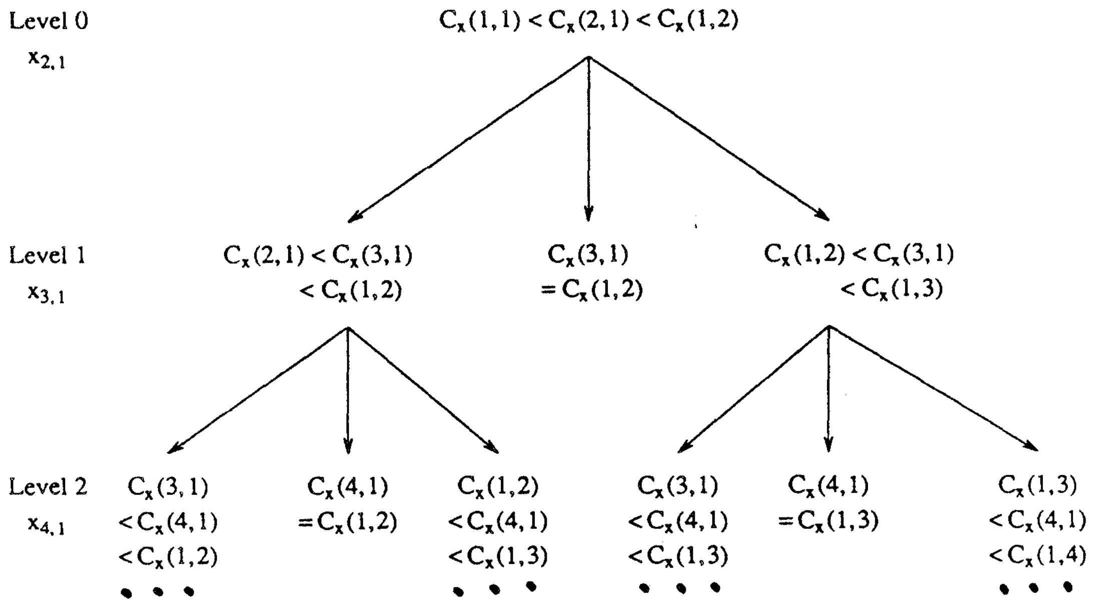

Level 0
$x_{2,1}$
$C_x(1,1) < C_x(2,1) < C_x(1,2)$
Level 1
$x_{3,1}$
$C_x(2,1) < C_x(3,1) < C_x(1,2)$
$C_x(3,1) = C_x(1,2)$
$C_x(1,2) < C_x(3,1) < C_x(1,3)$
Level 2
$x_{4,1}$
$C_x(3,1) < C_x(4,1) < C_x(1,2)$
$C_x(4,1) = C_x(1,2)$
$C_x(1,2) < C_x(4,1) < C_x(1,3)$
$C_x(3,1) < C_x(4,1) < C_x(1,3)$
$C_x(4,1) = C_x(1,3)$
$C_x(1,3) < C_x(4,1) < C_x(1,4)$
• • •
• • •
• • •
• • •

FIG. 5. Possible positions for $x_{2,1}, x_{3,1}, \ldots, x_{n,1}$.

If vectors $\vec{I}$ and $\vec{J}$ are restricted to have only unitary or zero projections on the x- and y-axes, then there will be eight possible directions for $\vec{I}$ pointing at 0, 45, 90, 135, 180, 225, 270, and 315°. For each direction of $\vec{I}$, there are six possible directions for $\vec{J}$, excluding the cases in which $\vec{I}$ and $\vec{J}$ are pointing in the same or opposite directions. Thus, there are $8 \times 6 = 48$ possible combinations of data distributions (Fig. 6a). Out of these 48 cases, there are only 16 equivalence classes (distributions in the same column of Fig. 6a belong to the same equivalence class).[1] Data distributions in class i', $1 \leqslant i' \leqslant 8$, can further be combined with the corresponding data distributions in class i, $1 \leqslant i \leqslant 8$, into a new equivalence class if a reversal circuit is available to reverse the order of data arriving simultaneously at the converter. The resulting eight standard distributions are shown in Fig. 6b.

4. GENERAL-PURPOSE CONVERTERS

In this section, we discuss the design of a general-purpose converter that can convert data from any distribution to any other distribution provided that the vectors representing the data distributions have zero or unitary projections on the x- and y-axes. We will give the mathematical representation of this converter as a series of transformations applied to the input data distribution to produce the required output distribution. It is assumed that vector

[1] Note that elements in each equivalence class are not unique according to Eq. (13b). Figure 6a shows one of the possible sets of equivalence classes.

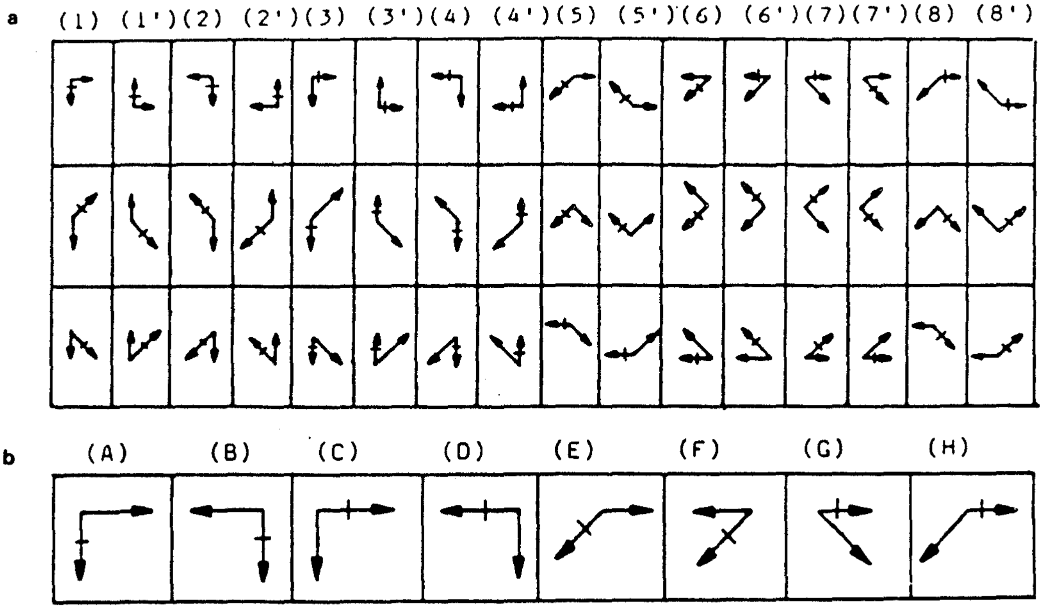


FIG. 6. Possible data distributions when $\vec{I}$ and $\vec{J}$ have either zero or unitary projections on the x- and y-axes. (The arrow with a bar across it is $\vec{I}$; the other is $\vec{J}$.) (a) All possible combinations of data distributions. (b) Eight standard output data distributions.

$\vec{I}$ is represented by the corresponding projections on the x- and y-axes. That is,

$$\vec{I} = I_x\vec{x} + I_y\vec{y} = [I_x, I_y]^T. \tag{15}$$

$\vec{J}$ can be represented similarly. The data-distribution vectors are represented as a 2 × 2 matrix $D = [\vec{I}, \vec{J}]$. A transformation process T is a 2 × 2 matrix, and a transformation on a data distribution is the product of the transformation matrix and the corresponding matrix representation of the data-distribution vectors. It is further assumed that the input distribution $D_i(\vec{I}^i, \vec{J}^i)$ and the output distribution $D_o(\vec{I}^o, \vec{J}^o)$ are given, and that they belong to one of the 48 data distributions in Fig. 6a. Figure 7 shows the transformation process.

The first transformation is on reducing the number of data streams from the outputs of the previous stage of the macropipeline. It is assumed that the input matrix enters the converter in n streams; that is, the distributions in the first row or columns (5) through (8') in the third row of Fig. 6a are used. If the output data from the previous stage require more than n streams, then both vectors $\vec{I}^i$ and $\vec{J}^i$ of this data distribution must have nonzero x- and y-projections; that is, the distribution belongs to those in the second row or columns (1) through (4') in the third row of Fig. 6a. In this case, the output data are multiplexed into n streams using n multiplexers before they are output from the previous stage (Fig. 8a). This multiplexing is equivalent to a linear transformation T_1.

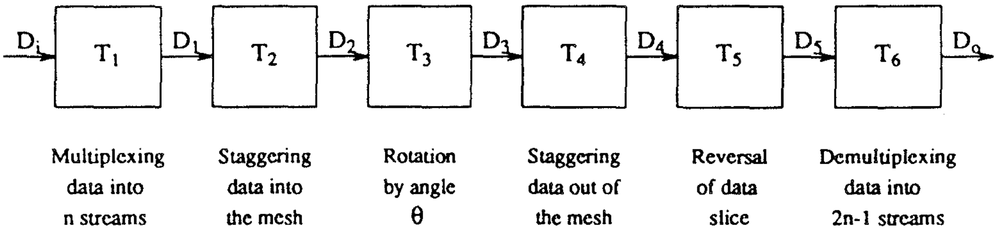

D_i
T_1
D_1
T_2
D_2
T_3
D_3
T_4
D_4
T_5
D_5
T_6
D_o
Multiplexing data into n streams
Staggering data into the mesh
Rotation by angle θ
Staggering data out of the mesh
Reversal of data slice
Demultiplexing data into 2n-1 streams

FIG. 7. The sequence of transformations in the conversion process.

$$T_1 = \begin{bmatrix} 1 & 0 \\ \alpha_1 & \beta_1 \end{bmatrix}, \tag{16}$$

which changes the data distribution in the second row or columns (1) through (4′) in the third row of Fig. 6a into that of the first row or columns (5) through (8′) in the third row. The data distribution after multiplexing can be represented by $D_1(\vec{I}^{\,1}, \vec{J}^{\,1})$.

$$D_1 = T_1 \cdot D_i,$$

$$\text{where} \begin{cases} \alpha_1 = \dfrac{-I^i_y J^i_y}{I^i_x J^i_y - I^i_y J^i_x}, & \beta_1 = -\alpha_1 \dfrac{I^i_x}{I^i_y} & \text{if } I^i_x, I^i_y, J^i_x, J^i_y \neq 0 \\ \alpha_1 = -\dfrac{J^i_y}{J^i_x}, & \beta_1 = 1 & \text{if } I^i_x = 0, J^i_x \neq 0, J^i_y \neq 0 \\ \alpha_1 = -\dfrac{I^i_y}{I^i_x}, & \beta_1 = 1 & \text{if } I^i_x \neq 0, I^i_y \neq 0, J^i_x = 0 \\ \alpha_1 = 0, & \beta_1 = 1 & \text{otherwise.} \end{cases} \tag{17}$$

The $n \times n$ array of data is routed into an $n \times n$ mesh of buffers with four-neighbor connections until they are filled (Fig. 8c). If the data distribution belongs to one of those in columns (5) through (8′) in rows 1 and 3 of Fig. 6a, then it needs to be transformed into one of the data distributions in column (1) through (4′) in row 1, which is the distribution of data that can be stored in the mesh of buffers. This transformation is represented by T_2.

$$T_2 = \begin{bmatrix} \alpha_2 & \beta_2 \\ 0 & 1 \end{bmatrix}, \tag{18}$$

which transforms $D_1(\vec{I}^{\,1}, \vec{J}^{\,1})$ into $D_2(\vec{I}^{\,2}, \vec{J}^{\,2})$.

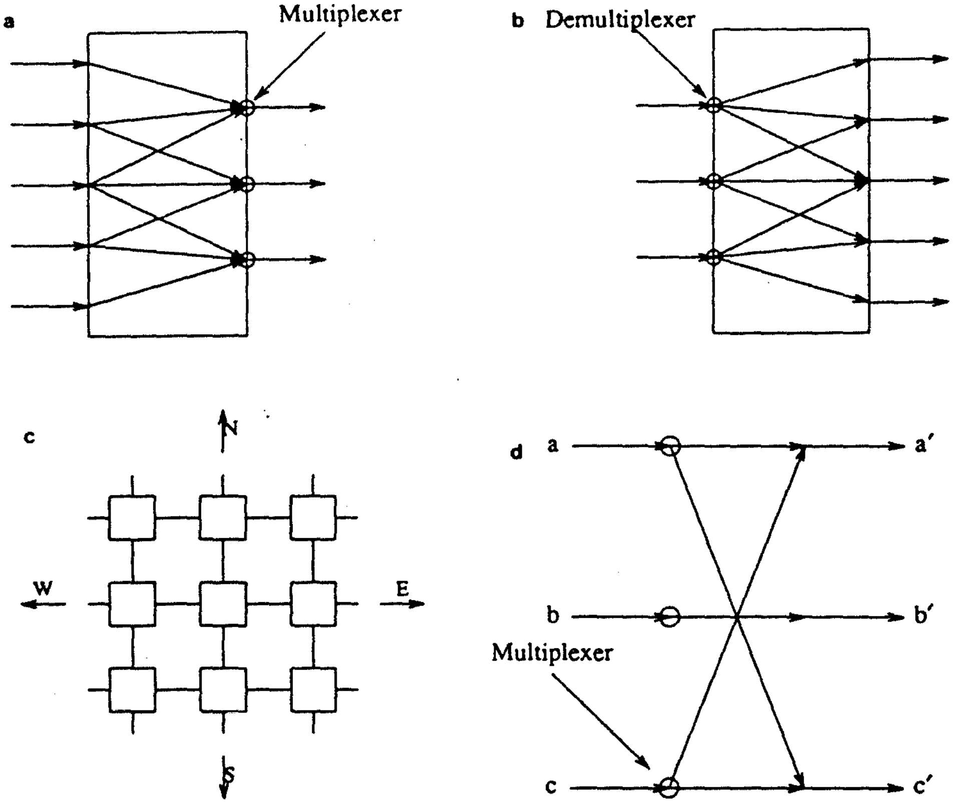


FIG. 8. Architecture of the general-purpose converter ($n = 3$). (a) Multiplexing data from ($2n - 1$) streams into n streams using n multiplexers ($n = 3$). (b) Demultiplexing data from n streams into ($2n - 1$) streams using n demultiplexers ($n = 3$). (c) 3×3 mesh of buffers. (d) Reversal network.

$$D_2 = T_2 \cdot D_1, \qquad \text{where} \qquad \begin{cases} \alpha_2 = \dfrac{1}{|I^1_x|}, \beta_2 = -\alpha_2 \dfrac{J^1_x}{J^1_y} & \text{if} \quad I^1_y = 0 \\ \alpha_2 = \dfrac{1}{|J^1_x|}, \beta_2 = -\alpha_2 \dfrac{I^1_x}{I^1_y} & \text{if} \quad J^1_y = 0. \end{cases} \tag{19}$$

The two-dimensional interconnections in the buffers allow data to be shifted in one of the four directions. Data are input in one direction and may come out from any one of the four directions. Accessing data in one of the four directions can be represented as a rotation of vectors $\vec{I}$ and $\vec{J}$ by an angle θ_3, where θ_3 is 0, 90, 180, or 270°. This rotation can be represented as transformation T_3.

$$T_3 = \begin{bmatrix} \cos(\theta_3) & -\sin(\theta_3) \\ \sin(\theta_3) & \cos(\theta_3) \end{bmatrix}, \qquad \theta_3 = 0, 90, 180, 270°, \tag{20}$$

which transforms $D_2(\vec{I}^2, \vec{J}^2)$ into $D_3(\vec{I}^3, \vec{J}^3)$.

$$D_3 = T_3 \cdot D_2, \tag{21}$$

where

$$\theta_3 = \begin{cases} 90^\circ & \text{if} \quad (I_x^2, I_y^o = 0 \text{ and } I_x^o = -I_y^2) \quad \text{or} \\ & \qquad (I_x^o, I_y^2 = 0 \text{ and } J_x^o = -J_y^2) \quad \text{or} \\ & \quad (J_x^2, J_y^o = 0 \text{ and } J_x^o = -J_y^2) \quad \text{or} \\ & \qquad (J_y^2, J_x^o = 0 \text{ and } I_x^o = -I_y^2) \\ 180^\circ & \text{if} \quad (I_x^o = -I_x^2 \text{ and } I_y^2 = 0 \text{ and } |J_y^o| = |J_y^2|) \quad \text{or} \\ & \quad (J_x^o = -J_x^2 \text{ and } J_y^2 = 0 \text{ and } |I_y^2| = |I_y^o|) \\ 270^\circ & \text{if} \quad (I_x^2, I_y^o = 0 \text{ and } I_x^o = I_y^2) \quad \text{or} \\ & \qquad (I_y^2, I_x^o = 0 \text{ and } J_x^o = I_y^2) \quad \text{or} \\ & \quad (J_x^2, J_y^o = 0 \text{ and } J_x^o = J_y^2) \quad \text{or} \\ & \qquad (J_y^2, J_x^o = 0 \text{ and } I_x^o = I_y^2) \\ 0^\circ & \quad \text{otherwise.} \end{cases} \tag{22}$$

T_3 is equivalent to transforming one of the distributions (1) through (4′) in the first row of Fig. 6a to another one in the same set.

In conjunction with the rotation, the shifts of data from the buffers may be controlled by different clocks to allow staggering of data in different rows. This is equivalent to transforming one of the distributions (1) through (4′) in the first row of Fig. 6a into one of the distributions in columns (5) thru (8′) in rows 1 and 3. This transformation can be represented by T_4.

$$T_4 = \begin{bmatrix} \alpha_4 & \beta_4 \\ 0 & 1 \end{bmatrix}, \tag{23}$$

where β_4 is the time difference between the output of the first element in row i and the first element in row $(i + 1)$. T_4 transforms $D_3(\vec{I}^3, \vec{J}^3)$ into $D_4(\vec{I}^4, \vec{J}^4)$.

$$D_4 = T_4 \cdot D_3, \qquad \text{where} \quad \begin{cases} \alpha_4 = I_x^o/I_x^3, & \beta_4 = J_x^o/J_y^3 \quad \text{if} \quad J_y^3 \neq 0 \\ \alpha_4 = J_x^o/J_x^3, & \beta_4 = I_x^o/I_y^3 \quad \text{if} \quad I_y^3 \neq 0. \end{cases} \tag{24}$$

Note that if the distributions are limited to those in Fig. 6a, then $\beta_4 = 0$ or ± 1. However, T_4 can also be applied to more general data distributions that will be discussed in Section 6.

Next, data may be routed through a reversal network to obtain the proper permutation (Fig. 8d). The reversal network maps data with output distribution (i) into that of (i') in Fig. 6a. This can be represented by a transformation T_5.

$$T_5 = \begin{bmatrix} 1 & 0 \\ 0 & \alpha_5 \end{bmatrix}, \tag{25}$$

which maps $D_4(\vec{I}^4, \vec{J}^4)$ into $D_5(\vec{I}^5, \vec{J}^5)$.

$$D_5 = T_5 \cdot D_4, \qquad \text{where} \qquad \alpha_5 = \begin{cases} \dfrac{J_y^4}{|J_y^4|}\dfrac{J_y^o}{|J_y^o|} & \text{if} \quad J_y^4 \neq 0 \\[2ex] \dfrac{I_y^4}{|I_y^4|}\dfrac{I_y^o}{|I_y^o|} & \text{if} \quad I_y^4 \neq 0. \end{cases} \tag{26}$$

Before data exit the converter and are sent into the systolic array in the macropipeline, they may be demultiplexed by n demultiplexers from n streams into $(2n - 1)$ streams as shown in Fig. 8b. This can be represented as a transformation T_6.

$$T_6 = \begin{bmatrix} 1 & 0 \\ \alpha_6 & \beta_6 \end{bmatrix}, \tag{27}$$

which maps $D_5(\vec{I}^5, \vec{J}^5)$ into the output distribution $D_o(\vec{I}^o, \vec{J}^o)$.

$$D_o = T_6 \cdot D_5, \qquad \text{where} \qquad \begin{cases} \alpha_6 = \dfrac{I_y^o}{I_x^5}, \quad \beta_6 = 1 - \alpha_6 \dfrac{J_x^5}{J_y^5} & \text{if} \quad I_y^5 = 0 \\[2ex] \alpha_6 = \dfrac{J_y^o}{J_x^5}, \quad \beta_6 = 1 - \alpha_6 \dfrac{I_x^5}{I_y^5} & \text{if} \quad J_y^5 = 0. \end{cases} \tag{28}$$

The transformations described above are sufficient to transform any one of the 48 distributions in Fig. 6a to any other distribution in the same figure. By using multiplexers (T_1) and by controlling the timing of different rows of data input into the mesh of buffers (T_2), any one of the 48 distributions in Fig. 6a can be transformed into a distribution represented by vectors in the x- and y-directions only. Note that these distributions belong to one of those in columns (1) through (4') in row 1 of Fig. 6a. To transform between any two distributions represented by vectors in the x- and y-directions, a rotation of angle θ, where θ is 0, 90, 180, or 270°, is needed. This can be achieved by selecting the direction to output the data in the mesh of buffers (T_3) and a reversal network (T_5). Likewise, by using demultiplexers (T_6) and by controlling the timing of the different rows of data output from the mesh of buffers (T_4), one of the data distributions in columns (1) through (4′) in row 1 of Fig. 6a can be transformed into any one of the 48 distributions.

The above design requires the entire matrix to be stored in the buffers before they are output. This simplifies the control but increases the delay. An alternative design uses demultiplexers to input data into selected buffers other than those on the perimeter. n demultiplexers, $d_1, d_2, \ldots, d_n$, are added to the n rows of buffers in Fig. 8c. In the resulting design shown in Fig. 9, d_1 and d_n are two-way demultiplexers, while the rest are four-way demultiplexers. For buffers in row i, $1 < i < n$, the four output lines of d_i are connected to cells 1, i, $(n - i + 1)$, and n. These connections are used to adjust the dataflow by outputting data as soon as possible and to obtain output distributions in columns (5) through (8′) in rows 1 and 3 of Fig. 6a. Note that this is equivalent to applying T_4 to the data distribution with $\beta_4 = 1$ if we route d_i to cell $(n - i + 1)$ and with $\beta_4 = -1$ if we route d_i to cell i. For example, to convert from input distribution (1) to output distribution (5) in the first row of Fig. 6a, demultiplexer d_i is connected to cell $(n - i + 1)$, $1 \leqslant i \leqslant n$. Elements in the first row will stay in the buffers for one time unit, while elements in the ith row will go through i buffers and, hence, will stay in the buffers for i time units. Data will be output in the eastern direction.

5. Special-Purpose Converters

In this section, we discuss the heuristic design of special-purpose converters. An optimal design of these converters is difficult because they are problem dependent.

The conversion between any pair of the eight standard distributions in Fig. 6b is straightforward and is illustrated in the following examples. To convert from distribution (A) to distributions (B), (C), or (D), n^2 buffers are needed.

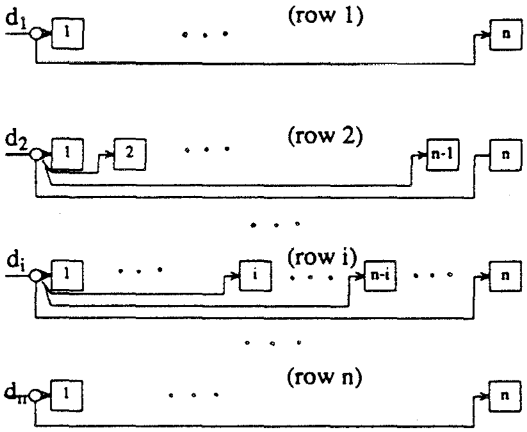


FIG. 9. Organization of buffers for the modified general-purpose converter.

The input data are propagated from left to right and are output in the western, southern, or northern direction after the buffers are filled. To convert from distribution (A) to (E), $n(n + 1)/2$ buffers are arranged as shown in Fig. 10a. The conversion from distribution (A) to (F) needs n^2 buffers. Data are output from the west after the buffers are filled. Data in row i are output one step after data in row $(i + 1)$. The conversion from (A) to (G) is similar to that from (A) to (E), and that from (A) to (H) is similar to that from (A) to (F). The conversion from distribution (E) to (A) requires $n(n + 1)/2$ buffers (Fig. 10b). For the conversions from distribution (E) to (B) or (C), n^2 buffers are needed. The conversion from (E) to (D) is similar to that in Fig. 1b. The conversion from (E) to (F) requires n^2 buffers, and data in row i are output one step ahead of data in row $(i + 1)$. The conversions from (E) to (G) or (H) are similar to that from (E) to (F).

The design of a special-purpose converter between data distributions not defined in Fig. 6 may be complicated, and a heuristic procedure is proposed here. First, $B_{\min}$, the minimum number of buffers, is found by the algorithm in Section 2. A feasible control circuit with $B_{\min}$ buffers is then searched. The control circuit contains demultiplexers that are individually controlled by a stored microprogram. If a feasible solution cannot be found easily or if the control circuit is too complex, then more buffers are added, and the procedure is repeated.

6. Data Conversion in Feature Extraction and Pattern Classification

In this section, we give an example on interfacing the systolic arrays in a macropipeline using converters. Specifically, the problem on feature extraction and pattern classification will be used [18, 8].

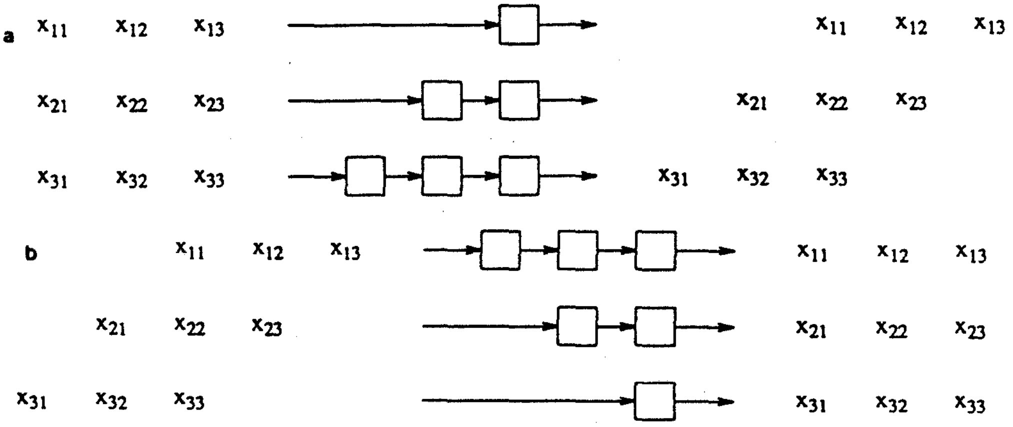


FIG. 10. Special-purpose converters. (a) Conversion from data distribution (A) to (E) ($n = 3$). (b) Conversion from data distribution (E) to (A) ($n = 3$).

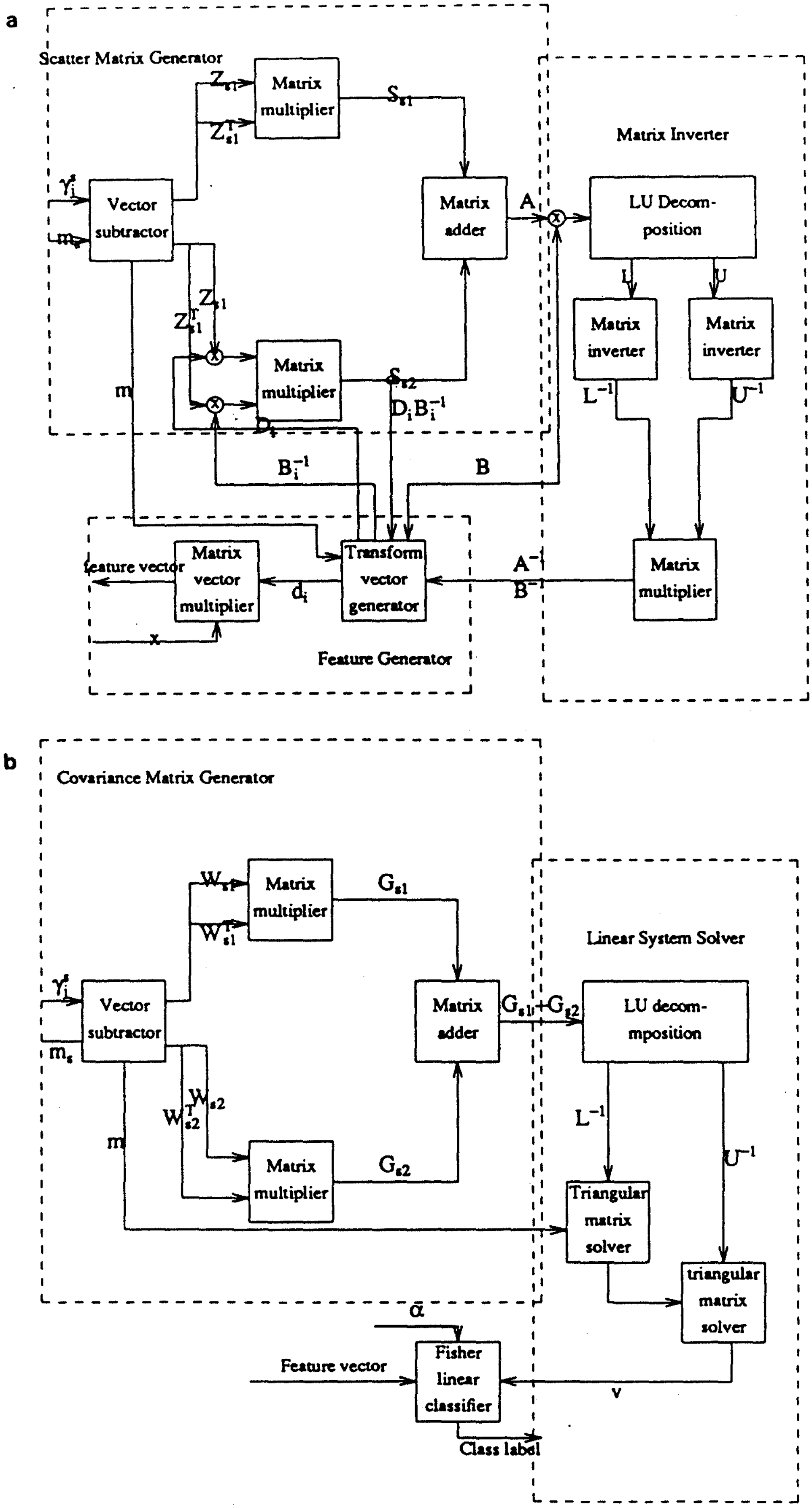
In feature extraction, given an $m' \times 1$ input vector $\mathbf{x}$, the feature extractor has to produce a set of m' transformation vectors, $\mathbf{D} = \{\mathbf{d}_i \mid i = 1, \ldots, m'\}$ using S, a set of training samples with known classes, where d_i is an $n \times 1$ column vector, m_s is the sample mean of class s, and γ_j^s is the jth training feature vector of class s. The output of the extractor is the feature vector $\mathbf{y} = \mathbf{D} \cdot \mathbf{x}$. Figure 11a shows the schematic design of a VLSI feature extractor, which has a macropipeline of matrix multiplication, *LU* decomposition, and triangular-matrix inversion.

For the pattern classifier, it is necessary to compute the feature offset vector $\mathbf{m} = \mathbf{m}_{s_1} - \mathbf{m}_{s_2}$, solve the linear system $(\mathbf{G}_{s_1} + \mathbf{G}_{s_2}) \cdot \mathbf{V} = \mathbf{m}$ for the discriminant vector $\mathbf{V}$, and use $\mathbf{V}$ to compute the discriminant function $\mathbf{F}(\mathbf{y})$ and classify the vector $\mathbf{y}$. Figure 11b shows a schematic diagram for the VLSI feature classifier, which has a macropipeline of matrix multiplication, *LU* decomposition, and triangular-system solver.

Figure 12 shows a fast systolic array to multiply two $n \times n$ matrices in a pipelined fashion ($n = 4$). The outputs of the systolic array exit in $(2n - 1)$ streams and are multiplexed by the converter into n streams. Figure 13 shows the systolic array for *LU* decomposition [10] and the associated input and output data formats. Although the outputs of the matrix-multiplication systolic array are in the same format as the inputs of the *LU*-decomposition systolic array, the outputs of the multiplication systolic array are multiplexed from $(2n - 1)$ streams into n streams to decrease the number of connections between the two chips. Hence, it is necessary to demultiplex the input data into $(2n - 1)$ streams in the *LU*-decomposition array. Note that this conversion is not needed if the two systolic arrays are on the same chip.

Figure 14 shows the triangular-matrix inverter and the associated input and output data distributions [13]. Figure 15 shows the conversion of the output matrix U from the *LU*-decomposition systolic array (Fig. 13) into the inputs of the matrix-inverter array (Fig. 14). Figures 15b and 15f show the data-distribution vectors of the inputs and outputs of the converter, respectively. Although the input distribution in Fig. 15b is not one of the 48 standard data distributions in Fig. 6a, it can be converted by n multiplexers into the data distribution in Fig. 15c. These multiplexers are an implementation of the transformation process T_1 in Eq. (16) with $\alpha_1 = \frac{1}{3}$ and $\beta_1 = \frac{2}{3}$. The data distribution in Fig. 15c is then converted into that of Fig. 15d by entering the data into the $n \times n$ mesh of buffers until they are filled. This is an implementation of transformation T_2 in Eq. (18) with $\alpha_2 = \frac{1}{2}$ and $\beta_2 = -\frac{1}{2}$. The data are output from the north side of the array, which is equivalent to a rotation of 270°. The resulting distribution is shown in Fig. 15e. Finally, n demultiplexers are used to convert the data distribution in Fig. 15e to that of Fig. 15f, which is transformation T_6 in Eq. (28) with $\alpha_6 = \beta_6 = 1$. The conversion for the triangular array L can be done similarly.

Figure 16 shows the conversion from the outputs of the matrix inverter


a
Scatter Matrix Generator
γ_i^s
m_s
Vector subtractor
Z_{s1}
Z_{s1}^T
Matrix multiplier
S_{s1}
Matrix adder
A
Z_{s1}
Z_{s1}^T
m
Matrix multiplier
S_{s2}
$D_iB_i^{-1}$
D_i
B_i^{-1}
B
Matrix Inverter
LU Decom- position
L
U
Matrix inverter
Matrix inverter
L^{-1}
U^{-1}
Matrix multiplier
A^{-1}
B^{-1}
Transform vector generator
d_i
Matrix vector multiplier
feature vector
x
Feature Generator
b
Covariance Matrix Generator
γ_i^s
m_s
Vector subtractor
W_{s1}
W_{s1}^T
Matrix multiplier
G_{s1}
Matrix adder
$G_{s1}+G_{s2}$
W_{s2}
W_{s2}^T
m
Matrix multiplier
G_{s2}
Linear System Solver
LU decom- mposition
L^{-1}
U^{-1}
Triangular matrix solver
triangular matrix solver
α
Feature vector
Fisher linear classifier
v
Class label

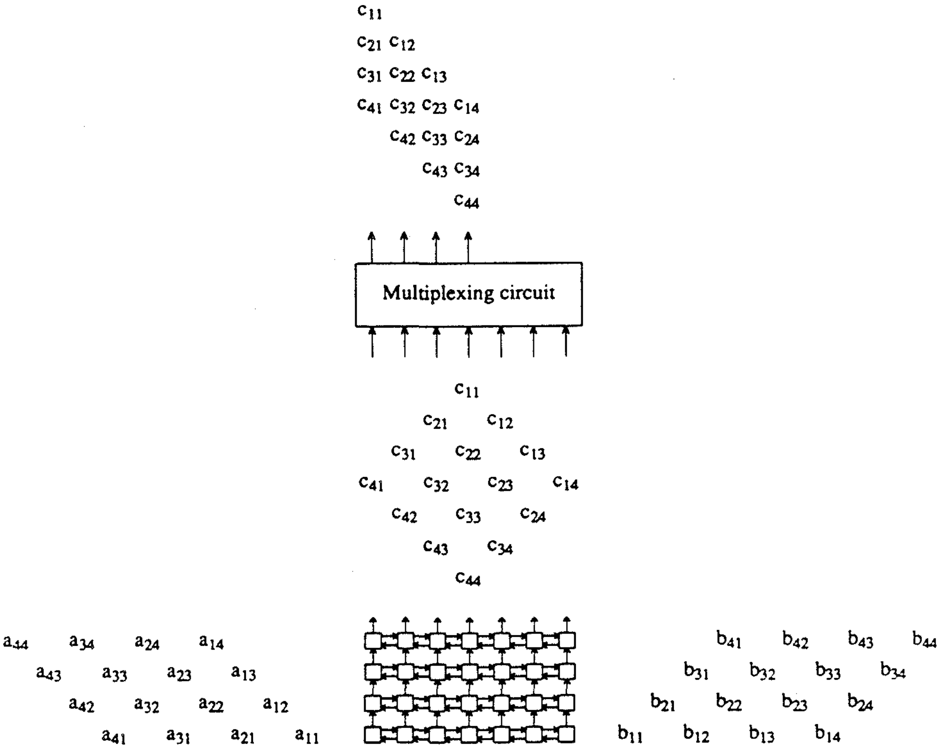


FIG. 12. Matrix-multiplication systolic array and the corresponding input and output data distributions.

(Fig. 14) to the inputs of the matrix-multiplication systolic array (Fig. 12). Figure 16b shows the data distribution of the outputs of the matrix inverter, which can be transformed by n multiplexers into the data distribution in Fig. 16c. Only n of the $(2n - 1)$ inputs to the multiplexers are used since the input matrix is an upper triangular matrix. The data will fill the $n \times n$ mesh of buffers and will be output in a staggered fashion to obtain the resulting data distribution in Fig. 16d. This corresponds to transformation T_4 with $\alpha_4 = 2$ and $\beta_4 = 1$. Finally, the data distribution in Fig. 16d is converted into that of Fig. 16e by a reversal network. This corresponds to T_5 with $\alpha_5 = -1$.

For the pattern classifier, operations similar to those in the feature extractor are needed except that it is necessary to solve the system $\mathbf{L} \cdot \mathbf{U} \cdot \mathbf{v} = \mathbf{f}$ after the LU decomposition $\mathbf{A} = \mathbf{L} \cdot \mathbf{U}$. This is done by first solving $\mathbf{L} \cdot \mathbf{E} = \mathbf{f}$ and then

FIG. 11. Applications of macropipelining in image processing and pattern recognition. (a) Feature extractor. (b) Pattern classifier.

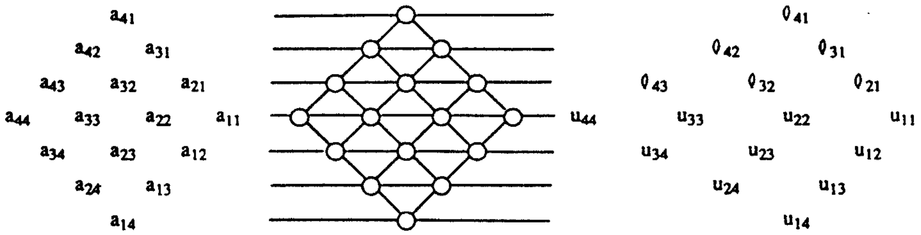


FIG. 13. *LU*-decomposition systolic array [10].

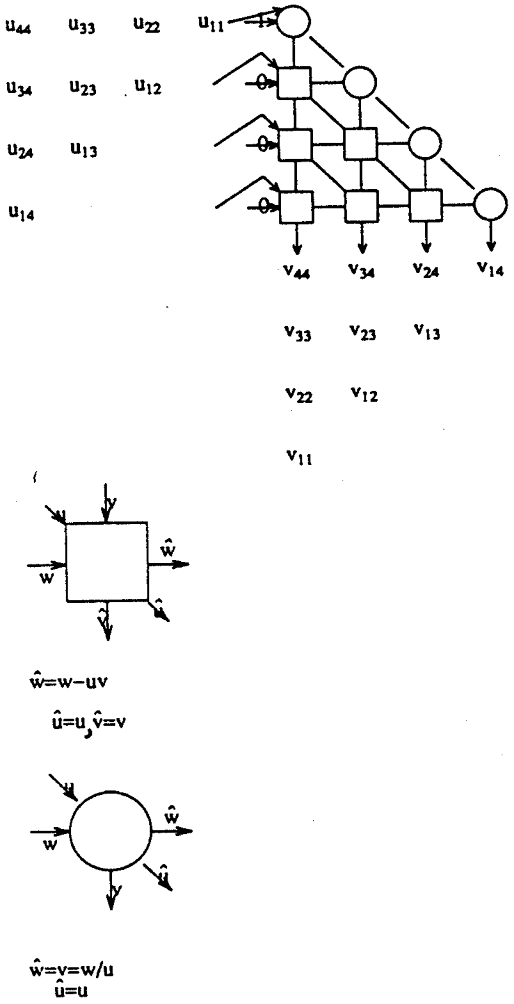


FIG. 14. Triangular-matrix inverter.

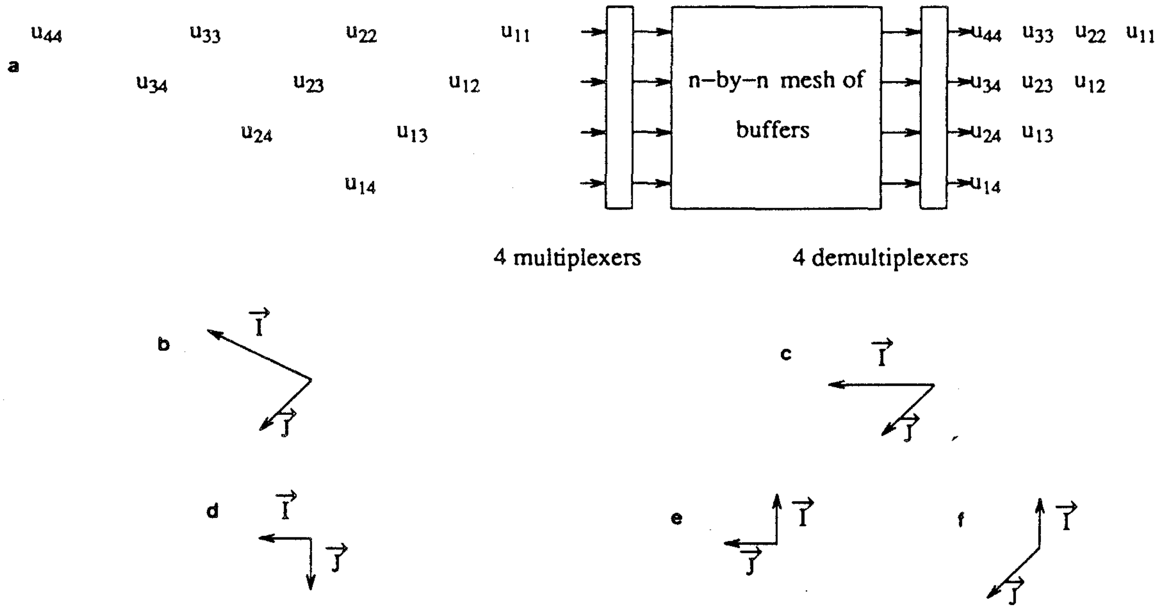


FIG. 15. Data conversion from the outputs of the *LU*-decomposition systolic array to the inputs of the triangular-matrix-inverter array. (a) The converter. (b) Input data distribution before entering the converter. (c) Input data distribution to buffers after multiplexing. (d) Data distribution after the buffers are filled. (e) Output data distribution from the buffers. (f) Output data distribution after demultiplexing.

U · **v** = **E** to get the solution vector **v**. Figure 17 shows a special-purpose converter to transform the output matrix *L* of the *LU*-decomposition array (Fig. 13) into the inputs required by the linear-system solver [10]. The su-

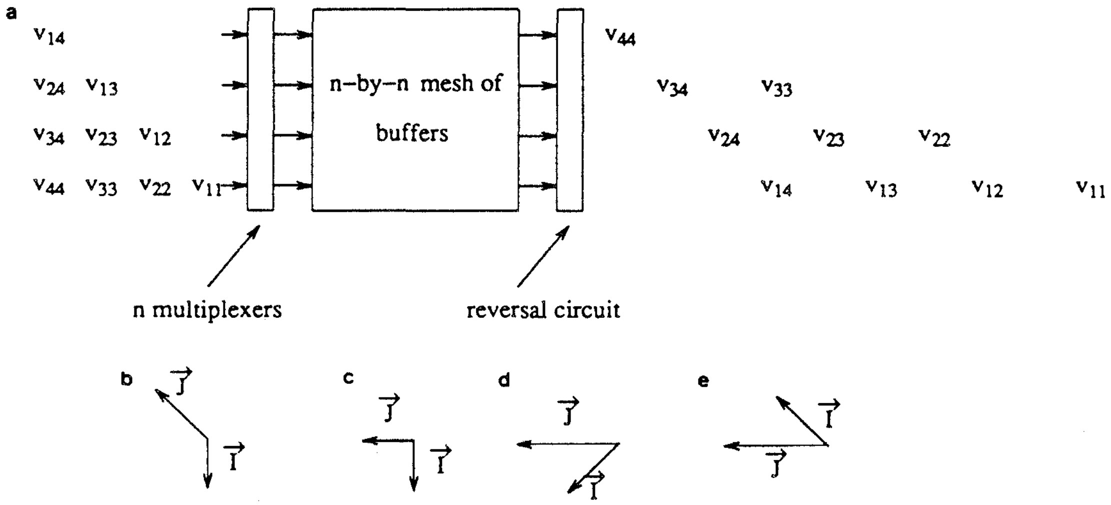


FIG. 16. Conversion from the outputs of the matrix inverter to the inputs of the matrix-multiplication systolic array. (a) The converter. (b) Input data distribution before multiplexing. (b) Input data distribution after multiplexing. (d) Output data distribution from the buffers. (e) Output data distribution after reversal.

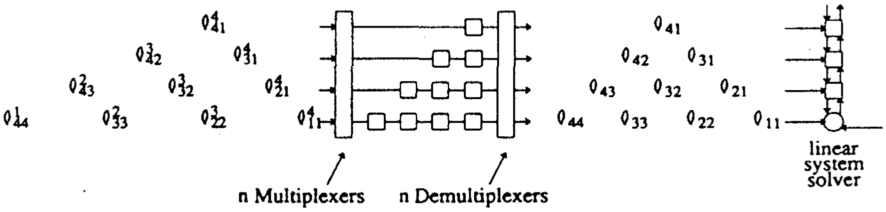


FIG. 17. Conversion of data in array L from the LU-decomposition array into the required input data format of the linear-system solver.

perscript in an input element indicates the number of time units that this data item will stay in the buffers. n multiplexers will route a data item to the appropriate row in the buffers, which cause the necessary delay, and n demultiplexers will convert the data to the required format of the linear-system solver. The conversion of array U into the required format is done similarly.

8. CONCLUSIONS

Macropipelines of systolic arrays can be used in a wide range of applications, especially in signal and image processing. To synchronize the dataflows in a macropipeline and to avoid the bottleneck of a common memory, converters are necessary to convert the output data from one systolic array into the required input-data format of the next systolic array in the pipeline. In this paper, an efficient algorithm is developed to find the minimum number of buffers for any conversion. A methodology to design a general-purpose converter is developed for the frequently used conversions. Methods to design special-purpose converters are also examined. The proposed design methods are exemplified on the design of converters for macropipelines in feature extraction and pattern classification.

REFERENCES

1. Agrawal, D. P., and Jain, R. Computer analysis of motion using a network of processors. *Proc. 5th Int'l Conf. on Pattern Recognition*, Dec. 1980.
2. Armstrong, C. V., *et al.* An adaptive multimicroprocessor array computing structure for radar signal processing applications. *Proc. 6th Annual Symposium on Computer Architecture*. IEEE/ ACM, 1979.
3. Briggs, F. A., Fu, K. S., Hwang, K., and Wah, B. W. PUMPS architecture for pattern analysis and image database management. *IEEE Trans. Comput.* C-31, 10 (Oct. 1982), 969–983.
4. Farrel, E. J. Processing limitations of ultrasonic image reconstruction. *Proc. Conf. on Pattern Recognition and Image Processing*, June 1978.
5. Fisher, A. L., Kung, H. T., and Sarocky, K. Experience with the CMU programmable systolic chip. *Proc. SPIE Sympos. Real-Time Signal Process.* 495 (1984), 120–129.

6. Fu, K. S., Hwang, K., and Wah, B. W. VLSI architecture for pattern analysis and image database management. In Kung, S. Y., *et al.* (Eds.). *VLSI and Modern Signal Processing*. Prentice-Hall, Englewood Cliffs, NJ, 1985.
7. Handler, W. The concept of macropipelining with high availability. *Elektron. Rechenanhagan*, No. 15 (1973), 269–274.
8. Hwang, K., and Su, S. P. VLSI architectures for feature extraction and pattern classification. *Comput. Vision Graphics Image Process.* **24** (Nov. 1983).
9. Kandle, D. A. A systolic processor for signal processing applications. *Computer* **20**, 7 (July 1987).
10. Kung, H. T. Highly concurrent systems. In Mead, C. A., and Conway, L. A. (Eds.). *Introduction to VLSI Systems*. Addison-Wesley, Reading, MA, 1980.
11. Kung, H. T. Why systolic architecture? *Computer* **15**, 1 (Jan. 1982), 37–46.
12. Kung, H. T., and Menzilcioglu, O. Warp: A programmable systolic array processor. *Proc. SPIE Sympos. Real-Time Signal Process.* **495** (1984), 130–136.
13. Li, G. J., and Wah, B. W. The design of optimal systolic arrays. *IEEE Trans. Comput.* **C-34**, 10 (Jan. 1985), 66–77.
14. Lin, F. C., Shen, J. W., Kuekes, P. J., Inn, Y. J., and McNamara, M. T. MOSAIC: A heterogeneous architecture for signal processors. *Proc. 12th DARPA Strategic Systems Symposium*, Monterey, CA, Oct. 1986.
15. Nicolac, G., and Hohne, K. H. Multiprocessor system for the real-time digital processing of video-image series. *Elektron. Rechenanhagan*, No. 21 (1979).
16. Nudd, G. R. Image understanding architectures. *Proc. National Computer Conference*. AFIPS Press, Reston, VA, 1980, pp. 377–390.
17. Nudd, G. R., Nash, J. G., Narayan, S. S., and Jain, A. K. An efficient VLSI structure for two-dimensional data processing. *Proc. Int'l. Conference on Computer Design*. IEEE, 1983, pp. 553–556.
18. Schreiber, R., and Kuekes, P. J. Application of systolic array technology to recursive filtering. In Kung, S. Y., *et al.* (Eds.). *VLSI and Modern Signal Processing*, Prentice-Hall, Englewood Cliffs, NJ, 1985.
19. Wah, B. W., Shang, W. J., and Aboelaze, M. Buffering in macropipelines of systolic arrays. *Proc. Workshop on Computer Architecture for Pattern Analysis and Image Database Management*, Nov. 1985, pp. 1–8.