# BUFFERING IN MACROPIPELINES OF SYSTOLIC ARRAYS

*Benjamin W. Wah, Weijia Shang, and Mokhtar Aboelaze*

**ABSTRACT**

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 computational complexity $\Theta(n^2)$, 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.

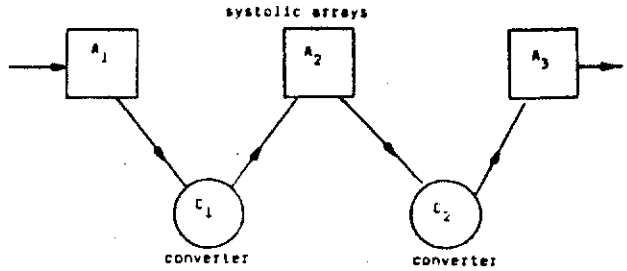
**Index Terms**: Buffer, conversion, data distribution, dynamic programming, macropipelining, systolic array.

## 1. INTRODUCTION

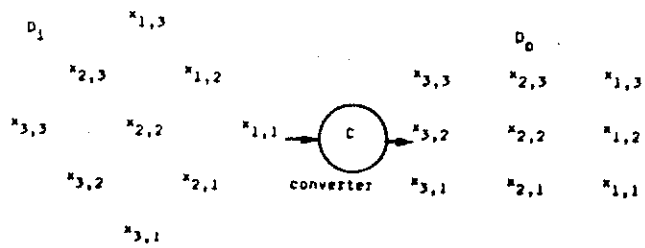
The evolution in Very-Large-Scale-Integration (VLSI) technology has had a great impact on computer architecture. Many existing algorithms in pattern recognition and image and signal processing can be implemented in a VLSI chip using multiple, regularly-connected processing elements (PEs) to exploit the great potential of pipelining and multiprocessing with applications in command, control, and communication systems. This type of array processors have been referred to as *systolic arrays* [10]. One of the many advantages of this approach is that each input item can be used a number of times once it is accessed, and thus a high computational throughput can be achieved with a modest I/O bandwidth. Other advantages include modular expandability, extensive concurrency, simple and regular data and control flows, and simplicity and uniformity of the processing cells.

In a large system, 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 [1,2]. Examples include real-time vision system [3], analysis of motion [4], image reconstruction from projections [5], radar signal processing [6], and air traffic control [7].

A *data distribution* for a systolic array is 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 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 distribution. A conventional approach is to use a common memory to buffer the outputs of the systolic arrays. However, this 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. This may not be always possible, especially when the macropipeline is reconfigurable. A third approach is to design, between two stages of the macropipeline, a converter that consists of multiple buffers and a control unit to select the appropriate buffers for inputs and outputs [8]. The concept is exemplified in Figure 1a. $C_1$ and $C_2$ are converters to convert the output data into the required input formats. Figure 1b illustrates this conversion. To convert data from distribution $D_i$ to $D_o$, 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 buffer the




(a) A macropipeline of systolic arrays.




(b) Conversion of data from one distribution to another.

Figure 1. Concept of macropipelining of systolic arrays.

This research was supported partly by CIDMAC, a research unit of Purdue University, sponsored by Purdue, Cincinnati Millicron Corporation, Control Data Corporation, Cummins Engine Company, Ransburg Corporation, and TRW, and partly by the National Science Foundation Grant DMC-85 19649.

B. W. Wah and M. Aboelaze are with the Department of Electrical and Computer Engineering and the Coordinated Science Laboratory, University of Illinois at Urbana-Champaign, Urbana, IL 61801. W. J. Shang is with the School of Electrical Engineering, Purdue University West Lafayette, IN 47907.

IEEE Computer Society Workshop on Computer Architecture for Pattern Analysis and Image Database Management, November 1985.

data in the first column of $D_o$ before they can be output. five buffers are needed for the second column. and three buffers are needed for the last.

In this paper. we will study the use of converters to interface the systolic arrays in a macropipeline. The design of converters depends of the type of macropipelines. 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. In this case, special-purpose converters are needed. In contrast, in a *dynamic macropipeline*, a (sub)set of systolic arrays are 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.

The objective of this study 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. The pipeline can be either synchronous in which the interarrival times of inputs are equal, or asynchronous in which the interarrival times may be different. In the remaining sections, we will study the minimum number of buffers for a given conversion, propose design procedures for general-purpose and special-purpose converters, and investigate tradeoffs in designing macropipelines.

## 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 sent out. In this section, an algorithm will be 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 [9]. 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 $\bar{I}$. Similarly, the column vector of X, denoted by $\bar{J}$, is defined as the directional distance between $x_{i,j}$ and $x_{i,j+1}$. A data distribution with vectors $\bar{I}$ and $\bar{J}$ is denoted by $D(\bar{I}, \bar{J})$. Two data distributions are illustrated in Figure 2.

The geometric layout of a distribution can be described in the Cartesian plane. Without loss of generality, $x_{1,1}$ for both the input and output distributions is assumed to be placed at the origin, and data is moving in the direction of the negative x-axis. Vectors $\bar{I}$ and $\bar{J}$ determine the locations of the 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 $\bar{I}$ and $\bar{J}$ on the x-axis. Likewise, $I_y$ and $J_y$ are the projections on the y-axis. Then,

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

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

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, 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 *i'th* (*input* or *output*) *step* is defined as the set of elements in the (input or output) distribution with the x-coordinate equal to i.

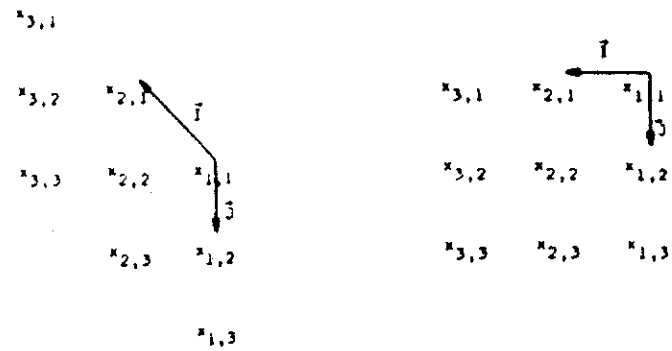



Figure 2. Two data distributions and their corresponding vectors. (The first data distribution has three streams of dataflow. The second one has five streams.)

### 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 before the i'th step of $D_o$ can be output, assuming that the (i-1)'th step has been output, and $B_i$ be the maximum number of buffers needed when the i'th step of $D_o$ is output. For the example in Figure 1, to output $x_{1,1}$, $x_{1,2}$, and $x_{1,3}$, $b_1 = 6$ buffers are needed to buffer the first three columns of $D_i$. Similarly, $b_2 = 5$, and $b_3 = 3$. Hence,

$$B_{min} = \max\{b_1, b_2, b_3\} = b_1 \tag{3}$$

$$B_i = \max\{b_i, B_{i-1}\} \tag{4}$$

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

An *input partition* partitions the input array X into $N_i$ disjoint subsets, $I_p$, $1 \leq p \leq N_i$, where

$$I_p = \{\, x_{i,j} \mid C_x(i,j) = (i-1)\,I_x^i + (j-1)\,J_x^i = a_p \} \tag{5}$$

and $N_i$ is the number of input steps. $\bar{I}^i$ and $\bar{J}^i$ are the row and column vectors of the input distribution, with $I_x^i$ and $J_x^i$ as the corresponding projections on the x-axis. $I_p$ represents the set of input elements with the same x-coordinate $a_p$. $I_1$ is the set of input data that arrive at the converter first, and $I_p$ is the p'th arrival set.

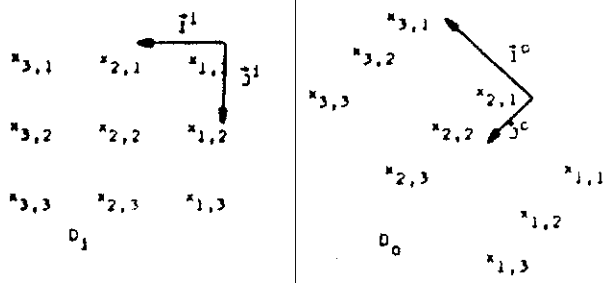
An *output partition* partitions the output array X into $N_o$ disjoint subsets $O_k$, $1 \leq k \leq N_o$, where

$$O_k = \{\, x_{i,j} \mid C_x(i,j) = (i-1)\,I_x^o + (j-1)\,J_x^o = a_k \} \tag{6}$$

and $N_o$ is the number of output steps for the output distribution. $\bar{I}^o$ and $\bar{J}^o$ are the vectors of the output distribution, with $I_x^o$ and $J_x^o$ as the corresponding projections on the x-axis. $O_k$ represents output elements with the same x-coordinate $a_k$. $O_1$ represents the set of data that departs from the converter first, and $O_k$ is the k'th departure set.

Figures 3a and 3b illustrates $D_i$ and $D_o$ of a 3-by-3 array X. Since X arrives in 3 steps, there are three input partitions ($N_i = 3$). Similarly, there are seven output partitions ($N_o = 7$) because the outputs depart in 7 steps. The partitions are indicated in Figure 3c. Let S and O be the sets of input and output partitions and $\Pi$ be their union.

$$S = \{I_p \mid 1 \leq p \leq N_i\} \tag{7}$$




(a) Input distribution $D_i$ and output distribution $D_o$.

$S = \{I_1, I_2, I_3\}$ $\quad O = \{O_1, \ldots, O_7\}$

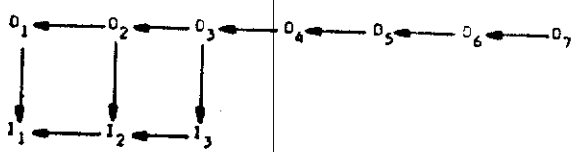
$I_1 = \{x_{1,1}, x_{1,2}, x_{1,3}\}$, $\quad O_1 = \{x_{1,1}\}$, $O_2 = \{x_{1,2}\}$,

$I_2 = \{x_{2,1}, x_{2,2}, x_{2,3}\}$, $\quad O_3 = \{x_{2,1}, x_{1,3}\}$, $O_4 = \{x_{2,2}\}$,

$I_3 = \{x_{3,1}, x_{3,2}, x_{3,3}\}$ $\quad O_5 = \{x_{3,1}, x_{2,3}\}$,

$O_6 = \{x_{3,2}\}$, $O_7 = \{x_{3,3}\}$

(b) Input and output partitions.




(c) Lattice for the partial ordering.

Figure 3. Partitions and partial ordering.

$$O = \{O_k \mid 1 \leq k \leq N_o\} \tag{8}$$

$$\Pi = S \bigcup O \tag{9}$$

For $\pi_i \in \Pi$, $|\pi_i|$ represents the number of elements in $\pi_i$.

The example in Figure 3 shows that there exists a relationship between the $O_k$s and $I_p$s. A partial ordering $^{\circ}\!\!\rightarrow\!\!^{\circ}$ can be defined on $\Pi$ as follows. If $I_k \rightarrow I_p$, then data in $I_k$ will arrive earlier than that of $I_p$. If $O_k \rightarrow O_p$, then data in $O_k$ will leave earlier than that of $O_p$. Further, if $O_k \rightarrow I_p$, then data in $I_p$ must arrive before data in $O_k$ can depart. To output the elements in $O_p$, all the elements in $I_q$s such that $O_k \bigcap I_q \neq \emptyset$ must have arrived at the converter. In summary,

(1) $I_k \rightarrow I_p$ if $k > p$;

(2) $O_k \rightarrow O_p$ if $k > p$;

(3) $O_k \rightarrow I_p$ if either $O_k \bigcap I_p \neq \emptyset$ or there exists an integer $q$ such that $I_q \bigcap O_k \neq \emptyset$ and that $I_q \rightarrow I_p$.

The above definitions imply that if $O_k \rightarrow I_p$, then $O_k \rightarrow I_{p-1} \rightarrow I_{p-2} \rightarrow \cdots \rightarrow I_1$. The integer $q_k$ such that $O_k \rightarrow I_{q_k}$ and that $O_k \not\rightarrow I_{q_k+1}$ is defined as the *key number* for $O_k$ since all the relationships among the $I_p$s and $O_k$ will be known once $q_k$ is found. The partial ordering of the partitions can be represented in a lattice. Figure 3d shows the lattice of the partial ordering for the example in Figure 3a. For instance, $q_3 = 2$ is the key number for $O_3$ since $O_3 \bigcap I_2 = \{x_{2,1}\}$ and $O_3$ can depart once elements in $I_2$ have arrived.

```
procedure compute_partial_ordering;
/* Inputs:  J^i and J^o: vectors for input distribution;
            J^o and J^o: vectors for output distribution;
            N_i: number of input steps;
            N_o: number of output steps;
  Outputs:  Two arrays Φ(2,N_o), S(N_i), where
            Φ(1,k) = |O_k|;
            Φ(2,k) = q_k, the key number for O_k;
            S(p) = |I_p| */
(1) Initialize Φ and S to zeroes.
(2) for i=1 to n do [
        for j=1 to n do [
            k := C_x^o(i,j) + 1;
            p := C_x^i(i,j) + 1;
            S(p) := S(p) + 1;
            Φ(1,k) := Φ(1,k) + 1;
            Φ(2,k) := max {Φ(2,k), p}
        ]
    ]
```

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

To use dynamic programming to find $B_{min}$, $O_1, O_2, \ldots, O_k$ are examined sequentially. If $q_k$ is the key number for $O_k$, then $I_1, I_2, \ldots, I_{q_k}$ must have arrived at the converter before elements in $O_k$ can depart. The reason is that either $I_p$, $1 \leq p \leq q_k$, contains data that are in $O_k$, or $I_p$ does not contain data in $O_k$ but $I_{q_k} \rightarrow I_p$. Therefore, elements $x_{i,j} \in I_1 \bigcup I_2 \bigcup \cdots \bigcup I_{q_k}$ that remain when $I_{q_k}$ arrives and $O_{k-1}$ has left must be buffered. In other words, $b_k$, the number of buffers needed for $O_k$, is

$$b_k = \sum_{i=1}^{q_k} |I_i| - \sum_{i=1}^{k-1} |O_i| \tag{10}$$

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.

$$B_0 = 0; \tag{11a}$$

$$B_k = \max\left\{B_{k-1}, B_{k-1} - |O_{k-1}| + \sum_{j=q_{k-1}+1}^{q_k} |I_j|\right\} \tag{11b}$$

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 $O_k$. $C_x^i(i,j)$ 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 $O_k$ and $I_p$, then $O_k \rightarrow I_p$, and the counters of $O_k$ and $I_p$ are incremented. $q_k$ is updated to $p$ if $p$ is larger the the original value of $q_k$. The algorithm to compute the partial ordering is shown in Figure 4. The computational complexity of the algorithm is $O(n^2)$. A better algorithm with a computational complexity of $O(\max\{N_i, N_o\} \times \min\{I_x^o, J_x^o\})$ can be devised, but will not be presented here.

The example in Figure 3 is used to illustrate the algorithm. Initially, all the key numbers are initialized to zeroes. $N_i = 3$, and $N_o = 7$. Since $C_x^o(1,1) = 0$, $C_x^i(1,1) = 0$, hence $x_{1,1} \in O_1$.

$x_{1,1} \in I_1$. $q_1$ and the counters for $O_1$ and $I_1$ are updated to ones. Similarly, it is found that $x_{1,3} \in O_3$ and $x_{1,3} \in I_1$. The counters for $O_3$ and $I_1$ are incremented, and $q_3$ is set to one. For $x_{2,1}$, it is found that $x_{2,1} \in I_2$ and that $x_{2,1} \in O_3$. The counters for $O_3$ and $I_2$ are incremented, and $q_3$ is set to $\max\{q_3, 2\} = 2$. Likewise, the remaining elements in $\bar{X}$ can be examined.

## 3. COMBINATIONS OF DATA DISTRIBUTIONS

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

$$I_x^1 = I_x^2 \text{ and } J_x^1 = J_x^2 \tag{12a}$$

$$I_y^1 \cdot I_y^2 \geq 0 \text{ and } J_y^1 \cdot J_y^2 \geq 0 \tag{12b}$$

where $I_x^1$ (resp. $J_x^1$) is the projection of $\bar{I}^1$ (resp. $\bar{J}^1$) of $D_1$ on the x-axis.

The first condition (Eq. (12a)) 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. (12b)) ensures that the data arriving at the systolic array at the same time have the same permutations. However, the number of streams of dataflow into the systolic array for the two data distributions may not be equal and can range from n to 2n−1. As an example, the two data distributions in Figure 2 are equivalent, but have different number of streams of dataflow.

The following theorem shows the number of possible equivalent partitions of data distributions.

**Theorem:** There are $\Omega(2^n)$ equivalent partitions of data distributions for an n-by-n array of data.

*Proof:* In proving the number of equivalent partitions, only the projections of the vectors on the x-axis have to be considered. Without loss of generality, assume that $\bar{J}$ is not orthogonal to the x-axis. Consider the x-projections of the first row of data, $C_x(1,1), C_x(1,2), \ldots, C_x(1,n)$. The problem here is to determine the number of possible x-projections for the remaining rows. Consider the x-projection of $x_{2,1}$. Assuming that $C_x(2,1) \geq C_x(1,1)$, 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, $C_x(3,1) = C_x(1,2)$, $C_x(1,1) < C_x(3,1) < C_x(1,2)$, and $C_x(1,2) < C_x(3,1) < C_x(1,3)$ (see Figure 5). When $C_x(3,1) = C_x(1,2)$, the positions of the remaining elements are determined. However, when either $C_x(1,1) < C_x(3,1) < C_x(1,2)$ or $C_x(1,2) < C_x(3,1) < C_x(1,3)$, then $C_x(4,1)$ can fall in three possible ranges, as shown on the second level of the tree in Figure 5. The same argument can be applied to the remaining levels of the tree for $x_{3,1}, \ldots, x_{n,1}$. In level $\ell$, $1 < \ell < n-3$, there are $\ell$ terminals, while in level n−2, there are $3\times2^{n-3}$ terminals. Therefore, the total number of terminals is

$$\sum_{\ell=1}^{n-3} \ell + 3\times2^{n-3} = \Omega(2^n) \tag{13}$$

A similar argument can be made when $C_x(1,1) > C_x(2,1)$ or $C_x(2,1) > C_x(1,2)$. Hence the number of possible distributions is $\Omega(2^n)$. □

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

If vectors $\bar{I}$ and $\bar{J}$ are restricted to have only unitary or zero projections on the x- and y-axis, then there will be eight possible directions for $\bar{I}$ pointing at 0°, 45°, 90°, 135°, 180°, 225°, 270°, and 315°. For each direction of $\bar{I}$, there are six possible directions for $\bar{J}$, excluding the cases in which $\bar{I}$ and $\bar{J}$ are pointing in the same or opposite directions. Thus there are 8×6=48 possible combinations of data distributions (Figure 6a). Out of these 48 cases, there are only sixteen equivalent classes (distributions in the same column of Figure 6a belong to the same equivalent class). The set of equivalent partitions of data distributions can be further reduced if a reversal circuit is available to reverse the order of data arriving simultaneously at the converter. In this case, distribution (i'), $1 \leq i \leq 8$, can be mapped into distribution (i) (Figure 6b).

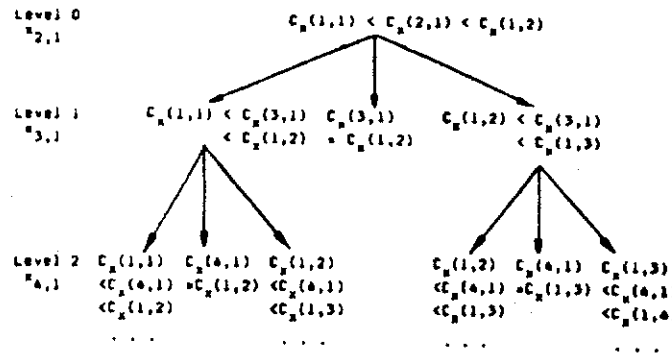


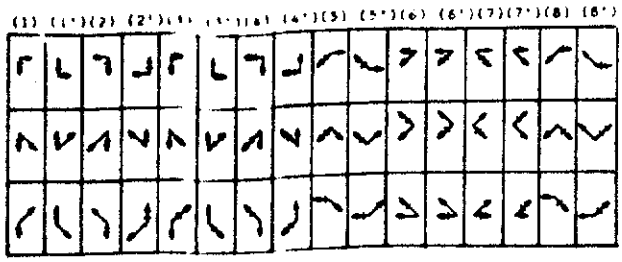

Figure 5. Possible positions for $x_{2,1}, x_{3,1}, \ldots, x_{n,1}$.

For a given output distribution, there can be eight possible input data distributions. This number can be further reduced by renaming the data. Referring to Figure 6b, suppose that input distribution (B) has to be converted to output distribution (E). By renaming data along $\bar{J}$ such that input distribution (A) is used, then data of output distribution (6') (Figure 6a) would be identical to that of output distribution (E). The number of input distributions can thus be reduced to two (Figure 6c).

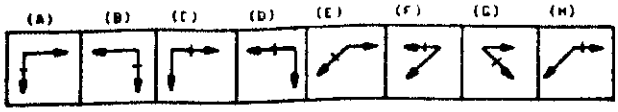
## 4. GENERAL-PURPOSE CONVERTERS

In this section, we propose 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-axis. It is assumed that data always enter the converter in n streams, that is, the distributions in the first row of Figure 6a are used. If the output data from the previous stage of the macropipeline require more than n streams, then they are first multiplexed into n streams in the previous stage before they are output. The multiplexing converts data from the data distributions in the second and third rows of Figure 6a into that in the first row. It also minimizes the number of connections between the systolic array and the converter.

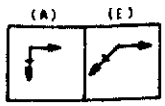
The n-by-n array of data are routed into an n-by-n mesh of buffers with four-neighbor connections until they are filled (Figure 7a). The interconnections in the buffers allow data to be shifted in one of the four directions. Further, the shifts can be controlled by different clocks to allow staggering of data from different rows or columns. Data are input in one direction and may come out from any one of the four directions. One of the eight output distributions in Figure 6b can be obtained by selecting data coming out from one of these four directions using multiplexors and proper timing. The staggering of data maps the data from one of the four output distributions in Figure 6b ((A) thru (D)) to one of the other four ((E) thru (H)). For the buffers in Figure 7a, the first and third elements may come from cells (1,1), (1,3), (3,1), or (3,3), while the second element may come from cells (1,2), (2,1), (2,3), or (3,2). Data may then be



(a) All possible combinations of data distributions.




(b) Eight standard output distributions.




(c) Two standard input distributions.

Figure 6. Possible data distributions when $\vec{I}$ and $\vec{J}$ have either zero or unitary projections on the x- and y-axis. (The arrow with a bar across it is $\vec{I}$; the other is $\vec{J}$.)

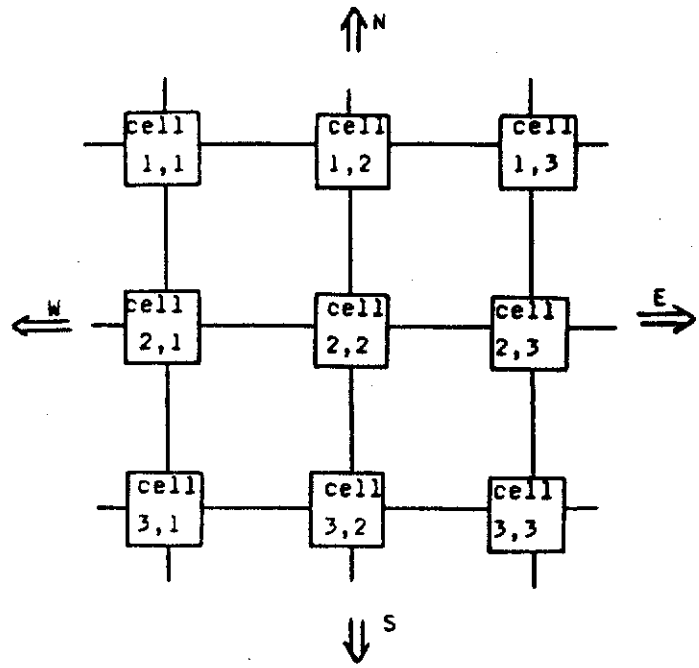
routed through a reversal network to obtain the proper permutation (Figure 7b). The reversal network essentially maps data with output distribution (i) into that of (i') in Figure 6a.

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 multiplexors to input data into the buffers rather than from the perimeter. n demultiplexers, $d_1$, $d_2$, ..., $d_n$, are added to the n rows of buffers in Figure 7a (Figure 7c). $d_1$ and $d_n$ are two-way demultiplexers, while the rest are four-way demultiplexers. For buffers in row i, 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 (E) thru (H) in Figure 6b. For example, to convert from input distribution (A) to output distribution (E), demultiplexor $d_i$ 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 i'th 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.

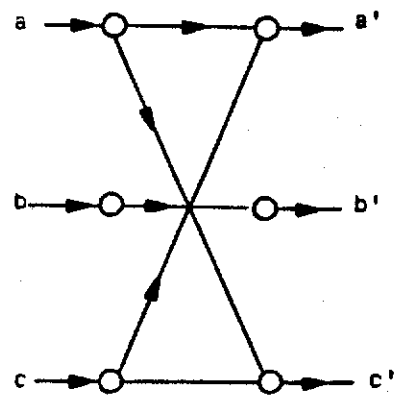
## 5. SPECIAL-PURPOSE CONVERTERS

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

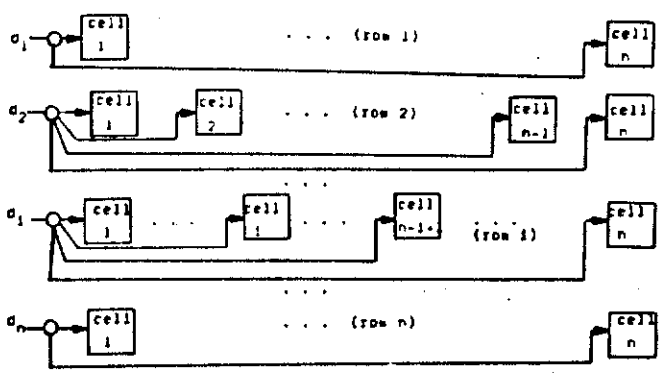
The conversion between any pair of the eight standard distributions in Figure 6b is straightforward and is illustrated in the following examples. To convert from distribution (A) to distributions (B), (C), or (D), $n^2$ buffers are needed. The input data are propagated from left to right and are output in the western, southern, or northern directions after the buffers are filled. To convert from distribution (A) to (E), n(n−1)/2 buffers




(a) 3-by-3 mesh of buffers.

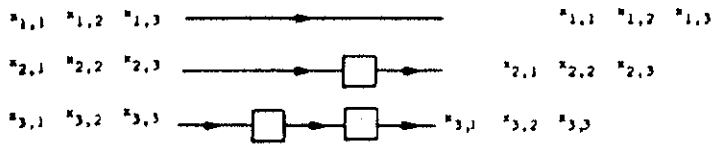



(b) Reversal network.

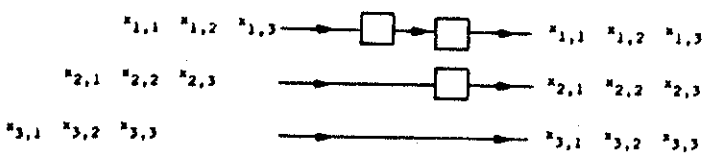



(c) Organization of the buffers for the modified general-purpose converter.

Figure 7. Architecture of the general-purpose converter (n=3).



(a) Conversion from data distribution (A) to (E) (n=3).



(b) Conversion from data distribution ((E) to (B) (n=3).

Figure 8. Special-purpose converters.

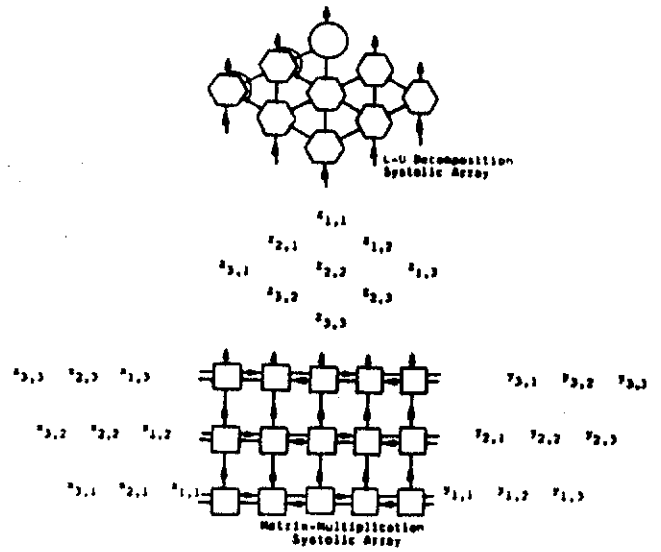



Figure 9. A macropipeline to multiply matrices A and B and to decompose the product matrix into submatrices L and U. (n=3, no intermediate converter is used)

are arranged as shown in Figure 8a. The conversion from distribution (A) to (F) will need $n^2$ buffers. Data are output from the north after the buffers are filled. Data in column i are output one step ahead of that of column i+1. The conversion from (A) to (G) is similar that from (A) to (E), and that from (A) to (H) is similar that from (A) to (F). The conversion from distribution (E) to (A) requires n(n−1)/2 buffers (Figure 8b). 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 Figure 1b. The conversion from (E) to (F) requires $n^2$ buffers, and data in column i are output one step ahead of data in column 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 Figure 6 may be complicated, and a heuristic procedure is proposed here. First, the minimum number of buffers, $B_{min}$ is found by the algorithm discussed in Section 2. A feasible control circuit with $B_{min}$ buffers is then searched. The control circuit contains multiplexors and demultiplexors that can be individually controlled by stored microprograms. If a feasible solution cannot be found easily or if the control circuit is too complex, more buffers are added, and the procedure is repeated.

## 6. DESIGN OF AN OPTIMAL MACROPIPELINE

In designing a macropipeline of systolic arrays, the independent optimization of individual systolic arrays [9] and the intermediate converters may not result in an optimal macropipeline. In this section, we will formulate the design of an optimal macropipeline as an optimization problem.

The objective function for the design will be the $AT^2$ measure, where A is the total area of the chips and T is the total time taken to solve the problem. The area of a systolic array is proportional to the number of processing elements in the array [9], and the area of the converter is proportional to the number of buffers. The constants of proportionality depend on the technology used, and the relative complexity between the processing elements of the systolic array and the buffers. The design of a $\ell$-stage macropipeline with $\ell$ systolic arrays and $\ell-1$ intermediate converters is

$$\text{minimize} \quad AT^2 \quad \text{where} \qquad (14)$$

$$A = \sum_{i=1}^{\ell} k_i A_i + \sum_{i=1}^{\ell-1} k_i' A_i'$$

$$T = \sum_{i=1}^{\ell} T_i + \sum_{i=1}^{\ell-1} T_i'$$

where $A_i$ (resp. $A_i'$) is the area of the i'th systolic array (resp. converter), $k_i$ (resp. $k_i'$) is the associated constants of proportionality, and $T_i$ (resp. $T_i'$) is the associated computation time. The complexity of the above optimization problem is $O(\ell^p)$, where p is the number of ways to design a single systolic array. We have found before that the worst-case complexity of designing an optimal systolic array is $O(n^3)$ [9], hence the optimization of a macropipeline is extremely difficult even for small pipelines. In this case, heuristic and suboptimal designs may have to be used.

In the remaining part of this section, an example is used to illustrate the design of a macropipeline. Suppose that two matrices X and Y are to be multiplied and that the product matrix is decomposed into submatrices L and U, where L is a lower triangular matrix and U is an upper triangular matrix. Figure 9 shows a possible solution without using a converter. Since the output of the multiplication array must be compatible with the input of the L-U decomposition array, the multiplication array is by no mean optimal with respect to the $AT^2$ measure. In this design,

$$A_1 = k_1 n(2n-1) \quad T_1 = 2n-1 \quad \text{(matrix mult.)}$$

$$A_2 = k_2 n^2 \quad T_2 = 4n \quad \text{(L-U decomp.)}$$

$$M_1 = (A_1+A_2)(T_1+T_2)^2 \propto (3n^2-n)(6n-1)^2$$

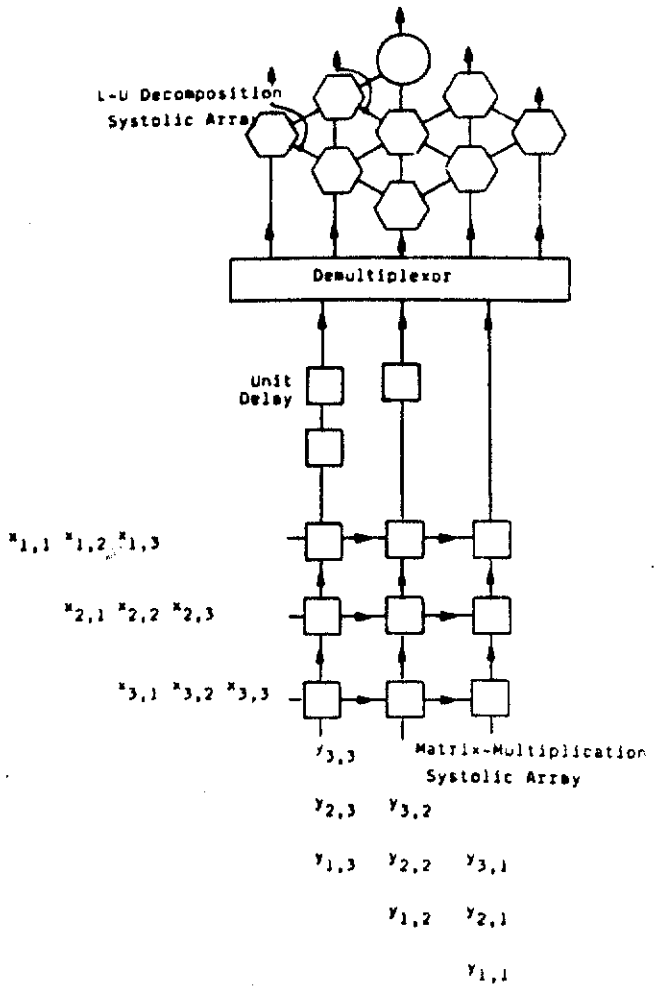



Figure 10. A macropipeline to multiply matrices A and B and to decompose the product matrix into submatrices L and U. (n=3, an intermediate converter is used)

Figure 10 shows the macropipeline in which the multiplication and L-U decomposition systolic arrays are optimal [9,11]. A converter is used to transform the output distribution of the multiplication array into the required input distribution of the L-U decomposition array. In this design,

$$A_3 = k_1 n^2 \qquad T_3 = 2n+1 \quad \text{(matrix mult.)}$$

$$A_4 = k_2 n^2 \qquad T_4 = 4n \quad \text{(L-U decomp.)}$$

$$A_1' = k_1' \frac{n(n+1)}{2} \qquad T_1' = 1 \quad \text{(buffers)}$$

$$M_2 = (A_3+A_4+A_1')(T_3+T_4+T_1')^2 \propto \left[2n^2+\frac{n(n+1)}{2}\right](6n+2)^2$$

Figure 11 shows a plot of the $M_1/M_2$ for different values of n and $k_1'/k_1$ assuming that $k_1 = k_2$. For those parts of the curves such that $M_1/M_2 > 1$, the use of intermediate converters is beneficial.

## 8. CONCLUSIONS

Macropipelines of systolic arrays have found a wide range of applications. To synchronize the data flow in the pipe, a converter is necessary. In this paper, an algorithm with a complexity $O(n^2)$ is presented to find the $B_{min}$ needed for any conversions. The data distributions are classified into 2×8 basic conversions. A structure for the general purpose converter has been presented.

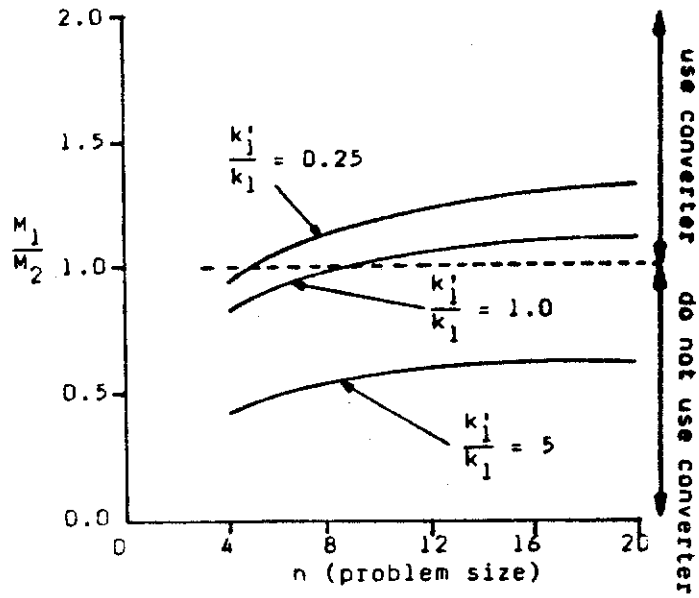



Figure 11. A plot showing the relationship between $M_1/M_2$ for different values of n. (It is assumed that $k_1 = k_2$).

Future work includes decomposition of systolic arrays to make the delay time for each stage of the pipe equal.

## REFERENCES


[1] G. R. Nudd, "Image Understanding Architectures," *Proc. National Computer Conference*, AFIPS Press, pp. 377-390, 1980.

[2] G. R. Nudd, J. G. Nash, S.S. Narayan, and A. K. Jain, "An Efficient VLSI Structure for Two-Dimensional Data Processing," *Proc. IEEE Int'l Conference On Computer Design*, pp. 553-556, 1983.

[3] G. Nicolae and K.H. Hohne, "Multiprocessor System For the Real-Time Digital Processing of Video-Image Series," *Elektronische Rechenanhagan*, No. 21, 1979.

[4] D. P. Agrawal and R. Jain, "Computer Analysis of Motion Using a Network of Processors," *Proc. 5th Int'l Conf. Pattern Recognition*, Dec. 1980.

[5] E. J. Farrel, "Processing Limitations of Ultrasonic Image Reconstruction," *Proc. Conf. on Pattern Recognition and Image Processing*, June 1978.

[6] C. V. Armstrong, et al., "An Adaptive Multimicroprocessor Array Computing Structure for Radar Signal Processing Applications," *Proc. 6th Annual Symposium on Computer Architecture*, 1979.

[7] W. Handler, "The concept of Macropipelining with High Availability," *Elektronische Rechenanhagan*, No. 15, pp. 269-274, 1973.

[8] F.A. Briggs, et al., "PUMPS Architecture for Pattern Analysis and Image Database Management," *IEEE Trans. on Computers*, Vol. C-31, No. 10, pp. 969-983, Oct. 1982.

[9] G. J. Li and B. W. Wah, "The Design of Optimal Systolic Arrays," *IEEE Trans. on Computers*, Vol. C-34, No. 1, Jan. 1985.

[10] H. T. Kung, "Why Systolic Architectures," *IEEE Computer*, pp. 37-46, Jan. 1982.

[11] C. Mead and L. Conway, *Introduction to VLSI Systems*, Addison-Wesley, 1980.