"Algorithms and Parallel VLSI Architectures II", ed P. Quinton & Y. Robert
pp. 205-210, Elsevier Science Pub., Bonas, France, June 3-6, 1991.

A PROGRAMMABLE VLSI ARRAY WITH CONSTANT I/O PINS

Mokhtar A. Aboelaze, Delei Lee,
Dept. of Computer Science
York University
4700 Keele Street
North York, Ontario M3P 1J3
CANADA

Benjamin W. Wah
Coordinated Science Laboratory
University of Illinois
1101 West Springfield Avenue
Urbana, IL 61801
U.S.A.

ABSTRACT

In this paper we present a method for designing systolic arrays with a constant number of I/O ports and a fixed bandwidth per port. This design is in the form of an L-by-N grid of PE's with $O(L)$ I/O ports, where L is a design parameter governed by the current technology. The boundary cases consist of a square array with $L=N$, and a linear array with $L=1$. A PE in this array consists of a simple ALU and memory. We illustrate the method by applying it on the design of systolic arrays for matrix multiplication and QR decomposition.

1. INTRODUCTION

Until recently, computation-intensive tasks in signal and image processing have been handled using expensive, high-performance, general-purpose computers. Although these systems are effective for certain cases, they are inadequate for real-time applications in which speed and throughput are crucial factors.

A novel architectural concept made possible by advances in VLSI technology is the systolic arrays. Systolic arrays make use of multiple regularly-connected processing elements (PE's) to exploit potential parallelism in the problem [Kun82, Kun80]. One can find in the literature many systolic arrays designed for different applications [KuL87, NaP87, McM87]. Most of these arrays are in the form of a square mesh. The major drawback of a square mesh is its $O(N)$ I/O ports, where N is the number of PE's. Since the number of I/O ports is limited by implementation factors, it imposes a limit on the number of PE's in the array.

Recently, there are active research on the design of linear systolic arrays [PrT88, PrT89, RaV84], that require fixed number of I/O ports. This design approach has shifted its emphasis from optimizing computations on the chip to optimizing I/O operations using the limited number of I/O ports.

In this paper, we investigate the design of a programmable systolic array with constant number of I/O ports and a fixed bandwidth per port. The array is in the form of an L-by-N rectangular mesh with $O(L)$ I/O ports, where $1 \leq L \leq N$. In one extreme, when $L = 1$, the array is a linear array. In the other extreme, when $L = N$, the array is a square mesh. Hence, L is a design parameter that reflects the ability of the current technology in accommodating I/O ports. Consequently, the design can grow with improving technology. Moreover, the ability to program it allows it to be used in more than one applications, and makes it easy to amortize its design cost.

2. THE ARRAY MODEL

The systolic array used in this paper is the *control flow systolic array* [Abo88]. The idea behind it is to use a more powerful PE than the one used in a traditional systolic array [Kun82] which performs simple arithmetic operations. Each PE is capable of performing a few simple instructions, which include read from local memory, write to local memory, perform a simple arithmetic operation, read data from input buffer, and write data to output buffer.

There are two ways to implement the control of the different PE's. The first is by having a microprogram in the form of an infinite loop stored in the local memory, and the second is by sending control signals together with the data. In the first approach, each PE receives data and control in each cycle, decodes the control bits, and performs some operations on the data. If the number of allowable operations is small, only a few control bits traveling with the data are sufficient. In this paper, we use the first method, *i.e.*, by storing a microprogram to be executed by each PE. A design of a 2-D digital filter using this method is presented elsewhere [AbL91].

A major advantage of using this approach is its versatility. By changing the stored microprogram, the array can be designed to perform more than one function [Abo88]. It is also composed of regularly connected PE's, hence, it is cost-effective and is suitable for VLSI implementation.

The architecture of the system consists of L-by-N PE's, each with a simple control, a microprogram memory, $O(N/L)$ words of local memory, and some I/O buffers. Note that N is a design parameter depending on the problem size; hence, an array designed for problem size N cannot be used to solve a problem of size $2N$ by concatenating two more such arrays.

3. DESIGN METHOD

The design method can be summarized in the following steps.

(a) Start with a 2-D systolic array for solving the given problem. We choose to start with a 2-D array because there are a large number of methods for designing 2-D arrays.

(b) Combine each of the N/L rows of the array to form one row. In this step we assign to the different PE's the operations it performs. This is straightforward as each PE is assigned the operations of $M = N/L$ PE's in the 2-D design. By merging the N rows in the original 2-D design into L rows, we have also merged the N input data streams into L data streams.

(c) Combine the data input to the columns into L data streams. To maintain $O(L)$ I/0 ports, the N data streams input to the columns are combined in L data streams. This requires pipelining the different data streams to their final destinations.

(d) Schedule the computation in each PE. In this step, we schedule the computations in the PE's so that an operation is not started until all its operands have arrived. Since we are mapping the computations in M PE's into a single PE, there are a very large number of possible mappings. In general, we need a mapping function of the form $\Phi_1(i,j,k) + \Phi_2(i,j)$, where $<i,j,k>$ is the index set of the computation to be performed, and $<i,j>$ is the coordinates of the PE to perform this operation. Note that Φ_1 is a partial ordering of the computations in each PE, and that Φ_2 determines the starting time of this computation. Φ_1 and Φ_2 should be chosen to preserve data-dependency. In practice, due to the uniformity of the communication and computations, we can have one one of two following choices: perform the computations in the M PE's sequentially, or interleave the computations in the M PE's.

(e) Calculate the velocities of the data and the number of buffers needed in each PE. We assign two parameters to every data stream: τ and σ, where τ is the time between using the same data in two neighboring PE's, and σ is the time between using the same data twice in the same PE. τ determines the speed of propagation, while σ determines the amount of buffers needed in each PE.

The speed of data flow can be controlled by inserting an appropriate number of buffers in the direction of the data flow. For instance, a shift register of length ρ can be used to delay a data item by ρ time units. However, a design using a fixed number of buffers in each PE is limited in its programmability and modularity. In our design, we assume that an appropriate delay is controlled by microcode stored in each PE.

4. MATRIX MULTIPLICATION

In this section, we apply the method discussed in the last section to design a L-by-N array for multiplying two N-by-N matrices, where $L \le N$.

Figure 1a shows an N-by-N systolic array for matrix multiplication, where N equals 4. This array completes the multiplication in $3N-2$ time units, and the result matrix is stored in the PE's.

In the N-by-N array (see Figure 1a), $PE_{i,j}$ performs matrix operations on elements with index set $<i,j,k>$, where $0 \le k < N$. By combining each of the $M = N/L$ rows into one row, $PE_{\hat{i},j}$ in the L-by-N array (see Figure 1b) performs operations on elements with index set $<i,j,k>$, where $0 \le k < N$, $M\hat{i} \le i < M\hat{i}+M$. In this case, a possible choice of Φ_1 is $i + kM - M\hat{i}$. This means that we interleave the operations in M PE's into one PE, and that we combine the M inputs to the M rows of PE's into one input. However, since each column input is used in M consecutive operations, we can pipeline the other $M-1$ column inputs during these $M-1$ time units.

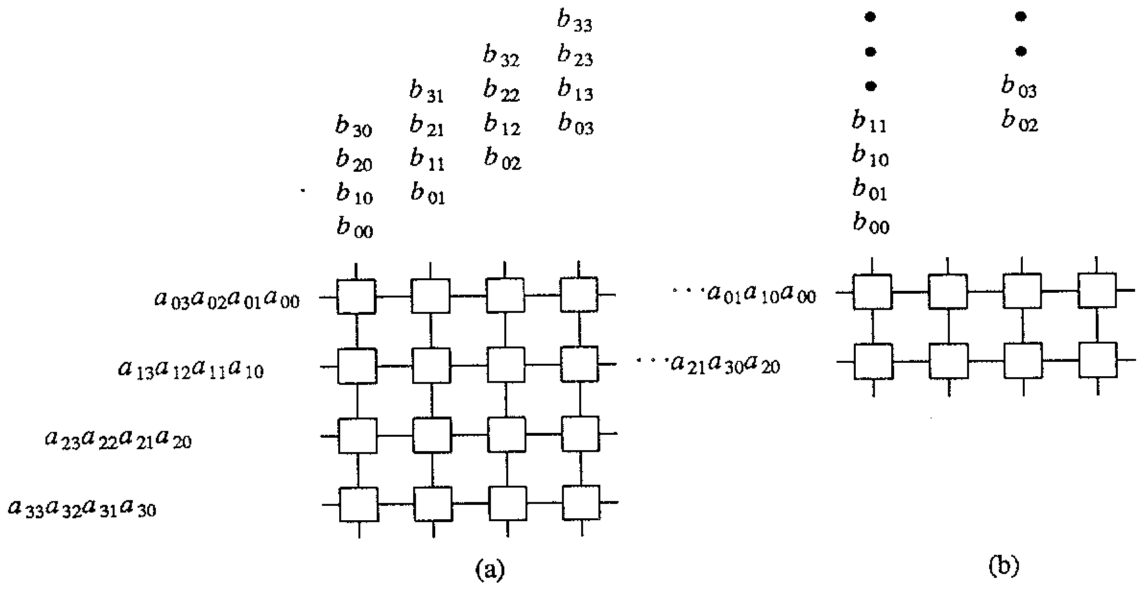

b_{33}
b_{32} b_{23}
b_{31} b_{22} b_{13}
b_{30} b_{21} b_{12} b_{03}
b_{20} b_{11} b_{02}
b_{10} b_{01}
b_{00}
$a_{03}a_{02}a_{01}a_{00}$
$a_{13}a_{12}a_{11}a_{10}$
$a_{23}a_{22}a_{21}a_{20}$
$a_{33}a_{32}a_{31}a_{30}$
(a)
b_{03}
b_{02}
b_{11}
b_{10}
b_{01}
b_{00}
$\cdots a_{01}a_{10}a_{00}$
$\cdots a_{21}a_{30}a_{20}$
(b)

Figure 1. Systolic arrays for matrix multiplication: (a) 4-by-4 systolic array (b) 2-by-4 systolic array.

$PE_{i,j}$ starts its calculation with $a_{i,0}$ and $b_{0,j}$. $a_{i,0}$ is input at time i, and it takes j time units to pipeline it to $PE_{i,j}$, hence requiring a total time of $i+j$ units. Similarly, $b_{0,j}$ is input at time j, and it arrives at $PE_{i,j}$ at time $i+j$ units. Hence, $\Phi_2(i,j) = i+2j$. The time to perform operation on element with index set $<i,j,k>$ in $PE_{i,j}$ is

$$t(i,j,k) = i + M k - M \lfloor i/M \rfloor + \lfloor i/M \rfloor + 2j$$

Figure 1b shows a 2-by-4 array for matrix multiplication.

Since the result matrix C is distributed in the different PE, we need M buffers per PE to store C. Recall that τ is the time between using the same data in two neighboring PE's, and σ is the time between using the same data twice in the same PE. Then

$$\tau_c = 0 \qquad \sigma_c = t(i,j,k+1) - t(i,j,k) = M$$

$$\tau_a = t(i,j+1,k) - t(i,j,k) = 2 \qquad \sigma_a = 0$$

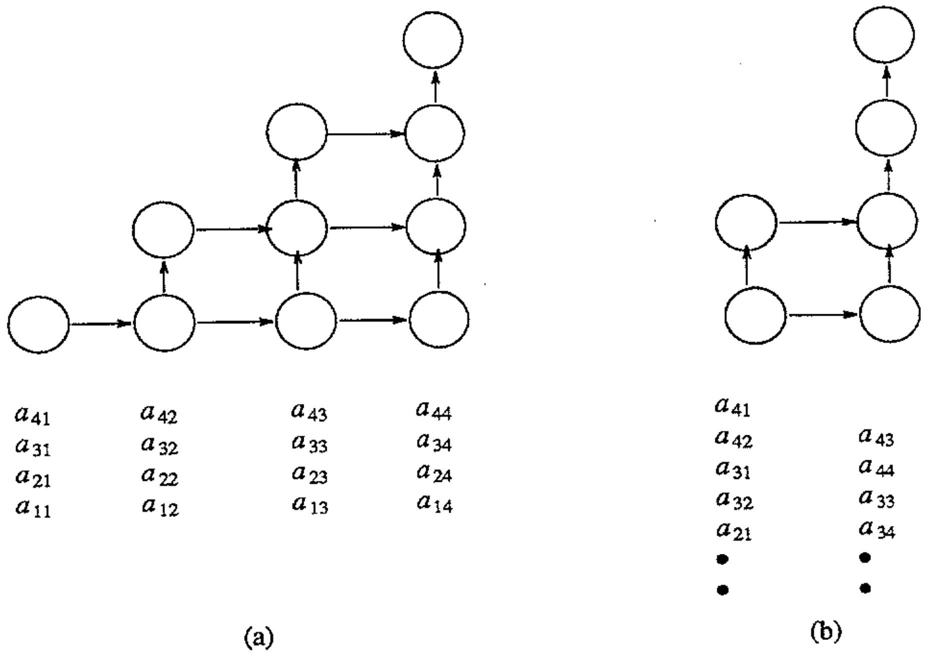
$$\tau_b = 0 \qquad \sigma_b = t(i+1,j,k) - t(i,j,k) = 1$$

Hence, the number of buffers required for each PE is L for storing C, 2 for delaying a, and 1 for holding b. Assuming in each PE that there is a memory of length at least $L+3$ words, we can use memory location 0 to $L-1$ for storing C, locations L, $L+1$ for delaying a, and location $L+2$ for holding b. We assume three data streams 1, 2, 3, each of which is identified by an input and an output buffer I_i, O_i, $i = 1, 2, 3$. Data stream 1 carries $a_{i,j}$ to the different PE's and travels in a horizontal direction. Data stream 2 travels in a vertical direction and propagates $b_{i,j}$ to the different PE's. Data stream 3 travels in a horizontal direction and pipelines $b_{i,j}$ to the PE's without I/O ports. The following microprogram can be used to multiply 2 matrices.

```
for $i$=0 to $N-1$
    for $j$=0 to $L-1$
        begin
        if ($j = 0$) then MEM[$L$+2] $\leftarrow I_2$
            else $O_3 \leftarrow I_2$                      /* pipeline $b$ to the next PE */
        $O_2 \leftarrow$ MEM[$L$+2]                        /* output b downwards */
        $O_1 \leftarrow$ MEM[$L+1$]                        /* delay $a$ */
        MEM[$L+1$] $\leftarrow$ MEM[$L$]                   /* for 2 time */
        MEM[$L$] $\leftarrow I_1$                          /* units */
        MEM[$j$] $\leftarrow$ MEM[$j$] + MEM[$L$+2] * MEM[$L+j$]
        end
```

5. QR DECOMPOSITION

QR decomposition, due to its numerical stability, is the preferred method for solving linear system of equations. In solving a system of equations $Ax = b$, matrix A can be written as the product of two matrices $A = QR$, where Q is a matrix with orthonormal columns, and R is an upper triangular matrix. This decomposition can be achieved by a series of Givens' Rotations. A triangular array for QR decomposition was designed by Kung [Kun88] and is shown in Figure 2a.



a_{41} a_{31} a_{21} a_{11}
a_{42} a_{32} a_{22} a_{12}
a_{43} a_{33} a_{23} a_{13}
a_{44} a_{34} a_{24} a_{14}
a_{41} a_{42} a_{31} a_{32} a_{21} • •
a_{43} a_{44} a_{33} a_{34} • •
(a)
(b)

Figure 2. Triangular array for QR decomposition: (a) Kung's design [Kun88]; (b) Reduced array.

In this section, we would like to design a reduced array for QR decomposition with L I/O ports. In this case, the array in Figure 2a can be reduced to a triangular L-by-N array by combining each of the $M = N/L$ columns into one column as shown in Figure 2b. In Figure 2b, operation on index element $<i,j,k>$ is performed in $PE_{\hat{j},k}$, where $\hat{j} = \lceil j/m \rceil$. The suitable mapping functions in this case are $\Phi_1 = M(N-i-1)+j-M\left\lfloor \lceil j/M \rceil - 1 \right\rfloor - 1$, and $\Phi_2 = \lceil j/M \rceil + k - 2$.

In this case, we have one vertical and one horizontal data streams: the horizontal data stream sends the θ across the columns, the vertical data stream sends the intermediate results of matrix A.

It is easy to prove that the number of buffers required for this design is one for θ, one for propagating the pipelined input, and $2M$ per PE for holding the intermediate results of A. Assuming that there are three input and three output buffers in each PE, x_i, y_i, x_o, y_o for vertical data streams, and θ_i, θ_o for horizontal data streams, the following microprogram implements the QR decomposition.

```
for $i = N-1$ to $k$
    for $j = 1$ to $M$
        begin
        $x = x_i$
        if $i = N-1$ then $y = y_i$
            else $y = x_0$
        if $j = 1$ then
            if $j = k$ then $\theta = \tan^{-1} x/y$
                else $\theta = \theta_i$
        $\theta_o = \theta$
        $x(j) = x \cos\theta + y \sin\theta$
    $y(j) = -x \sin\theta + y \cos\theta$
    $x_o = x(j)$
    $y_o = y(j)$
end
```

6. CONCLUSION

In this paper, we introduce the idea of a control flow systolic array, and apply it on the design of systolic arrays with constant number of input/output ports. The array is in the form of an L-by-N square mesh with $O(L)$ I/O ports, where L is a design parameter dependent on the current technology. The resulting array is a linear array by setting $L = 1$, whereas the resulting array is a square mesh by setting $L = N$. Designs for solving matrix multiplication and QR decomposition are used to illustrate the method.

7. ACKNOWLEDGEMENT

This research was supported by National Science and Engineering Council of Canada grants NSERC-OGP0043688 and NSERC-OGP0048888, and Joint Services Electronics Program Contract N00014-90-J-1270.

REFERENCES

[AbL91] M. A. Aboelaze, D.-L. Lee, and B. W. Wah, ''2-D Digital Filters on a Linear VLSI Array,'' *Proceedings of International Symposium on Circuits and Systems*, Singapore June 1991.

[Abo88] M. A. Aboelaze, *Systematic Design of Computational Arrays*, Ph.D. Thesis, Purdue University, West, Lafayette, IN., 1988.

[KuL87] S. Y. Kung, S. C. Lo, and P. S. Lewis, ''Optimal ystolic Design form the transitive Closure and the Shortest Path Problems,'' *IEEE Transactions on Computers*, Vol. C-36, No. 5, May 1987, pp. 603-614.

[Kun80] H. T. Kung, ''Highly Concurrent Systems Introduction to VLSI System,'' in *Introduction to VLSI Systems*, C. A. Mead and L. A. Conway, ed., Addison-Wesley, Reading MA, 1980.

[Kun82] H. T. Kung, ''Why Systolic Architecture,'' *IEEE Computer*, Vol. 15, No. 1, Jan. 1982, pp. 37-46.

[Kun88] S. Y. Kung, *VLSI Array Processors*, Prentice Hall, Englewood Cliffs, N.J., 1988.

[McM87] J. V. McCanny and J. G. McWhirter, ''Some Systolic Array Developments in the United Kingdom,'' *IEEE Computer*, Vol. 20, No. 7, July 1987, pp. 51-63.

[NaP87] J. G. Nash, K. W. Przytula, and S. Hansen, ''The Systolic/Cellular System for Signal Processing,'' *IEEE Computer*, Vol. 20, No. 7, July 1987, pp. 96-97.

[PrT88] V. K. Prasanna Kumar and Y. C. Tsai, ''Synthesizing Optimal Family of Linear Systolic Arrays for Matrix Computations,'' *Proc. International Confeence on Systolic Arrays*, K. Bromley, S. Y. Kung, and E. Swartzlander, ed., Computer Society Press, IEEE, May 1988, pp. 51-60.

[PrT89] V. K. Prasanna Kumar and Y. C. Tsai, ''Designing Linear Systolic Arrays,'' *Journal of Parallel and Distributed Computing*, Academic Press, New York, NY, 1989, pp. 441-463.

[RaV84] I. V. Ramakrishnan and P. J. Varman, ''Modular Matrix Multiplication on a Linear Array,'' *Proceedings of the 11th Annual Symposium on Computer Architecture*, ACM/IEEE, June 1984, pp. 232-238.