TWO-DIMENSIONAL DIGITAL FILTERING USING CONSTANT-I/O SYSTOLIC ARRAYS

Mokhtar A. Aboelaze, De-Lei Lee,
Dept. of Computer Science
York University
4700 Keele Street
North York, Ontario M3J 1P3
CANADA

Benjamin W. Wah
Coordinated Science Laboratory
University of Illinois
1308 West Main Street
Urbana, IL 61801-2307
U.S.A.

ABSTRACT

We present in this paper systolic arrays with constant number of input/output (I/O) ports for two-dimensional (2-D) FIR and IIR filtering. Our design has an array of $L \times N$ processing elements (PE's), where L ($\leq N$) is a technology-dependent parameter related to the number of I/O ports. Each PE in our design has a microprogrammed arithmetic logic unit (ALU), a control unit, a fixed number of I/O buffers, and $O(N/L)$ memory. Our design specializes to a square mesh when $L=N$, and a linear array when $L=1$. It can implement both FIR and IIR filtering in $O(N^2M/L)$ time which is asymptotically optimal.

1. INTRODUCTION

Systolic arrays [5] are widely considered to be efficient hardware solutions for satisfying the ever-increasing computational demands in 2-D digital signal processing. However, a large majority of previous designs require I/O ports that grow as polynomial functions of problem sizes. This is a major limitation that restricts the application of systolic arrays to relatively small problems. Recently, there have been considerable interests in the design of linear systolic arrays [6, 7] that require a constant number of I/O ports. These are exemplified by many designs of linear systolic arrays for 2-D image processing and 2-D signal filtering [1, 2, 3, 4]. Unfortunately, I/O in these linear designs is often the bottleneck, as it takes N^2 units of time to input an N-by-N array of numbers.

In this paper, we investigate the design of a programmable systolic array with constant number of I/O ports for 2-D FIR and IIR filtering. Our array is in the form of an L-by-N rectangular mesh with $O(L)$ I/O ports, where $1 \leq L \leq N$. Our design specializes to be a linear array when $L=1$, and a square mesh when $L=N$, where N is related to the problem size. Depending on the number of pins available, our design can provide a suitable trade-off between computational overhead and I/O complexity.

The architecture of each PE is simple: each PE has a control unit, an ALU that is capable of executing a small number of instructions, $O(c)$-word memory, and a fixed number of I/O buffers. We illustrate our design by showing optimal systolic arrays for implementing the M'th order 2-D FIR and IIR digital filters.

2. 2-D FIR DIGITAL FILTERING

An M'th-order 2-D FIR digital filter processes inputs $\{x_{i,j},\ 0 \leq i, j < N\}$ to form outputs $\{y_{m,n},\ 0 \leq i, j < N\}$, where

$$y_{m,n} = \sum_{i=0}^{M-1} \sum_{j=0}^{M-1} a_{i,j}\, x_{m-i,n-j}\,. \tag{1}$$

$$= \sum_{j=0}^{M-1} y^{(j)}_{m,n}\,, \tag{2a}$$

$$\text{where } y^{(j)}_{m,n} = \sum_{i=0}^{M-1} a_{i,j}\, x_{m-i,n-j}\,. \tag{2b}$$

2.1. 2-D FIR Filter in an N-by-M Processor Array

In this section we present the design of an N-by-M processor array for 2-D FIR filtering. Given an N-by-N array of numbers, we map the computation of each point in Eq. (2a) to a unique PE. Figure 1 shows the computation of $y_{4,4}$ in a 3-by-3 window ($M=3$). Column i of processors compute $y^{(i)}_{4,4}$, $0 \leq i < 3$, and the last row of processors compute $y_{4,4}$ according to Eq. (2b).

Research of M. Aboelaze were supported by National Science and Engineering Research Contract NSERC-OGP0009196. Research of D. Lee were supported by National Science and Engineering Research Contract NSERC-OGP0043688. Research of B. Wah was supported by Joint Services Electronics Program Grant N00014-90-J-1270.

Proc. IEEE Int'l Symp. on Circuits and Systems, 1993.

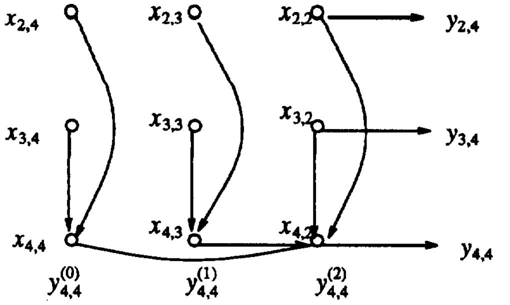

Figure 1. Computation of $y_{4,4}$

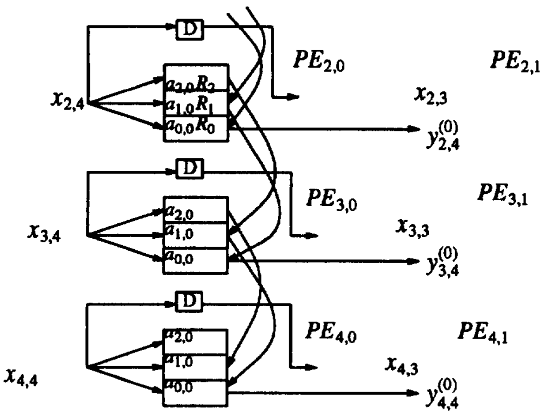

Figure 2
Calculation of $y^{(0)}_{4,4}$ in the first column of the PE array

Notice that the processor in the upper left-hand corner multiplies $x_{2,4}\, a_{2,0}$ to form part of $y_{4,4}$; it then multiplies $x_{2,4}\, a_{1,0}$ to form part of $y_{3,4}$, and $x_{2,4}\, a_{0,0}$ to form part of $y_{2,4}$. This means that each processor must have M different coefficients; *i.e.* $PE_{i,j}$ has $a_{\ell,j}$, $0 \le \ell < M$. Non-neighboring communication can be eliminated by pipelining intermediate results through neighboring PE's

The PE's can be arranged in an N-by-M mesh with each PE connected to its four neighbors. Each PE is assumed to have 6 I/O registers, 4 horizontal communication registers R_{w_1}, R_{w_2} (west), R_{e_1}, R_{e_2} (east), and two vertical communication registers, R_n (north), and R_s (south).

Figure 2 shows the data distribution in the first column of the array. Every PE has M registers (R_0, R_1, R_2 for $M=3$). $PE_{i,j}$ receives input x via R_{w_1}, y^ℓ (0 for the leftmost column) via R_{w_2}, and a partial result of R^ℓ (0 for the top row) via R_n. It stores the y's received in the appropriate registers, multiplies x by $a_{\ell,j}$, $0 \le \ell < M$, and adds the result to register R_ℓ. It then forwards the content of R_0 east to $PE_{i,j+1}$ via R_{e_2}, and sends x to $PE_{i,j+1}$ after one unit of delay via R_{e_1}. It also sends the contents of R_ℓ, $1 \le \ell < M$, south to be stored in $R_{\ell-1}$ in $PE_{i+1,j}$ via R_s.

Figure 2 further shows the computation of $y^{(0)}_{4,4}$. $PE_{2,0}$ receives $x_{2,4}$, multiplies it by $a_{2,0}$ and adds the result to R_2, multiplies it by $a_{1,0}$ and adds the result to R_1, and multiplies it by $a_{0,0}$ and adds the result to R_0. The content of R_0 is sent to the next column as $y^{(0)}_{2,4}$. At the same time, $x_{2,4}$ is sent to the next column after one unit of delay. In the next time step, $y^{(0)}_{2,4}$ meets $x_{2,3}$, and $y^{(1)}_{2,4}$ is computed. The contents of R_1 and R_2 of $PE_{2,0}$ are sent to $PE_{3,0}$ to be stored in R_0 and R_1, respectively. $PE_{3,0}$ stores $x_{3,4}a_{2,0}$ in R_2 to produce the first component of $y_{5,4}$. It also adds $x_{3,4}a_{1,0}$ to the content of R_1 to produce $x_{3,4}a_{1,0} + x_{2,4}a_{2,0}$. Finally, it adds $x_{3,4}a_{0,0}$ to the content of R_0. $PE_{3,0}$ then sends the content of register 0 to $PE_{3,1}$ as $y^{(0)}_{3,4}$. It also sends the contents of R_1 and R_2 to be stored in R_0 and R_1, respectively, in $PE_{4,0}$. $PE_{4,0}$ stores $x_{4,4}a_{2,0}$ in R_2, and adds $x_{4,4}a_{1,0}$ to register R_1. It adds $x_{4,4}a_{0,0}$ to register R_0 to produce $y^{(0)}_{4,4} = x_{4,4}a_{0,0} + x_{3,4}a_{1,0} + x_{2,4}a_{2,0}$. Finally, $PE_{4,0}$ sends the content of R_0 to $PE_{4,1}$ (next column), and sends the contents of registers R_1 and R_2 to $PE_{5,0}$ to be stored in registers R_0 and R_1, respectively.

The above array requires a memory of at least $2M+1$ words: M words to store the M filter coefficients, M words to store the intermediate results of $y^{(i)}$, and one memory location to simulate the delay register D.

The time required to complete the operations is $O(NM)$. Since the total number of operations is N^2M^2 and the number of processors is NM, the time complexity is asymptotically optimal. The number of buffer required for each PE is $2M+1$, where M is the order of the filter.

2.2. 2-D FIR Filter in a Constant I/O Mesh

The previous design needs $O(N)$ I/O ports, which is prohibitively expensive for large values of N. In this section, we show the design of a 2-D FIR filter on an L-by-M processor array with $O(L)$ I/O ports. This design reduces the number of I/O ports from N to L at the expense of increasing its time complexity from $O(NM)$ to $O(N^2M/L)$.

We accomplish this reduction in I/O ports by combining the $S = N/L$ PE's into one PE and the $S = N/L$ I/O ports into one port. In doing so, we merge the memories of the S PE's into the memory of one PE. Data movements in this design are the same as before, with the difference that data moving among the S PE's are now confined to one PE. Figure 3 shows this case when $S=3$ and $M=3$, and depicts the first column of the array

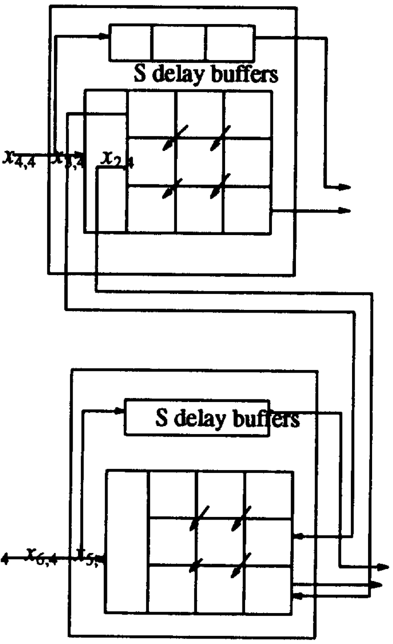

Figure 3
Combining the memory of 3 PE's into a single PE

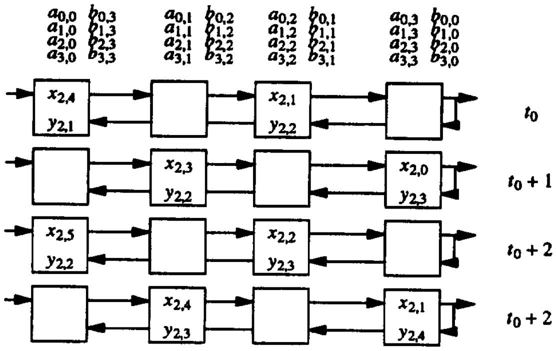

Figure 4
The first row of a VLSI array for IIR at times $t_0 \cdots t_0 + 2$

with the appropriate inputs. There are two differences between this design and the previous one.

(1) In combining S *PE*'s into one PE, data moving among these S PE's are now local movement in the same PE (represented by downward diagonal arrows in Figure 3).

(2) To guarantee correctness, x is delayed by S time units instead of one time unit.

Figure 3 shows the first column of such an array with the appropriate input for $M=3$ and $S=3$. Notice that 3 PE's are combined into one PE, which has 3 sets of buffers ($S=3$), with 3 buffers each ($M=3$) in the form of an M-by-S matrix. The arrows in Figure 3 indicate the data movement from R_t to R_{t-1}. However, each column of this matrix is used just once in a cycle, which means that we can replace the M S buffers by an array of S buffers and use this array M times.

Appendix I shows the algorithm (written in a C-like language) executed in each PE for 2-D FIR filtering in an L-by-N processor array. Note that the corresponding algorithm for a square processor array can be obtained by setting $S=1$.

3. 2-D IIR Digital Filter

A 2-D IIR digital filter is represented as follows.

$$y_{m,n} = \sum_{i=0}^{M-1} \sum_{j=0}^{M-1} a_{i,j}\, x_{m-i,n-j} + \sum_{\substack{i=0 \\ a+b\neq 0}}^{M-1} \sum_{j=0}^{M-1} b_{i,j}\, y_{m-i,n-j} \quad (3)$$

where $x_{i,j}$ is the input array, and $y_{i,j}$ is the output array. As is done in the FIR case, Eq. (3) can be rewritten as

$$y_{m,n} = \sum_{j=0}^{M-1} y^{(j)}_{m,n} \quad (4)$$

$$\text{where } y^{(j)}_{m,n} = \sum_{i=0}^{M-1} a_{i,j}\, x_{m-i,n-j} + \sum_{i=0}^{M-1} b_{i,j}\, y_{m-i,n-j}\,. \quad (5)$$

In computing $y_{i,j}$, we have to use the previously calculated y's. This can be achieved by feeding back the output of the PE array, as is shown in Figure 4, with two noticeable differences.

(1) $PE_{i,j}$ has the coefficients $a_{\ell,i}$, $b_{\ell,M-i-1}$, $0 \le \ell < M$. $PE_{i,j}$ calculates $x_{k,z}a_{\ell,i} + y_{k,z+2i+1-M}\, b_{\ell,M-i-1}$ as parts of $y_{k+\ell,z+i}$, $0 \le \ell < M$.

(2) In order for the right data to be at the right place at the right time, we have to input x to the PE array every other time unit. This lengthens the time for computing the IIR filter to $2N$ time units. Hence, the processors will alternate between idle cycles and busy cycles. Further, the speed of propagation of x is 1/3, *i.e.* x is delayed for 2 extra time units in each PE. The speed of propagation for y is 1.

Figure 4 shows the first row of the array for 4 consecutive time units. The rest of the columns behaves similarly as is in Figure 4. The algorithm for the 2-D IIR filtering is very similar to that for the 2-D FIR filtering except for two points.

(1) x is entered in a shift register of length 2 before it is sent to the next PE (delay of 3 per PE).

(2) Each PE has 6 horizontal I/O registers (R_{e_1}, R_{e_2}, R_{e_3}, R_{w_1}, R_{w_2}, R_{w_3}) and two vertical I/O registers (R_n, R_s), where R_{e_3} and R_{w_3} are used for storing the value of y for feedback.

Appendix II shows an algorithm for 2-D IIR filtering. Note that the shift register is represented as one operation; in practice, this can be implemented by using S memory locations. The procedure for 2-D IIR filtering using an L-by-M processor array is similar to that for 2-D FIR filtering except that a buffer of size S is used to delay y, and that a buffer of size $3S$ is used to delay x.

REFERENCES

[1] M. A. Aboelaze, D.-L. Lee, and B. W. Wah, "Two-Dimensional Digital Filtering using a Linear Processor Array," *Proc. Int'l Symp. on Circuits and Systems,* Singapore, June 1991, pp. 2943-2946.

[2] M. A. Sid-Ahmed, "A Systolic Realization for 2-D Digital Filters," *IEEE Trans. on Acoustics, Speech and Signal Processing,* Vol. 37, No. 4, April 1989, pp. 560-565.

[3] C. H. Chou and Y. C. Chen, "Modular Architectures for High Speed and Flexible Two-Dimensional Digital Filters," *Proc. Int'l Symp. on Circuits and Systems,* New Orleans, LA, May 1-3 1990, pp. 2320-2323.

[4] C.-H. Chou, "VLSI Architecture for High-Speed and Felxible Two-Dimentional Digital Filters," *IEEE Trans. on Signal Processing,* Vol. 39, No. 11, Nov. 1991, pp. 2515-2523.

[5] H. T. Kung, "Why Systolic Architecture," *IEEE Computer,* Vol. 15, No. 1, Jan. 1982, pp. 37-46.

[6] V. K. Prasanna Kumar and Y. C. Tsai, "On Mapping Algorithms to Linear and Fault Tolerant Systolic Arrays," *Proc. Int'l Conf. on Systolic Arrays,* 1988.

[7] V. K. Prasanna Kumar and Y. C. Tsai, "Designing Linear Systolic Arrays," *J. of Parallel and Distributed Computing,* Academic Press, 1989, pp. 441-463.

APPENDIX I

```
Procedure 2_D_FIR              /* on $L \times M$ array */
for ($\ell = 0$ to $N - 1$) do
    begin
    /* get the intermediate y's from $PE_{i-1,j}$
      and store them in $R_0 \cdots R_{M-2}$ * /
    for ($k = 0$ to $M - 2$ ) do
        $MEM[k] \leftarrow R_n$
    $MEM[M-1] \leftarrow 0$
    for ($s = 0$ to $s - 2$) do
        begin
        $MEM[0] \leftarrow MEM[0] + R_{w_2}$
        for ($k = 0$ to $M - 1$) do
            $MEM[k] \leftarrow MEM[k] + R_{w_1}\ a[k]$
        $R_{e_2} \leftarrow MEM[0]$            /*output $MEM[0]$ */
        for ($k = 0$ to $M - 2$) do
            $MEM[k] \leftarrow MEM[k+1]$
        $MEM[M-1] \leftarrow 0$
        /* Delay x for $S$ time units */
        store $R_{w_1}$ in $S$-buffer
        output $S$-buffer $\rightarrow$ $R_{e_1}$
        end
    for($k = 0$ to $M - 1$) do
        $MEM[k] \leftarrow MEM[k] + R_{w_1}\ a[k]$
    $R_{e_2} \leftarrow MEM[0]$                  /* output $MEM[0]$ */
    put $R_{w_1}$ in $S$-buffer          /* Delay x by $S$ time units */
    output $S$-buffer $\rightarrow$ $R_{e_1}$
    for ($k = 1$ to $k = M - 1$) do
        $R_s < -MEM[k]$
    end
```

APPENDIX II

```
Procedure 2_D_IIR              /* on $N \times M$ array */
for($\ell = 0$ to $N - 1$) do
    begin
    /* get the intermediate y's from $PE_{i-1,j}$
      and store them in $R_0 \cdots R_{M-2}$ */
    for ($k = 0$ to $M - 2$ ) do
        $MEM[k] \leftarrow R_n$
    $MEM[M-1] \leftarrow 0$
    $MEM[0] \leftarrow MEM[0] + R_{w_2}$
    for($k = 0$ to $M - 1$) do
        $MEM[k] \leftarrow MEM[k] + R_{w_1}\ a[k] + R_{e_3}\ b[k]$
    $R_{e_2} \leftarrow MEM[0]$            /*output $MEM[0]$ */
    Shift_register_of_length_3 $\leftarrow R_{w_1}$
    $R_{e_1} \leftarrow$ Shift_register_of_length_3
    $R_{w_3} \leftarrow R_{e_3}$
    for ($k = 1$ to $M - 1$) do
        $R_s \leftarrow MEM[k]$
    /* Output the content of $R_1$ to $R_{M-1}$ in $PE_{i+1,j}$ */
    end
```