TWO-DIMENSIONAL DIGITAL FILTERING
USING A LINEAR PROCESSOR ARRAY

Mokhtar A. Aboeleaze, De-Lei Lee
Department 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 linear VLSI array for implementing the $M$-th order 2-dimensional FIR and IIR digital filters. The linear array consists of $kM$ Processor Elements (PEs). Each PE consists of a control unit, ALU, storage unit, and input/output buffers. The control unit contains microinstructions to be executed by the ALU. Its design is simple and is capable of executing a few instructions such as reading a word from the memory, storing a result in the memory, and performing simple arithmetic operations. The storage element contains $O(N)$ words and can be accessed by the ALU. Input/output buffers are used for receiving data from other PEs and sending results before they are sent to neighboring PEs. The resulting speedup is $O(kM)$, where $k$ is a parameter governed by the tradeoff between the cost of the array and its speedup.

I. INTRODUCTION

Two-dimensional (2-D) digital filtering is very important in signal and image processing and exists in many applications such as radiology, computer tomography, computer vision, robotics, image transformation, and air reconnaissance [AdD84]. Many of these applications require real-time response, which call for the use of special-purpose hardware instead of general-purpose computers.

Special-purpose VLSI arrays have been introduced to handle computation-intensive tasks [Kun82]. The idea there is to exploit the inherent concurrency in these problems by using a large number of simple processors connected in a regular fashion. In order for these arrays to compete favorably with the more powerful and more expensive general-purpose computers, their design must be simple and their interconnections regular so that they can be laid out easily on a two-dimensional chip. Moreover, the arrays should be programmable so the same design can be used to solve more than one problem, hence amortizing the design cost of the array.

An important constraint on the design of a VLSI chip is number of input/output pins. While the manufacturing of VLSI chips with millions of transistors is feasible in the near future, the idea of manufacturing a chip with more than a few hundreds input/output pins is not realistic. This makes the linear arrays more attractive than a two-dimensional mesh of processors. Moreover, other factors, such as clock synchronization [FiK83] and the ease of reconfigurability in wafer scale integration [LeL85], are decisively in favor of linear arrays.

A systolic realization of 2-D digital filtering was introduced by Sid-Ahmed [Ahm89]. In his realization, broadcasting was used, although not recommended in a systolic realization. Chou and Chen [ChC90] developed a high-speed architecture for 2-D digital filtering. Their design requires $M$ PEs for implementing a $M$-by-$M$ filter, and the number of PEs cannot be extended beyond $M$. Finally, Kwan [Kwa90] proposed a multi-input, multi-output systolic array for first and second order 2-D IIR filtering.

In this paper, we develop a linear array for computing the 2-D FIR and IIR filters of an $N$-by-$N$ square image with a filtering of degree $M$. (The extension to rectangular images is straightforward.) This array contains $kM$ processors, $1 \leq k < M$, and completes the task in $N^2M^2/(kM)$ time units. Since the number of operations is $N^2M^2$, this array is asymptotically optimal. Note that in our design, $k$ is considered as a parameter that determines the tradeoff between the cost and the speedup of the systolic array.

2. THE LINEAR ARRAY MODEL

The processing model used in this paper is the Control Flow Systolic Array model developed by Aboeleaze [Abo88]. The idea there is to design an array processor of more powerful processing elements that are controlled by more complex control than that found in traditional systolic arrays. These processing elements can examine incoming data, determine the sequence of operations to be performed, and decide on the output ports that the results should be sent. Limited storage is also provided in each processor. The control used can be provided by an internal microprogram or by control signals traveling with the data. In this paper we assume that control is
provided by the microprogram, although our results also apply when control is supplied by the moving data.

A major advantage of this model is its versatility. The internal microprogram can be changed, hence the array can be applied to solving more than one problem and problems of various sizes [Abo88].

The architecture of each PE is simple: each PE contains a simple control unit/ALU that is capable of executing a small number of instructions, \( O(N) \)-word memory, and some I/O buffers. The existence of \( O(N) \) words of memory in each PE raises the question of modularity, namely, can we solve a problem of size \( 2N \) by connecting two or more arrays, each capable of solving a problem of size \( N \)? This is usually difficult, as the memory in each PE is a function of the problem. However, this problem can be circumvented if each PE contains a large enough memory. Figure 1 shows the architecture of a PE with two inputs and two outputs.

![Figure 1. Architecture of a PE](image)

3. 2-D FIR DIGITAL FILTER

A 2-D digital filter is given by the transfer function

\[
H(z_1, z_2) = \sum_{i=0}^{M-1} \sum_{j=0}^{M-1} a_{i,j} z_1^i z_2^j
\]

where \( M \) is the order of the filter, and \( a_{i,j} \) are the filter coefficients. If the input to the filter is \((x(i,j), 0 \leq i, j < N)\), then \( y(m,n) \), the output of this filter, is as follows:

\[
y(m,n) = \sum_{i=0}^{M-1} \sum_{j=0}^{M-1} a_{i,j} x(m-i,n-j)
\]

Eq. (2) can be rewritten as

\[
y(m,n) = \sum_{i=0}^{M-1} y^{(i)}(m,n)
\]

\[
y^{(i)}(m,n) = \sum_{j=0}^{M-1} a_{i,j} x(m-i,n-j)
\]

We can identify two levels of parallelism in computing \( y(m,n) \). Spatial parallelism can be applied in computing Eq. (3), and all \( y^{(i)}(m,n) \)'s can be calculated in parallel. Likewise, temporal parallelism can be applied in computing Eq. (4), and pipelining could be used to compute each of the \( a_{i,j} x(m-i,n-j) \).

In the simplest case, \( k = 1 \), and the array contains \( M \) PEs. In this case, spatial parallelism in Eq. (3) is exploited, and the speedup is \( O(M) \). As \( k \) increases, more temporal parallelism can be exploited. In the limit, \( k = M \), the resulting speedup is \( M^2 \), and the completion time is \( O(N^2) \). This is asymptotically optimal since we need \( O(N^2) \) to input the \( N \)-by-\( N \) image.

3.1. CASE 1 (\( k = 1 \))

In this section, we propose a linear systolic array for implementing 2-D digital filters. The array consists of \( M \) PEs, each of which has a control unit and \( N+M-1 \) memory words. The design of the controller is kept simple to facilitate VLSI implementation. The main function of the controller is to multiply two numbers and store the result in a specific location in the memory. The location of the memory word can be produced by a sequential counter (modulo \( (N+M-1) \)).

The inputs to the PE are two numbers \( x(i,j) \) and \( y(m,n) \), where \( x(i,j) \) is the input image, and \( y(m,n) \) is the partially calculated output. The coefficient of the filter are distributed among the PEs such that \( PE_i \) has \( (a_{0,i}, a_{1,i}, \ldots, a_{M-1,i}) \). The data to the array is input in a row major fashion, and the output is in the same format.

<table>
<thead>
<tr>
<th>Table 1. Timing of a degree 3 2-D FIR filtering of a 7-by-7 image</th>
<th>( y^{(0)}(m,n) )</th>
<th>( y^{(1)}(m,n) )</th>
<th>( y^{(2)}(m,n) )</th>
<th>( y^{(3)}(m,n) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( a_0 )</td>
<td>( a_{0,0} ), ( a_{0,1} ), ( a_{0,2} )</td>
<td>( a_{1,0} ), ( a_{1,1} ), ( a_{1,2} )</td>
<td>( a_{2,0} ), ( a_{2,1} ), ( a_{2,2} )</td>
<td>( a_{3,0} ), ( a_{3,1} ), ( a_{3,2} )</td>
</tr>
<tr>
<td>( a_1 )</td>
<td>( y(0,0) ), ( y(0,1) ), ( y(0,2) )</td>
<td>( y(1,0) ), ( y(1,1) ), ( y(1,2) )</td>
<td>( y(2,0) ), ( y(2,1) ), ( y(2,2) )</td>
<td>( y(3,0) ), ( y(3,1) ), ( y(3,2) )</td>
</tr>
<tr>
<td>( a_2 )</td>
<td>( y(0,0) ), ( y(0,1) ), ( y(0,2) )</td>
<td>( y(1,0) ), ( y(1,1) ), ( y(1,2) )</td>
<td>( y(2,0) ), ( y(2,1) ), ( y(2,2) )</td>
<td>( y(3,0) ), ( y(3,1) ), ( y(3,2) )</td>
</tr>
<tr>
<td>( a_3 )</td>
<td>( y(0,0) ), ( y(0,1) ), ( y(0,2) )</td>
<td>( y(1,0) ), ( y(1,1) ), ( y(1,2) )</td>
<td>( y(2,0) ), ( y(2,1) ), ( y(2,2) )</td>
<td>( y(3,0) ), ( y(3,1) ), ( y(3,2) )</td>
</tr>
</tbody>
</table>

In the linear array, \( PE_i \) calculates \( y^{(i)}(m,n) \)

\[
y^{(i)}(m,n) = \sum_{j=0}^{M-1} a_{i,j} x(m-i,n-j)
\]

where \( a_{i,j} \), \( 0 \leq i < M, 0 \leq j < M \).
receives \( x(i,j) \) and calculates the partial results of 
\( y(i+s,j+f) = y(i+s,j+f) + x(i,j) \cdot a_{s,f}, \quad 0 \leq s < M \). It then 
sends \( y(i,j) \) to the next PE. \( x(i,j) \) is stored (or delayed) in 
this PE for \((N+M-1)\) time units before it is sent to the next 
PE. Table 1 shows the timing for the various operations in 
this array for a 7-by-7 image and a 2-D FIR filter of degree 3. 
The \( y \)'s in the table represent the partial results of \( y(m,n) \).

As an illustration, at time 2, \( PE_0 \) receives \( x(0,0) \) and 
starts computing the partial results of \( y^{(0)}(0,2), y^{(0)}(0,1), 
y^{(0)}(0,0) \). It then sends \( y^{(0)}(0,0) \) to \( PE_1 \). At time 3, \( PE_0 \) 
receives \( x(0,1) \) and starts computing the partial results of 
\( y^{(0)}(0,3), y^{(0)}(0,2), y^{(0)}(0,1) \). It then sends \( y^{(0)}(0,1) \) to \( PE_1 \). 
\( PE_0 \) continues to compute the first row of the image. At time 
9, \( PE_0 \) receives \( x(1,0) \) and starts computing 
\( y^{(0)}(1,2), y^{(0)}(1,1), y^{(0)}(1,0) \). It then sends \( y^{(0)}(1,0) \) to \( PE_1 \). 
At time 11, \( PE_0 \) gets \( x(1,2) \) and starts computing 
\( y^{(0)}(1,4), y^{(0)}(1,3), y^{(0)}(1,2) \). It then sends \( y^{(0)}(1,2) \) and 
\( x(0,0) \) to \( PE_1 \). A similar process is repeated in the PEs until 
the filtering operation is completed.

In Figure 2, we present a simple procedure for performing 
2-D FIR digital filtering. We assume that each PE has 
local memory with the \( i \)th location labeled \( MEM[i] \). \( M+1 \) 
registers \( Z[0] \cdots Z[M] \), a register \( X \) for storing \( x(i,j) \), 
two input lines labeled \( I_1 \) and \( I_2 \), two output lines labeled \( O_1 \) 
and \( O_2 \), and \( a \) loaded in register \( a[1] \).

3.2. TIMING AND SPACE ANALYSIS

To find the total time of the filtering operation, we note 
that data is fed to the array in a row major form, and the output 
in the same form. If \( T_{PE} \) is the time each PE takes to perform 
\( M \) multiply-and-add operations, then the total time is 
\((M+M^2)T_{PE} \). The total time is, therefore,
\[ T = O((M + N^2)M T_a), \]
where \( T_a \) is the time to perform one multiply-and-add.

To find the space requirement, we need \( M \) buffers to 
calculate the \( y \)'s, \( M \) buffers to store the filter coefficients, 
\((N+M-1)\) buffers to delay the input for \((N+M-1)\) time units, 
and one buffer for \( x(i,j) \). The total space required is, therefore, 
\( N + 3M \).

3.3. CASE 2 (k > 1)

In the previous design, we exploited the parallelism in 
Eq. (3). However, the computation of Eq. (4) is done sequentially. 
The result is a cycle of length \( M T_a \), where \( T_a \) is the 
time to perform one multiply-and-add operation. That is true 
because every processor has to perform \( M \) multiply-and-add 
operations in each cycle.

One way to improve the processing time in each cycle is 
to use \( M k \) PEs and to carry out the \( M \) multiply-and-add 
operations by the \( k \) processors in parallel. This shortens 
the duration of each cycle from \( M T_a \) to \( M T_{a/k} \) and results in a 
total processing time of \( O(M^2/k) \). If \( k = M \), then the total 
time becomes \( O(N^2) \), which is the time required to input 
the data into a linear array. In this case, the \((N+M-1)\) units of 
delay should be implemented at the boundary processors 
only, namely, \( PE_1 \), where \( 0 = n k, 1 \leq n < M \).

---

**procedure 2-D FIR:**

/* Each PE receives two input data, \( X \) in input \( I_1 \), and \( Y \) 
in \( I_2 \). It calculates \( Y \) and output \( X \) to output \( O_1 \) and 
\( Y \) to output \( O_2 \); \( a[1] \) contains the filter coefficients. */

begin
  \[ h = 0; \]
  \[ X \leftarrow I_1; \]
  \[ Z[M] \leftarrow I_2; \]
  for \( i = 0 \) to \( M-1 \) do
    \[ Z[i] \leftarrow Z[i+1] + X \cdot a[i]; \]
    \[ O_2 \leftarrow Z[0]; \]
    \[ O_1 \leftarrow MEM[h]; \]
    \[ MEM[h+1] \leftarrow X \mod (N+M-1); \]
  \[ h = h + 1 \mod (N+M-1); \]
end

Figure 2. The microprogram stored in each PE to implement 
2-D FIR filtering.

---

4. 2-D IIR FILTER

A 2-D IIR digital filter is represented by the equation

\[ y(m,n) = \sum_{i=0}^{M-1} \sum_{j=0}^{M-1} a_{i,j} x(m-i,n-j) + \sum_{i=0}^{M-1} \sum_{j=0}^{M-1} b_{i,j} y(m-i,n-j) \]

where \( x(i,j) \) is the input image, and \( y(i,j) \) is the output 
image. To calculate the 2-D IIR filter, we need a feedback loop. 
This can be achieved by feeding back the output from 
\( PE_M \) to \( PE_{M-1} \) and back to \( PE_0 \). The output should 
be delayed for one row of data before sending it to the previous 
PE; this can be achieved by introducing a delay of \( N-M-1 \) 
units in the feedback path. Figure 3 shows the procedure for 
2-D IIR filtering.

This procedure is similar to the FIR case except that the 
output of the last PE is fed back to the previous PE's with a 
delay of \( N-M-1 \) at each PE. Figure 4 shows the array for 
2-D IIR filtering.

The IIR filtering takes the same time as in the FIR case. 
It requires \((N-M-1)\) extra buffers per PE to achieve the 
required delay in the feedback loop, i.e., \( 2N+4M \) buffers in 
total.

5. SUMMARY

In this paper, we introduce a control-flow linear array 
for implementing 2-D digital filtering. We use \( M \) processors 
to achieve a linear speedup of \( M \). The PEs used are 
simple and modular up to a pre-specified value of \( N \), where 
\( N \) is the dimension of the input image. The PEs can also be 
reprogrammed to solve a variety of problems.
procedure 2_D_IIR;
/* PEi receives three input data, X in input I1, Y in
I2 from PEi-1, and Y in I3 from PEi+1.
It computes Y and output X to output O1, Y to output O2
to PEi+1, and sends the feedback value on O3 to PEi-1.
a[ ] contains the filter coefficients. */
begin
    h = 0;
    e = H;  /* H > N+M-1 */
    X ← I1;
    Z[M] ← I3;
    if (i ≠ M-1) then
        begin
            MEM[e] ← I3;
            Y ← MEM[e+1] mod (N-M-1);
            e = e+1 mod (N-M-1);
        end
    else
        Y ← Z[1] + X*a[0]
    for i=0 to M-1 do
        Z[i] ← Z[i+1] + X*a[i] + Y*b[i];
    O3 ← Y;
    O2 ← Z[0];
    O1 ← MEM[h];
    MEM[h-1] ← X mod (N+M-1);
    h = (h+1) mod (N+M-1)
end

Figure 3. The microprogram stored in each PE to implement:
2-D FIR filtering.

Figure 4. A linear array for 2-D IIR filter

6. REFERENCES