VLSI Architectures for the Finite Impulse Response Filter

KAM HOI CHENG AND SARTAJ SAHNI, MEMBER, IEEE

Abstract—**We review the various VLSI architectures that have been proposed for the finite impulse response filter problem. In addition, new architectures are proposed and improved designs for some of the earlier architectures are developed.**

I. Introduction

VLSI architectures for a variety of problems have been proposed by several authors. A bibliography of over 150 research papers dealing with this subject appears in [1]. In this paper, we are concerned solely with the finite impulse response (FIR) filter problem. The input to this problem is a $1 \times w$ vector $A = (a_1, a_2, \cdots, a_w)$ and a $1 \times (n+w)$ vector $X = (x_0, x_1, \cdots, x_{n+w-1})$. The output is a $1 \times (n+1)$ vector $Y = (y_0, y_1, \cdots, y_n)$ where

$$y_i = \sum_{j=1}^{w} a_j x_{i+j-1}, \qquad i = 0, 1, \cdots, n. \tag{1}$$

The FIR problem is of interest to VLSI designers as (1) is difficult to solve in real time for wide-band signal processing applications. Furthermore, the FIR problem has relatively favorable sensitivity properties with respect to round-off errors. Consequently, most application needs can be satisfied using fixed-point arithmetic of moderate precision. Additionally, the problem may be solved by algorithms requiring a simple control and flow of data. This facilitates highly parallel and pipelined computation.

VLSI architectures for this problem have been proposed earlier in [2]–[6]. The design of [3] and [4] uses a unidirectional chain of processors as in Fig. 1(a). Here data flow is from left to right. The design of [2], [5], and [6] employs a bidirectional chain of processors as in Fig. 1(b). In this, data flows both from left to right and from right to left.

In this paper, we examine each of the above two architectures. In addition, we consider broadcast chains [Fig. 1(c)] as used in [7]–[11] (among others) for the matrix multiplication, back substitution, LU decomposition, and the adaptive recursive filtering problems. A broadcast line has the property that data put on this line and becomes available at all PE's on the line in O(1) time. Furthermore, the systolic ring architecture [Fig. 1(d)] used in [3] for the LU decomposition problem is also examined.

Manuscript received May 28, 1985; revised July 29, 1985. This work was supported in part by the National Science Foundation under Grant MCS-83-05567.

K. H. Cheng is with the Department of Computer Science, University of Houston, Houston, TX 77004.

S. Sahni is with the Department of Computer Science, University of Minnesota, Minneapolis, MN 55455.

IEEE Log Number 8406184.

In evaluating our designs, we assume that the VLSI system will be attached to the host processor using a bus as in Fig. 2. The evaluation of a VLSI design should take the following into account.

1) Bus bandwidth—How much data is to be transmitted between the host and the VLSI system in any cycle? This figure is denoted by B.

2) Speed—How much time does the VLSI system need to complete its task? This time may be decomposed into the times T_C (time for computations) and T_D (time for data transmissions both within the VLSI system and between the host and the VLSI system).

One may expect that by using a very high bandwidth B and a large number of processors P, we can make T_C and T_D quite small. So, T_C and T_D are not in themselves a very good measure of the effectiveness with which the resources B and P have been used. Let D denote the total amount of data that needs to be transmitted between the host and VLSI system. The ratio

$$R_D = B * T_D / D$$

measures the effectiveness with which the bandwidth B has been used. Clearly, $R_D \geqslant 1$ for every VLSI design. As an example, consider the multiplication of two $n \times n$ matrices. The host needs to send $2n^2$ elements to the VLSI system and receive n^2 elements back. So, $D = 3n^2$. With a bandwidth of n, T_D must be at least $3n$. T_D will exceed $3n$ if the bandwidth is not used to capacity at all times.

Let C denote the time spent for computation by a single processor algorithm. The ratio

$$R_C = P * T_C / C$$

measures the effectiveness of processor utilization. Once again, we see that $R_C \geqslant 1$ for every VLSI design. Consider the problem of multiplying two $n \times n$ matrices A and B to get X. Each element of X is the sum of n products. We shall count one multiplication and addition as one arithmetic (or computation) step. Hence, $C = n^3$. If $P = n$, then $T_C \geqslant n^2$.

In evaluating a VLSI design, we shall be concerned with T_C and T_D and also with R_C and R_D. We would like R_C and R_D to be close to 1. Finally, we may combine the two

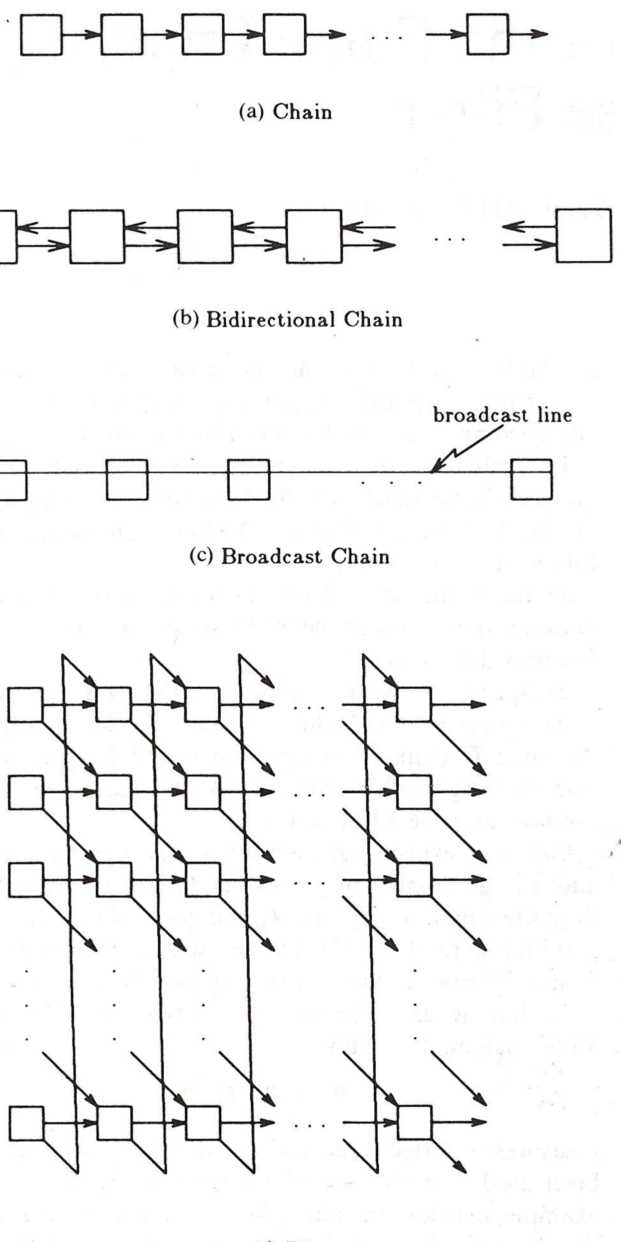

Fig. 1. Different VLSI architectures. (a) Chain. (b) Bidirectional chain. (c) Broadcast chain. (d) Ring.

efficiency ratios R_C and R_D into the single ratio $R = R_C * R_D$. A design that makes effective use of the available bandwidth and processors will have R close to 1.

The efficiency measure R as defined here is the same as that used in [8] and [9] to evaluate VLSI designs for matrix multiplication and back substitution. This measure is also quite similar to that proposed in [7]. In fact, the two measures become identical when $T_C = T_D$.

For each of the designs considered in this paper, we compute R_C, R_D, and R. In several cases, our designs have lower efficiency ratios than all earlier designs using the same model. In comparing different architectures for the same problem, one must be wary about overemphasizing the importance of R_C, R_D, and R. Clearly, using $P=1$ and $B=1$, we can get $R_C = R_D = R = 1$ and no speedup at all. So, we are really interested in minimizing T_C and T_D while keeping R close to 1. Some of our designs reduce T_C at the expense of R.

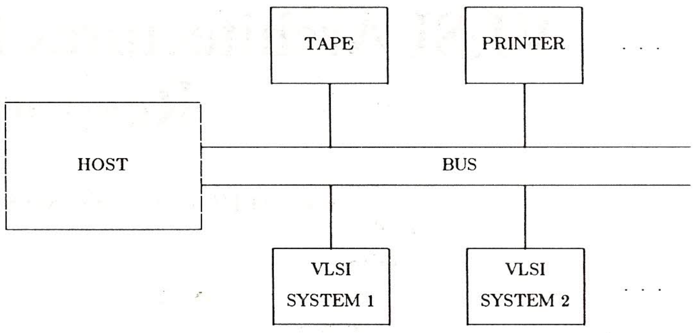

Fig. 2. Connecting to a host computer.

II. Finite Impulse Response (FIR) Filter

A. The Problem

Input:	A $1 \times w$ vector A and a $1 \times (n+w)$ vector X.
Output:	A $1 \times (n+1)$ vector Y that satisfies (1).
Parameters:	$C = (n+1)w$, $D = 2(n+w)+1$.

B. Broadcast Chain

The broadcast capability permits a very straightforward and efficient solution to the FIR filter problem. We provide two such solutions. The first solution uses w PE's and has a throughput of $T_C = n + w$. The VLSI architecture is shown in Fig. 3(a) and the algorithm executed by the system is given in Fig. 4. Each PE has three registers, A, X, and Y. $Z(i)$ denotes register Z of PE(i), $Z \in \{A, X, Y\}$.

The algorithm consists of two FOR loops. The first initializes the A register of each PE such that $A(i) = a_i$, $1 \leqslant i \leqslant w$. Following this initialization, the system goes through $n + w - 1$ steps in which the following happens.

1) Each PE receives the same x value from the broadcast line.

2) Each PE sends its Y value to the PE on its right. PE(1) initializes its Y register to zero. PE(w) outputs its Y register.

3) Each PE updates its Y register to $Y + AX$.

The first two events occur concurrently, while the third occurs after the first two have completed. It is easy to see that the first w Y's output by PE(w) are garbage and the next $n+1$ Y's output are $y_0, y_1, \cdots, y_n$ as given by (1).

In computing the performance figures for this VLSI system, we note that $B = 2$ as the system requires at most one input and one output to occur concurrently. Observe that there is no Y input to the system as PE(1) simply initializes $Y(1)$ to 0 internally. So, we get $P = w$, $B = 2$, $T_C = n + w$, $T_D = n + 2w + 1$, $R_C \sim 1 + w/n \sim 1$, $R_D \sim 1 + w/(n+w) \sim 1$ and $R \sim 1$.

The throughput of the VLSI system may be reduced to $o(n/k)$ by using k independent subsystems as in Fig. 3(b). Each subsystem has w PE's. Subsystem b computes y_i, $\lceil (n+1)/k \rceil b \leqslant i < \lceil (n+1)/k \rceil (b+1)$, $0 \leqslant b \leqslant k$. The per-

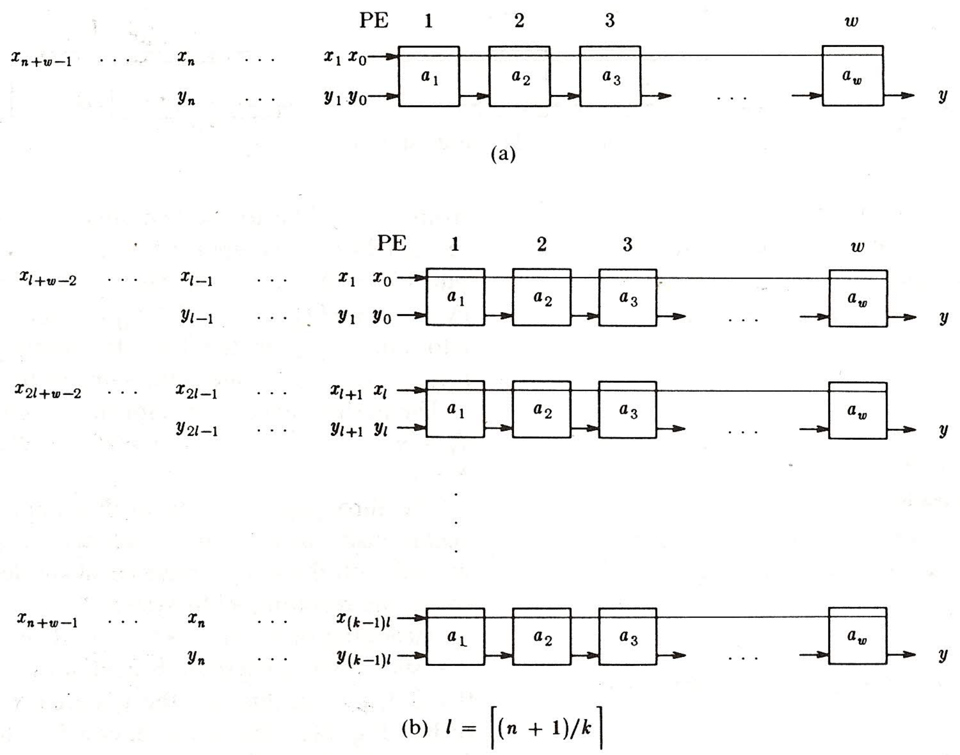
PE 1 2 3 w
x_{n+w-1} ... x_n ... x_1 x_0
y_n ... y_1 y_0
a_1 a_2 a_3 a_w
y
(a)
x_{l+w-2} ... x_{l-1} ... x_1 x_0
y_{l-1} ... y_1 y_0
x_{2l+w-2} ... x_{2l-1} ... x_{l+1} x_l
y_{2l-1} ... y_{l+1} y_l
x_{n+w-1} ... x_n ... $x_{(k-1)l}$
y_n ... $y_{(k-1)l}$
(b) $l = \lceil (n+1)/k \rceil$

Fig. 3. *w* PE broadcast chain.

```
for $j \leftarrow 1$ to $w$ do
    $A(j) \leftarrow a_j$          { input $a_j$ }
end
for $j \leftarrow 0$ to $n + w - 1$ do
    do in parallel
        broadcast $x_j$ to $X(i)$,      $1 \le i \le w$
        $Y(i) \leftarrow Y(i-1)$      $2 \le i \le w$
        $Y(1) \leftarrow 0$
        output $Y(w)$        { output $y_{j-w}$ }
    end
    $Y(i) \leftarrow Y(i) + A(i) * X(i)$        $1 \le i \le w$      { each PE computes }
end
output $Y(w)$        { output $y_n$ }
```

Fig. 4. Algorithm for *w* PE broadcast chain.

formance figures are $P = kw$, $B = 2k$, $T_C = \lceil (n+1)/k \rceil + w - 1$, $T_D = \lceil (n+1)/k \rceil + 2w$, $R_C \sim 1 + (kw)/n \sim 1$, $R_D \sim 1 + 2kw/(n+w) \sim 1$, and $R \sim 1$.

One deficiency of this improved throughput design is that the y_i's are not output in order (for example, y_1 is output after $y_{\lceil (n+1)/k \rceil}$).

Our second broadcast chain VLSI design for the FIR filter problem employs $n+1$ PE's. Hence, it is applicable only when n is known in advance. Often, in applications, the value of n is determined by examining the y_i's. Hence, it is a dynamic quantity.

The n PE architecture is shown in Fig. 5 and the corresponding algorithm is shown in Fig. 6. As in the previous design, each PE has the three registers: A, X, and Y. PE(i) computes y_i, $0 \leqslant i \leqslant n$. The contents of the X registers following the first FOR loop are shown in Fig. 5. In the second FOR loop, the $n+1$ PE's compute their respective y_i's. This takes w iterations in which each PE does the following:

1) receive a_j into its A register on the jth iteration,

2) shift its X register content to the PE on its right. PE(n) inputs a new X while PE(0) simply loses its old X value,

3) update its Y value to $Y + AX$.

Events 1) and 2) are performed concurrently while event 3) occurs after the first two have been completed. Once w iterations of this loop are completed, $Y(i) = y_i$, $0 \leqslant i \leqslant n$. The $n+1$ Y values may now be output in $n+1$ time.

In obtaining the performance figures, we assume that the initialization of the Y registers to zero is overlapped with the initialization of the A registers. Consequently, we get $P = n+1$, $B = 2$, $T_C = w$, $T_D = 2n + w + 1$, $R_C = 1$, $R_D \sim 2$, and $R \sim 2$.

As in the case of our earlier design, the throughput can be improved by using k subsystems, each having $\lceil (n+1)/k \rceil$ PE's. The distribution of the y_i's computed by each subsystem is the same as in the earlier design. The performance figures are $P \sim n+1$, $B = 2k$, $T_C = w$, $T_D = 2\lceil (n+1)/k \rceil + w - 1$, $R_C \sim 1$, $R_D \sim 2 + (kw)/(n+w) \sim 2$, and $R \sim 2$. Note that when $k = n+1$, this reduces to the case of $n+1$ independent PE's.

C. Bidirectional Chain

Kung [2] and Leiserson [5] have proposed an O(n) bandwidth bidirectional chain architecture to solve the band-matrix × vector problem. Since the FIR filter problem is a special case of the band-matrix × vector problem,

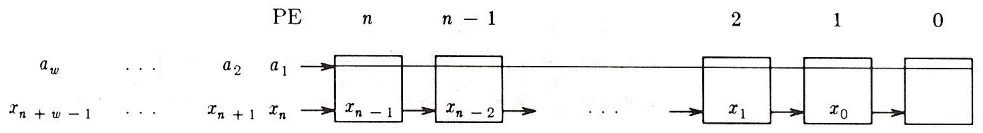

Fig. 5. n PE broadcast chain.

```
Y(i) ← 0        0 ≤ i ≤ n
for j ←0 to n − 1 do        {input x_0,...,x_{n−1}}
    do in parallel
        X(i) ← X(i + 1)        0 ≤ i ≤ n − 1
        X(n) ← x_j
    end
end
for j ←1 to w do
    do in parallel
        broadcast a_j to A(i),        0 ≤ i ≤ n
        X(i) ←X(i + 1)        0 ≤ i ≤ n − 1
        X(n) ←x_{n+j−1}
    end
    Y(i) ← Y(i) + X(i) * A(i)        0 ≤ i ≤ n
end
output y_0,...,y_n
```

Fig. 6. Algorithm for n PE broadcast chain.

the same architecture can also be used to solve the FIR filter problem. However, since only one row of coefficients of the band matrix is unique, only O(1) bandwidth is required. The resulting architecture is shown in Fig. 7.

The x values are input from the right with one x value entering every two cycles. The a values may be entered from the left using the left-to-right connections. Hence, the initialization process takes w cycles and leaves $A(i) = a_i$, $1 \leqslant i \leqslant w$ and $X(i) = x_{\lfloor i/2 \rfloor}$ for i odd, $1 \leqslant i \leqslant w$. Following this, there are $2n + w$ cycles in which each processor updates its Y register to $Y + AX$ and then transmits its Y register content right and its X register content left. A new y and x value enter the system every two cycles.

The performance figures for this design are readily seen to be $P = w$, $B = 2$, $T_C = 2n + w$, $T_D = 2(n + w)$, $R_C \sim 2$, $R_D \sim 2$, and $R \sim 4$.

Leiserson [5] has pointed out that pairs of processors may be combined, as only half the processors are active at any time. When this is done, R_C and R become 1 and 2, respectively.

Robert and Tchuente [6] have proposed a $w+1$ PE bidirectional chain with improved throughput. The same throughput can be obtained using only w PE's. Our new bidirectional chain design is similar to that of [6] and to that proposed in [9] for the back-substitution problem. Fig. 8(a) and (b) shows an example for each of the cases: w even and w odd. The corresponding algorithm is shown in Fig. 9. The PE's at the left compute all terms involving the a_i's for i odd while those on the right compute all terms for i even. All PE's (except PE(w)) have three registers A, X, and Y. PE(w) has an additional register Z. Y values move towards the middle PE, PE(w), while X values move from the middle to the two ends. In each cycle, each PE (except PE(w)) computes $Y \leftarrow Y + AX$, while PE(w) computes $Y \leftarrow (Y + Z) + AX$. Hence, the computation time per cycle is $\max\{t_{\text{ADD}}, t_{\text{MUL}}\} + t_{\text{ADD}}$ where t_{ADD} is the time to add and t_{MUL} is the time to multiply. Assuming that $t_{\text{ADD}} \leqslant t_{\text{MUL}}$, this time is the same as for the other designs.

The performance figures for the design are $P = w$, $B = 2$, $T_C = n + \lfloor w/2 \rfloor + 1$, $T_D = n + 2w + 1$, $R_C \sim 1$, $R_D \sim 1$, and $R \sim 1$.

The throughput can be further improved. One way to accomplish this is to use k subsystems of w PE's each as we did with the w PE broadcast chain design. As remarked there, the resulting VLSI system does not compute the y_i's in ascending order of i. An alternative is to distribute the y_i's such that subsystem b computes y_i, $i = b + jk$, $j = 0, 1, 2, 3, \cdots$. In this case the y_i's are computed in the right order. Fig. 8(c) shows the design for the case $w = 8$ and $k = 2$. The first subsystem is used to compute all the y_i's for i even and the second to compute all the y_i's for i odd. The computation of each such y_i is divided into four independent subcomputations. We note that the method may be extended to get an $o(n/k)$ throughput design for any fixed k by computing k y_i's in each cycle. Unlike the back-substitution problem [9], the scalar product (1) does not need to be put into another form since all the x_i's are available before computation starts. The VLSI system that incorporates this is quite a bit more complex. We shall not present the details here.

D. Unidirectional Chain

References [3] and [4] have proposed a unidirectional chain design to solve the FIR filter problem. Fig. 10(a) shows their architecture. The corresponding algorithm is given in Fig. 11. Each PE has four registers: A, X, Y, and Z. The Z register is used to delay the rightward motion of the x's. Each x value enters a PE through its Z register, then moves to the X register in the next cycle, and exits on the following cycle to the Z register of the PE on its right.

The operation of the VLSI system has two phases to it. The first is the initialization phase in which $A(i)$ is initialized to a_i. While this is happening, half the PE's get their X and Z registers initialized to x values. Fig. 10(b) shows the status after this. In each cycle of the second phase, the X, Z, and Y values move rightwards and the Y values get updated. Fig. 10(c) shows the status following w cycles of the second phase. The algorithm is given in Fig. 11.

The performance figures of this design are $P = w$, $B = 2$, $T_C = n + w$, $T_D = n + 2w + 1$, $R_C \sim 1 + w/n \sim 1$, $R_D \sim 1 + w/(n + w) \sim 1$, and $R \sim 1$.

Once again, the throughput may be improved using the strategy of Section II-B. An alternative which results in the

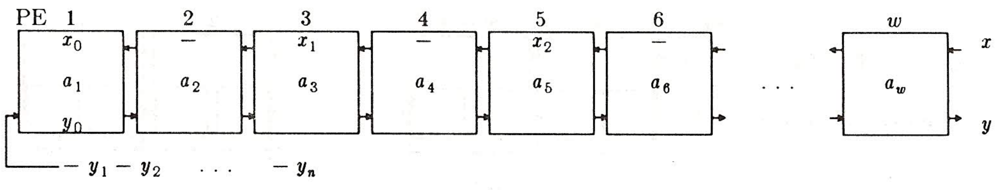

Fig. 7. Bidirectional chain.

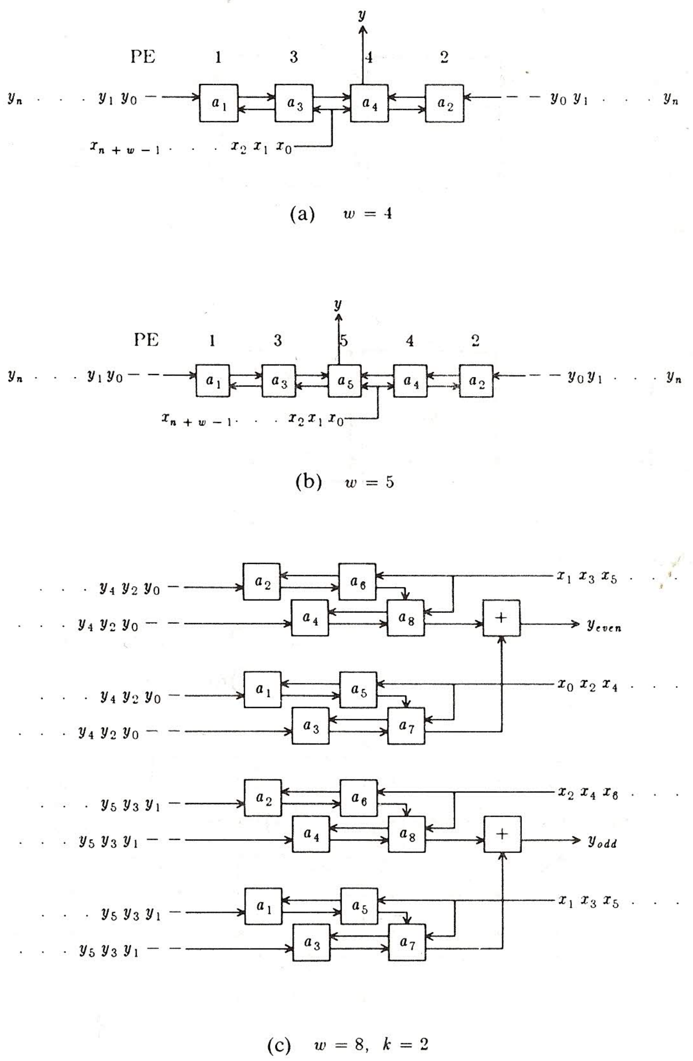

Fig. 8. Improved throughput bidirectional chain. (a) $w = 4$. (b) $w = 5$. (c) $w = 8$, $k = 2$.

```
$A(i) \leftarrow a_i$    $1 \le i \le w$    { input $a_i$ }
for $j \leftarrow 1$ to $\lfloor (w-1)/2 \rfloor$ do
    do in parallel
        $X(w) \leftarrow X(w-1) \leftarrow x_{j-1}$
        $X(i) \leftarrow X(i+2)$    $1 \le i \le w-2$
    end
end
for $j \leftarrow 1$ to $n + w$ do
    do in parallel
        $X(w) \leftarrow X(w-1) \leftarrow x_{j + \lfloor (w-1)/2 \rfloor - 1}$
        $X(i) \leftarrow X(i+2)$    $1 \le i \le w-2$
        $Y(i) \leftarrow Y(i-2)$    $3 \le i \le w$
        $Y(1) \leftarrow Y(2) \leftarrow 0$
        $Z(w) \leftarrow Y(w-1)$
        output $Y(w)$    { output $y_{j - \lfloor w/2 \rfloor - 2}$ }
    end
    do in parallel
        $Y(w) \leftarrow (Y(w) + Z(w)) + A(w) * X(w)$
        $Y(i) \leftarrow Y(i) + A(i) * X(i)$    $1 \le i \le w-1$
    end
end
output $Y(w)$    { output $y_n$ }
```

Fig. 9. Algorithm for improved throughput bidirectional chain.

y_i's being output in ascending order of i is described below. For any fixed k, each PE has $k+1$ X registers and k Y registers. A total of w PE's are used. The $k+1$ registers are set up in k rows with one row having two registers. The row with two registers shifts up cyclically as we move to the right on the chain of PE's. Fig. 12(a) shows the case $k = 2$, $w = 5$ and Fig. 12(b) shows the case $k = 3$, $w = 5$. Each PE has enough hardware to update all its registers in parallel. In each cycle of the VLSI system, the following events occur.

1) Each PE transmits X values to the right. For some X registers, this is an inter-PE transfer [e.g., x_{11} moves to the register currently containing x_9 in Fig. 12(a)]. For others, this is an intra-PE transfer (e.g., x_{13} is transferred to the register currently containing x_{11}). The leftmost PE inputs k x values into its k leftmost X registers.

2) Each PE transmits k Y values to the right. The leftmost PE zeros its k Y registers.

3) Each PE updates its k Y registers by computing $Y \leftarrow Y + AX$. The X value used varies with each PE. For example, in Fig. 12(a), each PE uses its square register to update all even y_i's and the circle register for the odd y_i's.

Events 1) and 2) occur in parallel while event 3) occurs after events 1) and 2) have completed. Let us trace y_8 through the system of Fig. 12(a). We see that the terms accumulated are $x_{12}a_5 + x_{11}a_4 + x_{10}a_3 + x_9a_2 + x_8a_1$. A formal correctness proof is easily obtained.

In obtaining the performance figures for this design, we count each PE as k PE's as the hardware in each is about k times that in each of the PE's used in our earlier designs. So, we get $P \sim kw$, $B = 2k$, $T_C = \lceil (n+1)/k \rceil + w - 1$,

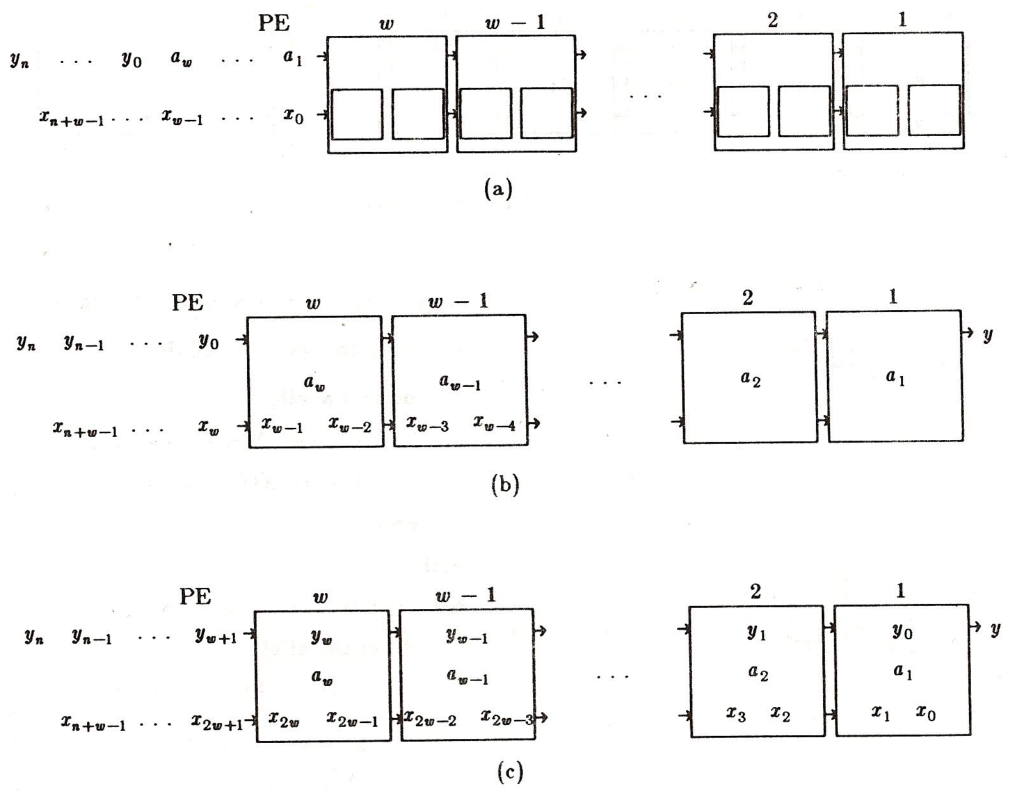

Fig. 10. Unidirectional chain.

```
for $j \leftarrow 1$ to $w$ do
    do in parallel
        $A(w) \leftarrow a_j$              { input $a_j$ }
        $A(i) \leftarrow A(i+1)$           $1 \le i \le w-1$
        $Z(w) \leftarrow x_{j-1}$          { input $x_{j-1}$ }
        $X(i) \leftarrow Z(i)$             $1 \le i \le w$
        $Z(i) \leftarrow X(i+1)$           $1 \le i \le w-1$
    end
end
for $j \leftarrow 1$ to $n+w$ do
    do in parallel
        $Y(w) \leftarrow 0$                { initialize $y_{j-1}$ to 0 }
        $Y(i) \leftarrow Y(i+1)$           $1 \le i \le w-1$
        output $Y(1)$                      { output $y_{j-w-1}$ }
        $Z(w) \leftarrow x_{j+w-1}$
        $X(i) \leftarrow Z(i)$             $1 \le i \le w$
        $Z(i) \leftarrow X(i+1)$           $1 \le i \le w-1$
    end
    $Y(i) \leftarrow Y(i) + A(i) * X(i)$   $1 \le i \le w$
end
output $Y(1)$          { output $y_n$ }
```

Fig. 11. Algorithm for unidirectional chain.

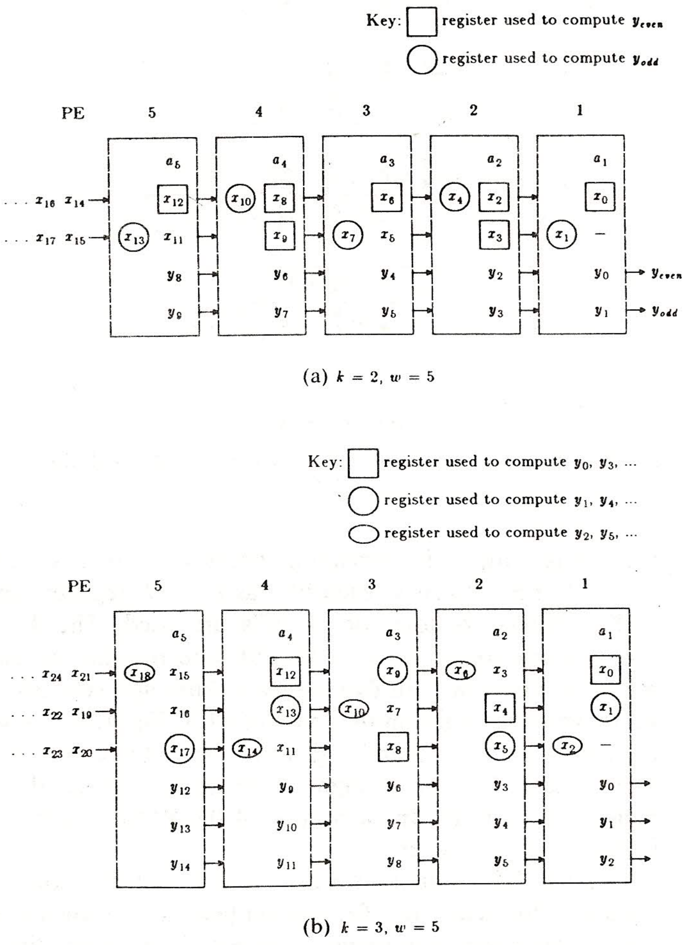
Key: □ register used to compute y_{even}
○ register used to compute y_{odd}
PE
(a) $k = 2$, $w = 5$
Key: □ register used to compute y_0, y_3, ...
○ register used to compute y_1, y_4, ...
⬭ register used to compute y_2, y_5, ...
(b) $k = 3$, $w = 5$

Fig. 12. Improved throughput unidirectional chain. (a) $k = 2$, $w = 5$. (b) $k = 3$, $w = 5$.

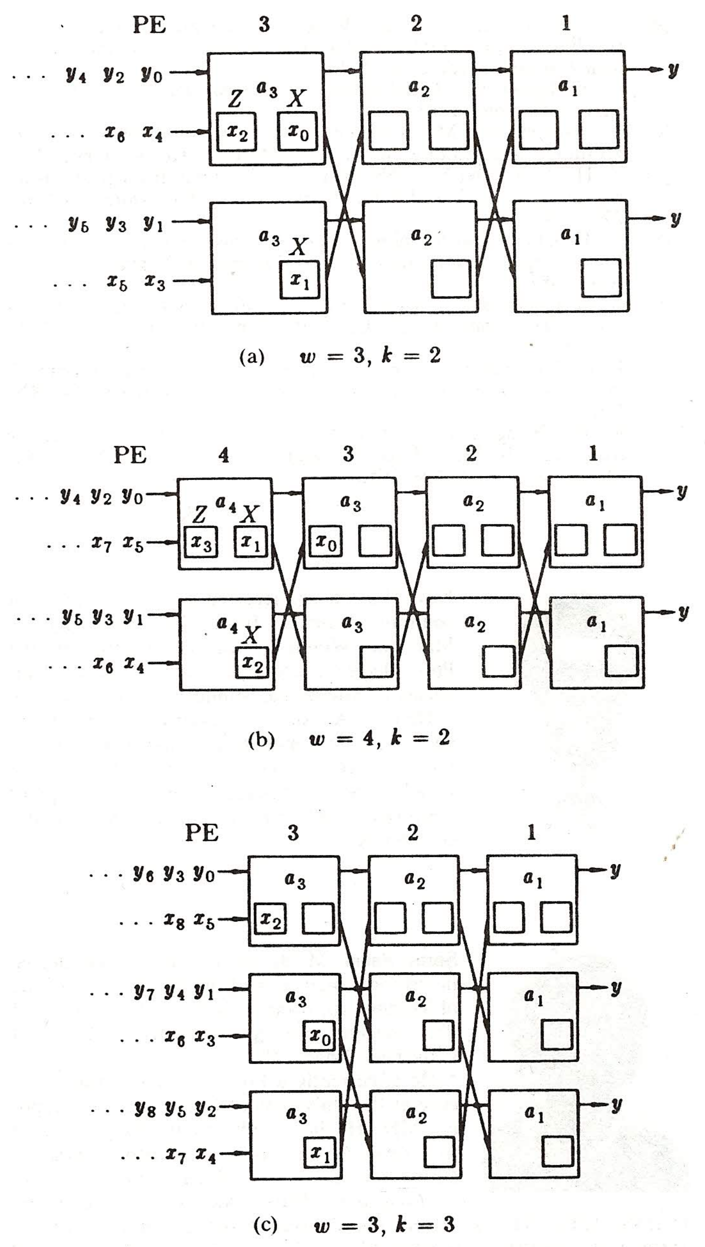

Fig. 13. Systolic ring. (a) $w=3$, $k=2$. (b) $w=4$, $k=2$. (c) $w=3$, $k=3$.

$T_D\lceil (n+1)/k \rceil + 2w$, $R_C \sim 1 + (kw)/n \sim 1$, $R_D \sim 1 + 2kw/(n+w) \sim 1$, and $R \sim 1$.

E. Systolic Ring

A simpler improved throughput architecture results from using a systolic ring. Our design requires kw PE's and is as shown in Fig. 13. First, consider the case $k=2$. The upper row of PE's [Fig. 13(a) and (b)], row 1, computes all the even y_i's ($y_0, y_2, \cdots$), while the bottom row of PE's computes all the odd y_i's ($y_1, y_3, \cdots$). Fig. 13(a) shows the case for $w=3$, $k=2$, where k is the number of rows of PE's and Fig. 13(b) shows the case for $w=4$, $k=2$. As can be seen, there is a minor difference in the input pattern between the case w odd and the case w even. We will consider only the case when w is even. The case when w is odd is similar or it can be simulated by adding one more term so that w becomes even. The working of the system is described formally in Fig. 14. $P(i, j)$ refers to register P of PE(j) of row i, $P \in \{A, X, Y, Z\}$, $i \in \{1,2\}$.

```
for $j \leftarrow 1$ to $w$ do        { $w$ is assumed even }
    do in parallel
        $A(1,w) \leftarrow A(2,w) \leftarrow a_j$            { input $a_j$ }
        $A(p,i) \leftarrow A(p,i+1)$                 $1 \le i \le w-1,\ \ 1 \le p \le 2$
        $Z(1,w) \leftarrow x_{2(j-w/2-1)+1}$         $j > w/2$
        $X(2,w) \leftarrow x_{2(j-w/2-1)}$           $j > w/2$
        $X(1,i) \leftarrow Z(1,i)$                   $1 \le i \le w$
        $Z(1,i) \leftarrow X(2,i+1)$                 $1 \le i \le w-1$
        $X(2,i) \leftarrow X(1,i+1)$                 $1 \le i \le w-1$
    end
end
for $j \leftarrow 1$ to $\lfloor n/2 \rfloor + w$ do
    do in parallel
        $Y(p,w) \leftarrow 0$                        $1 \le p \le 2$
        $Y(p,i) \leftarrow Y(p,i+1)$                 $1 \le i \le w-1,\ \ 1 \le p \le 2$
        output $Y(p,1)$                              $1 \le p \le 2$
        $Z(1,w) \leftarrow x_{2(j+w/2-1)+1}$
        $X(2,w) \leftarrow x_{2(j+w/2-1)}$
        $X(1,i) \leftarrow Z(1,i)$                   $1 \le i \le w$
        $Z(1,i) \leftarrow X(2,i+1)$                 $1 \le i \le w-1$
        $X(2,i) \leftarrow X(1,i+1)$                 $1 \le i \le w-1$
    end
    $Y(p,i) \leftarrow Y(p,i) + A(p,i) * X(p,i)$     $1 \le i \le w,\ \ 1 \le p \le 2$
end
output $Y(p,1)$        $1 \le p \le 2$
```

Fig. 14. Algorithm for systolic ring.

The performance figures for this design are $P = 2w$, $B = 4$, $T_C = \lfloor n/2 \rfloor + w$, $T_D = \lfloor n/2 \rfloor + 2w + 1$, $R_C \sim 1 + (2w)/n \sim 1$, $R_D \sim 1 + (3w)/(n+w) \sim 1$, and $R \sim 1$.

The above design can be generalized to the case of kw PE's for k any constant. Fig. 13(c) shows the case when $w = k = 3$. The resulting architecture is similar to the systolic ring architecture for LU decomposition proposed by Kung and Lam [4]. The only difference is that data moves in the same direction instead of moving in opposite directions. The performance figures for this design are easily seen to be $P = kw$, $B = 2k$, $T_C = \lfloor n/k \rfloor + w$, $T_D = \lfloor n/k \rfloor + 2w + 1$, $R_C \sim 1 + kw/n$, $R_D \sim 1 + (2k-1)w/(n+w)$, and $R \sim (1 + kw/n)(1 + (2k-1)w/(n+w))$.

III. Summary

In this paper, we have reviewed previously known VLSI architectures for the FIR problem and also developed some new ones. Specifically, we have developed new solutions for broadcast chains, bidirectional chains, and rings. In addition, we have shown how the throughput may be improved for all of the architectures considered. The performance figures for the basic VLSI architectures considered in this paper are summarized in Table I. As can be

TABLE I
PERFORMANCE SUMMARY

Performance	Architecture				
	Broadcast Chain		Bidirectional Chain		Unidirectional Chain
			[KUNG79]	Our	[KUNG82]
P	w	$n+1$	$w/2$	w	w
B	2	2	2	2	2
T_C	$n+w$	w	$2n+w$	$n+\lceil w/2 \rceil$	$n+w$
T_D	$n+2w$	$2n+w$	$2(n+w)$	$n+2w$	$n+2w$
R_C	1	1	1	1	1
R_D	1	2	2	1	1
R	1	2	2	1	1

$C = (n+1)w,\ D = 2(n+w)+1$

TABLE II
PERFORMANCE SUMMARY

Performance	Architecture				
	Broadcast Chain		Bidirectional Chain	Unidirectional Chain	Systolic Ring
P	kw	$n+1$	$kw+k(k-1)$	kw	kw
B	$2k$	$2k$	$k(k+1)$	$2k$	$2k$
T_C	$\left\lceil \frac{n+1}{k} \right\rceil + w$	w	$o(n/k)$	$\left\lceil \frac{n+1}{k} \right\rceil + w$	$\left\lceil \frac{n}{k} \right\rceil + w$
T_D	$\left\lceil \frac{n+1}{k} \right\rceil + 2w$	$2\left\lceil \frac{n+1}{k} \right\rceil + w$	$o(n/k)$	$\left\lceil \frac{n+1}{k} \right\rceil + 2w$	$\left\lceil \frac{n}{k} \right\rceil + 2w$
R_C	$1+(kw)/n$	1	1	$1+(kw)/n$	$1+(kw)/n$
R_D	$1+\frac{2kw}{n+w}$	$2+\frac{kw}{n+w}$	$(k+1)/2$	$1+\frac{2kw}{n+w}$	$1+\frac{(2k-1)w}{(n+w)}$
R	1	2	$(k+1)/2$	~ 1	~ 1

$C = (n+1)w,\ D = 2(n+w)+1$

seen, all three architectures: broadcast chain, unidirectional chain, and bidirectional chain approach an R of 1 while providing about the same throughput. Table II summarizes the performance figures for the improved throughput designs. Only in the case of a bidirectional chain are we unable to get a design with $R=1$.

REFERENCES

[1] H. T. Kung, "A listing of systolic papers," Dep. Comput. Sci., Carnegie-Mellon Univ., Pittsburgh, PA, May 1984.

[2] H. T. Kung, "Let's design algorithms for VLSI systems," in *Proc. CALTECH Conf. VLSI*, Jan. 1979, pp. 65–90.

[3] H. T. Kung, "Why systolic architectures?," *IEEE Comput.*, vol. 15, no. 1, pp. 37–46, Jan. 1982.

[4] H. T. Kung and M. Lam, "Wafer scale integration and two-level pipelined implementations of systolic arrays," *J. Parallel Distributed Processing*, vol. 1, no. 1, 1984.

[5] C. E. Leiserson, *Area-Efficient VLSI Computation*. Cambridge, MA: MIT Press, 1983.

[6] Y. Robert and M. Tchuente, "Designing efficient systolic algorithms," Laboratoire IMAG, Saint Martin D'Heres, France, 1984.

[7] K. H. Huang and J. A. Abraham, "Efficient parallel algorithms for processor arrays," *IEEE Int. Conf. Parallel Processing*, 1982, pp. 271–279.

[8] K. H. Cheng and S. Sahni, "VLSI systems for matrix multiplication," Dep. Comput. Sci., Univ. of Minnesota, Minneapolis, MN, Aug. 1984.

[9] K. H. Cheng and S. Sahni, "VLSI architectures for back substitution," Dep. Comput. Sci., Univ. of Minnesota, Minneapolis, MN, Nov. 1984.

[10] K. H. Cheng and S. Sahni, "VLSI architectures for LU decomposition," Dep. Comput. Sci., Univ. of Minnesota, Minneapolis, MN, Jan. 1985.

[11] K. H. Cheng and S. Sahni, "A new VLSI system for adaptive recursive filtering," Dep. Comput. Sci., Univ. of Minnesota, Minneapolis, MN, Apr. 1985.

Kam Hoi Cheng, received the B.Sc. degree in computer science from the University of Manitoba, Winnipeg, Canada, in 1980, and the Ph.D. degree in computer science from the University of Minnesota, Minneapolis, MN, in 1985.

He is an Assistant Professor of Computer Science at the University of Houston, Houston, TX. His research interests include VLSI systems, parallel and distributed computing, computer networks, computer architecture, algorithms, and complexity.

Sartaj Sahni (M'79) received the B.Tech. degree in electrical engineering from the Indian Institute of Technology, Kanpur, India, and the M.S. and Ph.D. degrees in computer science from Cornell University, Ithaca, NY.

He is currently a Professor of Computer Science at the University of Minnesota, Minneapolis, MN. He has published in *Journal of the Association for Computing Machinery*, *Journal of Computers and Systems Science*, *SIAM Journal on Computing*, IEEE TRANSACTIONS ON COMPUTERS, IEEE TRANSACTIONS ON COMPUTER-AIDED DESIGN OF INTEGRATED CIRCUITS AND SYSTEMS, *ACM Transactions on Mathematical Software*, *Operations Research*, *International Journal on Theoretical Computer Science*, *Mathematics of Operations Research*, *Journal of Parallel and Distributed Computing*, *Journal of Algorithms*, and the *Journal of Statistical Computation and Simulation*. His publications are on the design and analysis of efficient algorithms, parallel computing, interconnection networks, and design automation. He is a coauthor of the texts *Fundamentals of Data Structures* and *Fundamentals of Computer Algorithms*, and author of the texts *Concepts in Discrete Mathematics* and *Software Development in Pascal*.

Dr. Sahni is the area editor for *Parallel Algorithms* and *Data Structures for the Journal of Parallel and Distributed Computing*.