Parallel Computing 12 (1989) 53–69
North-Holland

VLSI architectures for back substitution *

Kam Hoi CHENG ** and Sartaj SAHNI

Computer Science Department, 4-192 EE&CS Building, University of Minnesota, Minneapolis, MN 55455, U.S.A.

Received August 1987
Revised January 1989

Abstract. VLSI architectures involving unidirectional, bidirectional and broadcast chains as well as unidirectional rings are examined for the back substitution problem. We review some of the earlier designs and also develop new designs with superior performance.

Keywords. VLSI architectures, systolic systems, back substitution.

1. 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 [5]. In this paper, we are concerned solely with the back substitution problem. The inputs to this problem are a non-singular lower triangular matrix A and a column vector b. The objective is to determine the unique column vector x with the property $Ax = b$. Throughout this paper, we assume that A is $n \times n$. So, x and b are $n \times 1$. The classical approach to obtain x is to use back substitution. [1] x is obtained using the formula

$$x_i = \left(b_i - \sum_{j=1}^{i-1} a_{ij} x_j \right) / a_{ii}, \quad 1 \leqslant i \leqslant n.$$

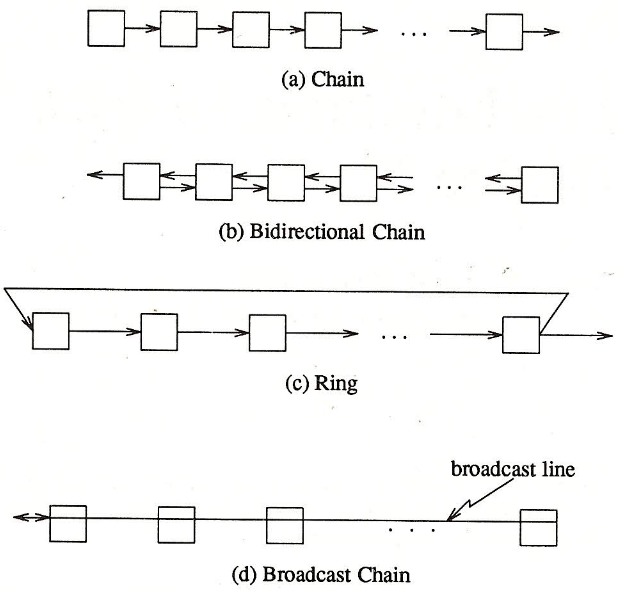
The x_i's are computed in the order $x_1, x_2, \ldots, x_n$. The matrix A is assumed to be dense below the diagonal. As a result, we do not develop any special methods to handle sparsity or bandedness. Furthermore, it should be noted that the factorization process generally used to obtain A may ensure that A is unit lower triangular, i.e., all diagonal elements are 1. In this case, the divisions by a_{ii}, above, may be eliminated.

VLSI architectures for the back substitution problem have been proposed earlier in [2,4,6–8]. The design of [2] employs a unidirectional chain of processors as in Fig. 1(a). The data flow is left to right. The design of Kung and Leiserson [4,7,8] employs a bidirectional chain as in Fig. 1(b). Here, data is permitted to flow both from left to right and from right to left. In [6], a ring architecture such as in Fig. 1(c) is proposed for this problem. Data can flow in only a single direction (either clockwise or counterclockwise) around the ring.

* This research was supported in part by the National Science Foundation under grant MCS-83-05567 and MIP-86-17374.

** Current address: Computer Science Department, University of Houston, Houston, TX, U.S.A.

[1] Strictly speaking, we are really doing forward substitution and not back substitution. In back substitution, A is upper triangular. However, forward and back substitution are quite similar and we prefer to use the generic term "back substitution" here.



(a) Chain

(b) Bidirectional Chain

(c) Ring

(d) Broadcast Chain

Fig. 1.

We examine each of the above three architectures here. In addition, we consider the broadcast chain used in [3] and [1] (among others) for the matrix multiplication problem. To our knowledge, there has been no earlier research on the use of broadcast chains for back substitution. An example of a broadcast chain is given in Fig. 1(d). A broadcast line has the property that data put on this line becomes available at all PEs on the line in O(1) time.

In evaluating our designs, we assume that the VLSI system will be attached to the host processor using a bus. 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 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 [1] to evaluate VLSI designs for matrix multiplication. This measure is also quite similar to that proposed in [2]. 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 improved efficiency ratios than all earlier designs using the same model. In comparing different architectures for the same problem, one must be wary about over emphasizing 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 speed up 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.

2. Back substitution

2.1. The problem

Input: A lower triangular matrix A and an $n \times 1$ vector b.
Output: An $n \times 1$ vector x such that $Ax = b$.
Parameters: $C = \frac{1}{2}n(n+1) \sim \frac{1}{2}n^2$, $D = \frac{1}{2}n(n+5) \sim \frac{1}{2}n^2$.

2.2. Chain with O(n) bandwidth

An n-PE chain may be used as in Fig. 2 (dashes indicate no input). Each PE has one adder, multiplier and divider. PE(j) computes x_j. Row j of A (excluding the elements beyond the main diagonal) is input to PE(j) element by element beginning at time j. PE(j) inputs b_j at time j. The computed x-values flow to the right and get output from the right end of the chain. A formal description of the algorithm executed by each PE is given in Algorithm 1. In this

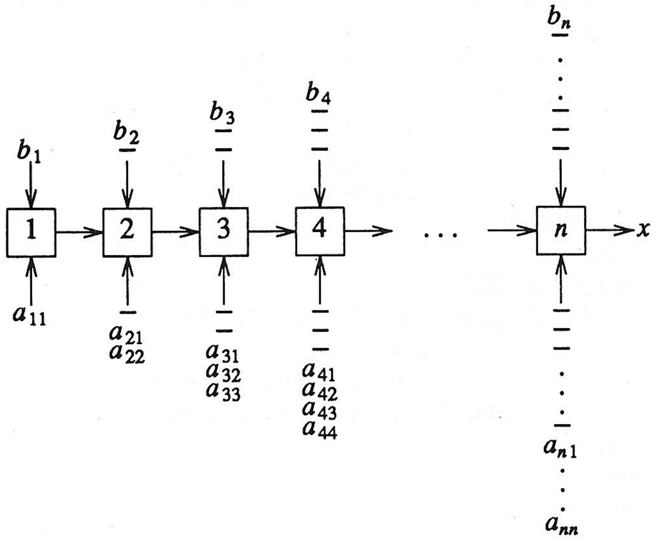


Fig. 2.

```
for $i \leftarrow 1$ to $2n - 1$ do
   do in parallel   {Input/Output}
      if $i \le n$ then $c_i \leftarrow b_i$
            else output $x_n$
      $A_j \leftarrow a_{j,i-j+1}$,   $(i + 1)/2 \le j \le \min\{i, n\}$
      $x_j \leftarrow x_{j-1}$,        $2 \le j \le n$
   end
   do in parallel
      if $i$ odd then [ $x_{\lceil i/2 \rceil} \leftarrow c_{\lceil i/2 \rceil} / A_{\lceil i/2 \rceil}$
                   $c_j \leftarrow c_j - A_j * x_j$,   $j \ne \lceil i/2 \rceil$ ]
             else [ $c_j \leftarrow c_j - A_j * x_j$,   $1 \le j \le n$ ]
   end
end
output $x_n$
```

Algorithm 1.

algorithm, the notation Y_i, $Y \in \{c, x, A\}$ means register Y of PE(i). The correctness of this algorithm is readily established.

From Fig. 2 and Algorithm 1, we see that maximum input/output occurs when $i = n$. At this time, a bandwidth of $\lceil \frac{1}{2}n \rceil + 1$ is needed.

Performance

The performance figures for this scheme are $P = n$, $B = \lceil \frac{1}{2}n \rceil + 1$, $T_C = 2n - 1$, $T_D = 2n$, $R_C \sim 4$, $R_D \sim 2$ and $R \sim 8$.

```
for $i \leftarrow 1$ to $2n - 1$ do
   do in parallel   {Input/Output}
      if $i \le n$ then $c_{(i-1) \bmod n/2 + 1} \leftarrow b_i$
      $A_j \leftarrow a_{j,i-j+1}$,            $(i + 1)/2 \le j \le \min\{i, n/2\}$
      $A_j \leftarrow a_{j+n/2,i-j-n/2+1}$,   $(i + 1)/2 \le j + n/2 \le \min\{i, n\}$
      case
         $1 < i \le n/2$   : $x_j \leftarrow x_{j-1}$,   $2 \le j < n/2$
         $n/2 < i \le n$   : $x_j \leftarrow x_{j-1}$,   $2 \le j < n/2$
                             $x_1 \leftarrow x_{n/2}$
         $i > n$           : $x_j \leftarrow x_{j-1}$,   $2 \le j < n/2$
                             output $x_{n/2}$
      endcase
   end
   do in parallel
      if $i$ odd then [ $k \leftarrow (\lceil i/2 \rceil - 1) \bmod n/2 + 1$
                     $x_k \leftarrow c_k / A_k$
                     $c_j \leftarrow c_j - A_j * x_j$,   $j \ne k$ ]
                else [ $c_j \leftarrow c_j - A_j * x_j$,   $1 \le j \le n/2$ ]
   end
end
output $x_{n/2}$
```

Algorithm 2.

2.3. *Ring with* O*(n) bandwidth*

An examination of Algorithm 1 reveals that at most $\lceil\frac{1}{2}n\rceil$ PEs do useful work at any time. Hence, we may double up the use of the first $\lceil\frac{1}{2}n\rceil$ PEs by requiring PE(i) to compute both x_i

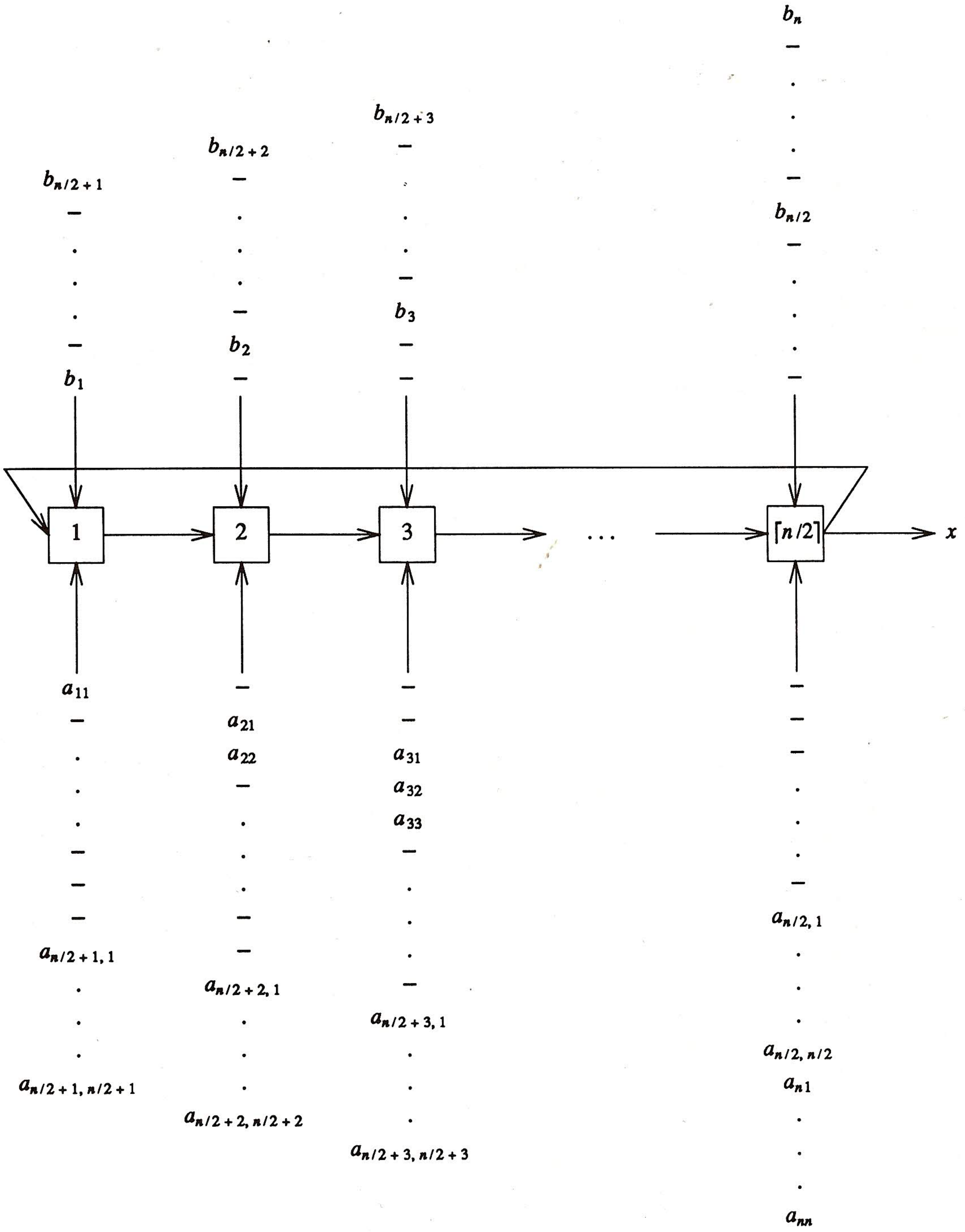


Fig. 3.

and $x_{i+\lceil n/2\rceil}$. In case n is odd, PE($\lceil \frac{1}{2}n\rceil$) computes only $x_{\lceil n/2\rceil}$. For simplicity in exposition, we assume that n is even (in case n is odd, an extra column and row may be added to A and an extra value added to b).

The input data pattern is shown in Fig. 3 and the algorithm executed by each PE is given in Algorithm 2 Once again, correctness is easily established.

Performance

The performance figures are $P = \lceil \frac{1}{2}n\rceil$, $B = \lceil \frac{1}{2}n\rceil + 1$, $T_C = 2n - 1$, $T_D = 2n$, $R_C \sim 2$, $R_D \sim 2$ and $R \sim 4$. So even though only half as many PEs are used, T_C and T_D are unchanged!

2.4. Bidirectional chain with O(n) bandwidth

Kung [7] has proposed the n-PE bidirectional chain of Fig. 4. PE(1) computes the x-values. On the first cycle, it does this by inputting b_1 and a_{11} and computing $x_1 = b_1/a_{11}$. Every odd cycle thereafter, a c-value is received from PE(2) and an a-value input. A new x is computed by dividing these two values. Computed x-values are output from PE(1) and also transmitted rightwards.

On cycle, i, PE(i) inputs b_i and $a_{i,1}$, $i > 1$; x_1 is received from the left. The result $c = b_i - a_{i,1}x_1$ is transmitted leftwards. By the time this reaches PE(1), $b_i - \Sigma_{j=1}^{i-1} a_{ij}x_j$ has been computed.

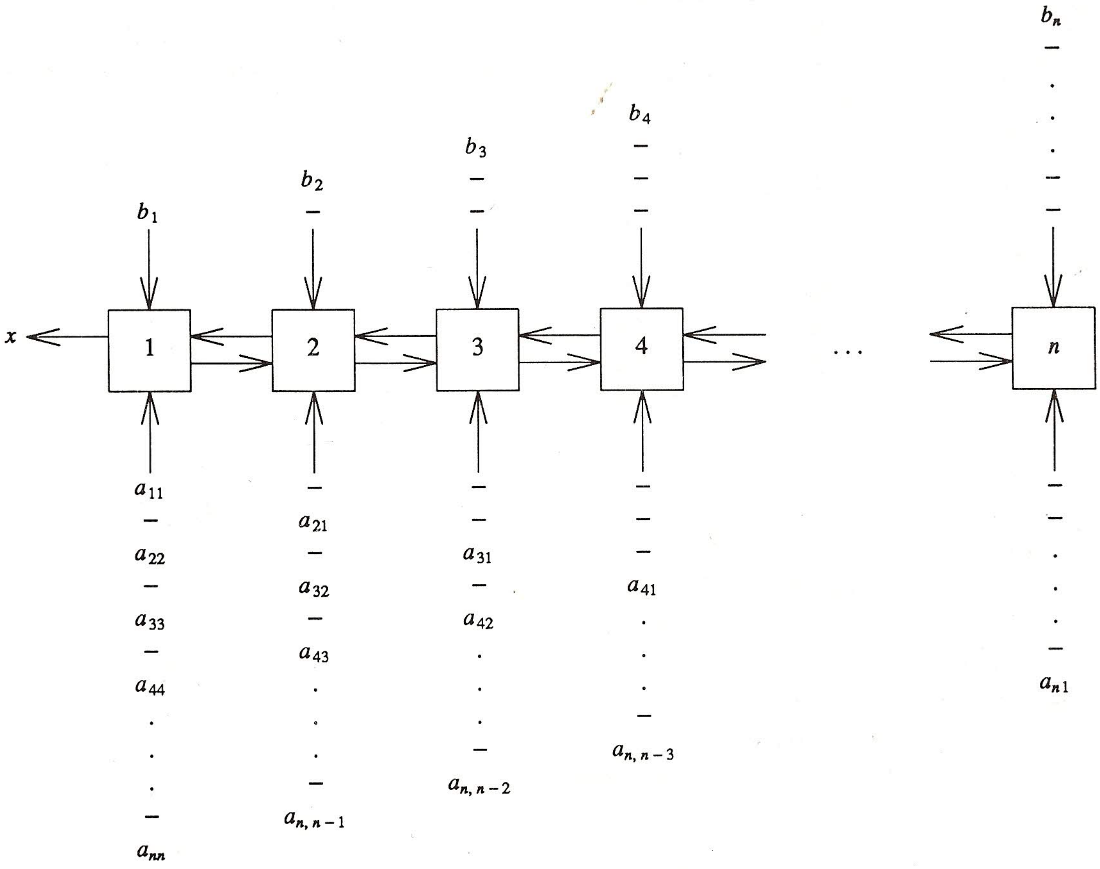


Fig. 4.

Performance

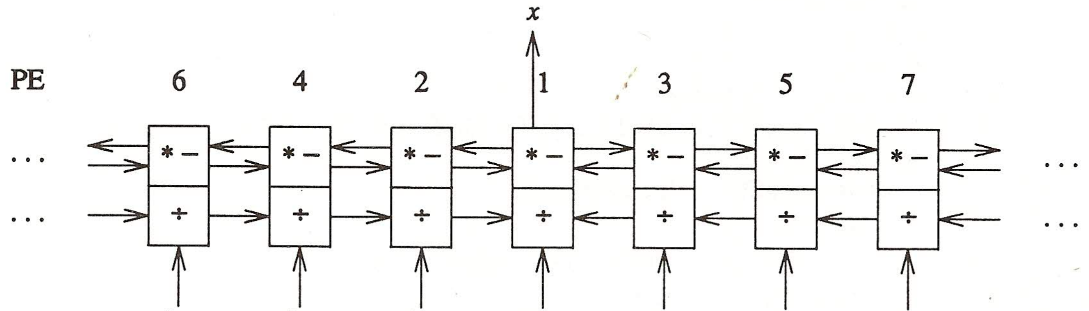
As can be seen, odd PEs perform useful work in odd cycles and even PEs in even cycles. So, adjacent pairs of PEs may be simulated by a single PE to get an equivalent $\lceil\frac{1}{2}n\rceil$ PE architecture. When this is done, the performance figures become $P=\lceil\frac{1}{2}n\rceil$, $B=\lceil\frac{1}{2}n\rceil+2$, $T_C=2n-1$, $T_D=2n$, $R_C\sim 2$, $R_D\sim 2$ and $R\sim 4$.

It is possible to improve the throughput of the bidirectional chain by providing each PE with its own divider that can function in parallel with the rest of the PE. Our design requires n PEs and is as shown in Fig. 5. The output is generated from the middle PE. The PEs to the left of the middle PE compute all but the last term needed for the even x_i's. Similarly, the PEs on the right compute all terms but the last needed for the odd x_i's.

The input data pattern is quite regular. Assume that the PEs are indexed as in Fig. 5 with the middle PE given the index 1. The first input to PE(i) is b_i and the second a_{ii}. Let $A(i, j)$ denote the $(j+2)$th input to PE(i). Then, $A(1, 1)=a_{21}$, $A(1, 2)=a_{32}$, $A(1, 3)=a_{43}$, $A(2, 1)=a_{41}$, $A(5, 1)=0$, etc. Formally, we have

$$A(i, j)=\begin{cases} a_{j+1,j}, & i=1\\ a_{i+2\lceil(j+1-\lceil i/2\rceil)/2\rceil,\, j+1-\lfloor i/2\rfloor}, & i>1\end{cases}$$

where $a_{ik}=0$ for $k<1$.



b_6	b_4	b_2	b_1	b_3	b_5	b_7
a_{66}	a_{44}	a_{22}	a_{11}	a_{33}	a_{55}	a_{77}
—	—	a_{41}	a_{21}	a_{31}	—	—
—	a_{61}	a_{42}	a_{32}	a_{52}	a_{51}	—
a_{81}	a_{62}	a_{63}	a_{43}	a_{53}	a_{72}	a_{71}
a_{82}	a_{83}	a_{64}	a_{54}	a_{74}	a_{73}	a_{92}
$a_{10,3}$	a_{84}	a_{85}	a_{65}	a_{75}	a_{94}	a_{93}
$a_{10,4}$	$a_{10,5}$	a_{86}	a_{76}	a_{96}	a_{95}	$a_{11,4}$
·	$a_{10,6}$	$a_{10,7}$	a_{87}	a_{97}	$a_{11,6}$	$a_{11,5}$
·	·	$a_{10,8}$	a_{98}	$a_{11,8}$	$a_{11,7}$	·
·	·	·	$a_{10,9}$	$a_{11,9}$	·	·
	·	·	$a_{11,10}$	·	·	·
		·	·	·	·	
			·	·		
			·			

Fig. 5.

Each x_i is computed using the formula

$$x_i = \frac{b_i}{a_{ii}} - \frac{a_{i1}}{a_{ii}}x_1 - \frac{a_{i2}}{a_{ii}}x_2 - \cdots - \frac{a_{i,i-1}}{a_{ii}}x_{i-1}. \tag{1}$$

Each term of the form $(a_{ij}/a_{ii})x_j$ is obtained by computing a_{ij}/a_{ii} in one cycle and then multiplying by x_j in the next cycle. It should be noted that computing the x_i's in this way affects the numerical properties of back substitution. In the traditional method (cf. Section 1), only one division per x_i is performed. We do not investigate the effects of numerical errors on the scheme suggested by (1). Further, as pointed out in the introduction, the a_{ii}'s may all be 1 as a result of the factorization scheme used to generate A. In this case the numerical properties are not affected and the divisions by a_{ii} may be eliminated from the following discussion.

The c registers hold the partial sums for (1); the Z registers hold terms of the form a_{ij}/a_{ii}; the D registers hold the divisors a_{ii}; and the e registers hold a single term of the form $(-a_{ij}/a_{ii})x_j$. The e registers are needed as, often, a PE computes two terms of an x_i. The first term is saved in the e register and then combined on the next cycle with the second term and the contents of the c register. The functioning of the systolic system is described formally in Algorithm 3.

Performance

From Algorithm 3 and Fig. 5, we see that $P = n$, $B = n + 1$, $T_{\text{C}} = n + 1$, $T_{\text{D}} = n + 3$, $R_{\text{c}} \sim 2$, $R_{\text{D}} \sim 2$ and $R \sim 4$.

```
zero all registers
$c_i \leftarrow b_i$,    $1 \le i \le n$
$D_i \leftarrow a_{ii}$,    $1 \le i \le n$
$c_i \leftarrow c_i/D_i$,    $1 \le i \le n$
{ Let $U = \{3, 4, 7, 8, 11, 12, \ldots\}$, $V = \{2, 5, 6, 9, 10, 13, 14, \ldots\}$ }
for $j \leftarrow 1$ to $n$ do
  do in parallel    {Input/Output}
    $A_1 \leftarrow a_{j+1,j}$
    $A_i \leftarrow a_{i+2\lceil (j+1-\lceil i/2 \rceil)/2 \rceil,\, j+1-\lfloor i/2 \rfloor}$,    $i > 1$
    $x_2 \leftarrow x_1$,    $j \ge 2$
    $x_i \leftarrow x_{i-2}$,    $i \ge 3,\ j \ge 2$
    output $x_1$,    $j \ge 2$
    if $j$ odd then [$c_1 \leftarrow c_3$, $j \ge 3$; $D_1 \leftarrow D_2$]
                else [$c_1 \leftarrow c_2$; $D_1 \leftarrow D_3$]
    $c_{i-2} \leftarrow c_i$,    $i \ge 4$,  $\lfloor (i+1)/2 \rfloor < j$
    $D_{i-2} \leftarrow D_i$,    $i \ge 4$,  $\lfloor (i-1)/2 \rfloor \le j$
  end
  do in parallel
    $x_1 \leftarrow c_1 - Z_1 * x_1$
    if $j$ odd then [$c_i \leftarrow (c_i + e_i) - Z_i * x_i$,  $i \in V$ and $\lfloor i/2 \rfloor < j$
                     $e_i \leftarrow -Z_i * x_i$,            $i \in U$ and $\lfloor (i+1)/2 \rfloor < j$]
                else [$c_i \leftarrow (c_i + e_i) - Z_i * x_i$,  $i \in U$ and $\lfloor i/2 \rfloor < j$
                     $e_i \leftarrow -Z_i * x_i$,            $i \in V$ and $\lfloor (i+1)/2 \rfloor < j$]
    $Z_i \leftarrow A_i / D_i$,    $1 \le i \le n$
  end
end
output $x_1$
```

Algorithm 3.

The throughput cannot be further improved using formula (1), as to compute x_i, we need to know x_{i-1}. Hence x_i can be computed, at best, one cycle after x_{i-1} has been computed. We can bring T_C down to $\mathrm{o}(\frac{1}{2}n)$ by computing two x_i's each cycle using the formulae

$$\begin{aligned}
x_i &= \frac{b_i}{a_{ii}} - \sum_{j=1}^{i-2} \frac{a_{ij}}{a_{ii}} x_j - \frac{a_{i,i-1}}{a_{ii}} x_{i-1} \\
&= \frac{b_i}{a_{ii}} - \sum_{j=1}^{i-2} \frac{a_{ij}}{a_{ii}} x_j - \frac{a_{i,i-1}}{a_{ii}} \left[\frac{b_{i-1}}{a_{i-1,i-1}} - \sum_{j=1}^{i-2} \frac{a_{i-1,j}}{a_{i-1,i-1}} x_j \right] \\
&= \frac{b_i}{a_{ii}} - \frac{a_{i,i-1}}{a_{ii}a_{i-1,i-1}} b_{i-1} - \sum_{j=1}^{i-2} \left(\frac{a_{ij}}{a_{ii}} - \frac{a_{i,i-1}a_{i-1,j}}{a_{ii}a_{i-1,i-1}} \right) x_j
\end{aligned} \tag{2}$$

for $i > 1$ and odd, and

$$\begin{aligned}
x_i &= \frac{b_i}{a_{ii}} - \frac{a_{i,i-1}}{a_{ii}a_{i-1,i-1}} b_{i-1} - \sum_{j=1}^{i-3} \left(\frac{a_{ij}}{a_{ii}} - \frac{a_{i,i-1}a_{i-1,j}}{a_{ii}a_{i-1,i-1}} \right) x_j \\
&\quad - \left(\frac{a_{i,i-2}}{a_{ii}} - \frac{a_{i,i-1}a_{i-1,i-2}}{a_{ii}a_{i-1,i-1}} \right) x_{i-2} \\
&= \frac{b_i}{a_{ii}} - \frac{a_{i,i-1}}{a_{ii}a_{i-1,i-1}} b_{i-1} - \frac{Z_i b_{i-2}}{a_{i-2,i-2}} - \sum_{j=1}^{i-3} \left(\frac{a_{ij}}{a_{ii}} - \frac{a_{i,i-1}a_{i-1,j}}{a_{ii}a_{i-1,i-1}} - \frac{Z_i a_{i-2,j}}{a_{i-2,i-2}} \right) x_j
\end{aligned} \tag{3}$$

where

$$Z_i = \left(\frac{a_{i,i-2}}{a_{ii}} - \frac{a_{i,i-1}a_{i-1,i-2}}{a_{ii}a_{i-1,i-1}} \right)$$

for $i > 2$ and even.

First, x_1 is computed as $x_1 = b_1/a_{11}$ and x_2 as $x_2 = b_2/a_{22} - a_{21}b_1/a_{11}$. Once x_1 and x_2 have been computed, x_3 and x_4 can be computed. To compute x_3, x_1 is needed and to compute x_4, x_1 is needed. So both x_3 and x_4 can be output one cycle after x_1 and x_2. In the meantime, x_5 and x_6 can start computation using x_1 and x_2. In the next cycle, the computation of x_5 and x_6 can be completed using x_3 from the previous cycle. The VLSI system that incorporates this uses more hardware than Fig. 5 and is quite a bit more complex. We shall not present the details here. We note that the method may be extended to get a T_C of $\mathrm{o}(n/k)$ for any fixed k.

2.5. Broadcast chain with O(n) bandwidth

The architecture and data pattern are shown in Fig. 6. The algorithm used is given in Algorithm 4.

Performance

The performance figures for the broadcast chain are $P = n$, $B = n$, $T_C = n$, $T_D = n + 2$, $R_C \sim 2$, $R_D \sim 2$ and $R \sim 4$.

One should note that each computation of Fig. 6 takes $t_{SUB} + t_{MUL} + t_{DIV}$ time, while in all of our previous designs, a computation required only $\max\{t_{DIV}, t_{MUL} + t_{SUB}\}$ where t_{DIV} is the time to divide, t_{MUL} is the time to multiply, and t_{SUB} is the time to subtract. The performance of the broadcast chain may be improved slightly by overlapping divisions with other oper-

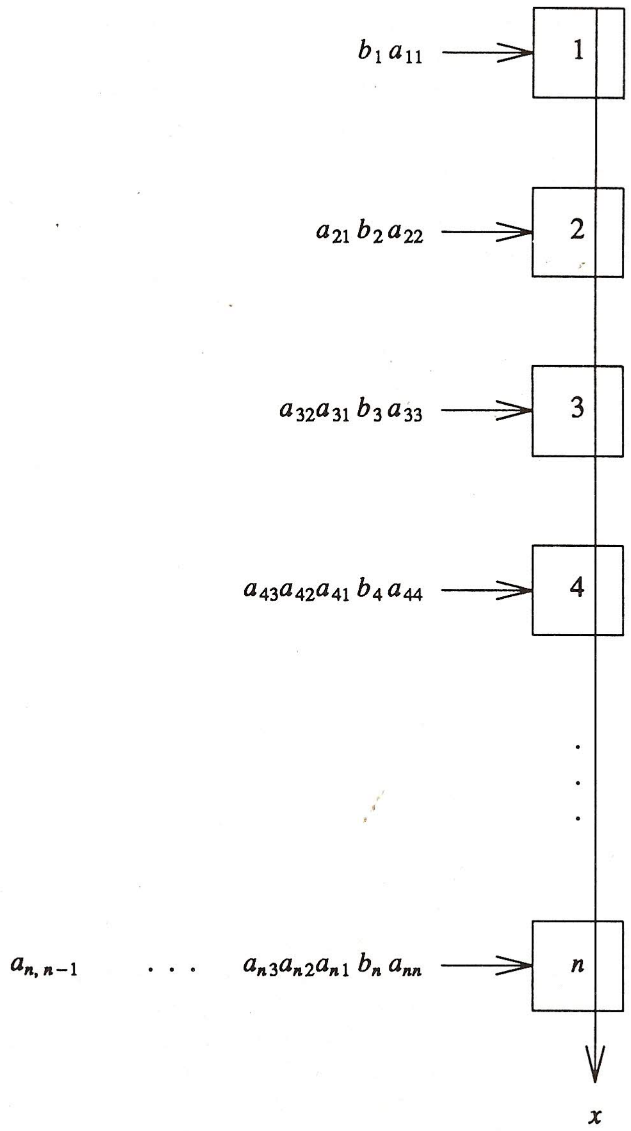

$b_1 a_{11}$
1
$a_{21} b_2 a_{22}$
2
$a_{32} a_{31} b_3 a_{33}$
3
$a_{43} a_{42} a_{41} b_4 a_{44}$
4
$a_{n,n-1}$ . . . $a_{n3} a_{n2} a_{n1} b_n a_{nn}$
n
x

Fig. 6.

```
$D_j \leftarrow a_{jj}$     {input}
$c_j \leftarrow b_j$     {input}
$x_1 \leftarrow c_1 / D_1$
for $i \leftarrow 1$ to $n - 1$ do
   do in parallel
      broadcast $x_i$ to $x_j$,   $j > i$
      output $x_i$
      $A_j \leftarrow a_{ji}$,          $j > i$   {input}
   end
   do in parallel
      $x_j \leftarrow (c_j - A_j * x_j)/D_j$,   $j = i + 1$   {compute $x_{i+1}$}
      $c_j \leftarrow c_j - A_j * x_j$,       $j > i + 1$
   end
end
output $x_n$
```

Algorithm 4.

ations. The new design is as in Fig. 7. Equation (1) is used to compute the x_i's. The dividers are loaded with a_{ii}. Each input to PE(i) is divided by a_{ii} at the divider and then transmitted to the remainder of the PE. The divisions take place in parallel with a multiply and subtract operation in the same PE. As a result of this, each computation step takes $\max\{t_{\mathrm{DIV}}, t_{\mathrm{SUB}} + t_{\mathrm{MUL}}\}$ when Fig. 7 is used rather than $t_{\mathrm{DIV}} + t_{\mathrm{SUB}} + t_{\mathrm{MUL}}$. The performance figures using Fig. 7 are $P = n$, $B = n$, $T_{\mathrm{C}} = n + 1$, $T_{\mathrm{D}} = n + 3$, $R_{\mathrm{C}} \sim 2$, $R_{\mathrm{D}} \sim 2$ and $R \sim 4$.

The R value of both preceding designs may be reduced to $\sim \frac{9}{4}$ by using half as many PEs. For example, the design of Fig. 6 translates into that of Fig. 8. The $\frac{1}{2}n$-PE broadcast chain is first used to compute $x_1, x_2, \ldots, x_{n/2}$ in exactly the same way as Fig. 6 is used when A is an $\frac{1}{2}n \times \frac{1}{2}n$ matrix. Next, the remaining $\frac{1}{2}n$ x's are computed by having the $\frac{1}{2}n$ PEs behave like the last $\frac{1}{2}n$ PEs of Fig. 6 with the required x_i's being re-input and broadcast. To compute the first $\frac{1}{2}n$ x's, $\frac{1}{2}n$ compute steps and $\frac{1}{2}n + 2$ data move steps are needed. For the remaining $\frac{1}{2}n$ x's, an additional n compute steps and $n + 2$ data move steps are needed. Hence, $P = \frac{1}{2}n$, $B = \frac{1}{2}n + 1$,

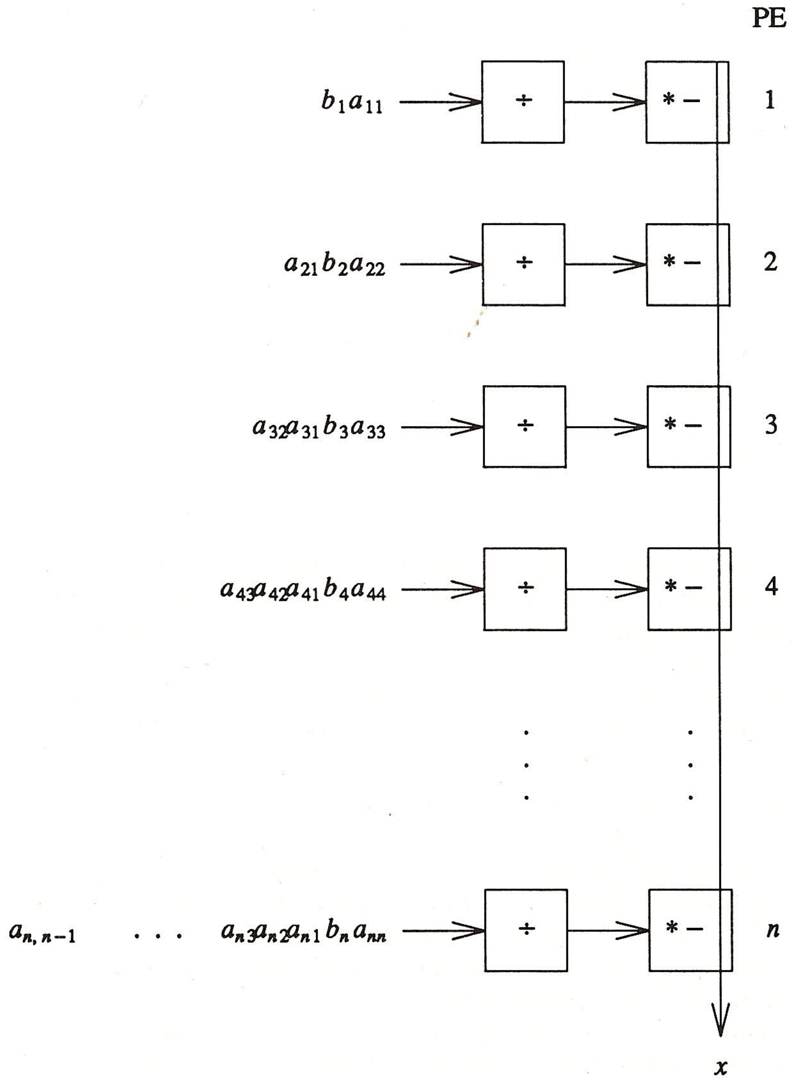

PE
$b_1 a_{11}$
÷
$* -$
1
$a_{21} b_2 a_{22}$
÷
$* -$
2
$a_{32} a_{31} b_3 a_{33}$
÷
$* -$
3
$a_{43} a_{42} a_{41} b_4 a_{44}$
÷
$* -$
4
$a_{n,n-1}$ . . . $a_{n3} a_{n2} a_{n1} b_n a_{nn}$
÷
$* -$
n
x

Fig. 7.

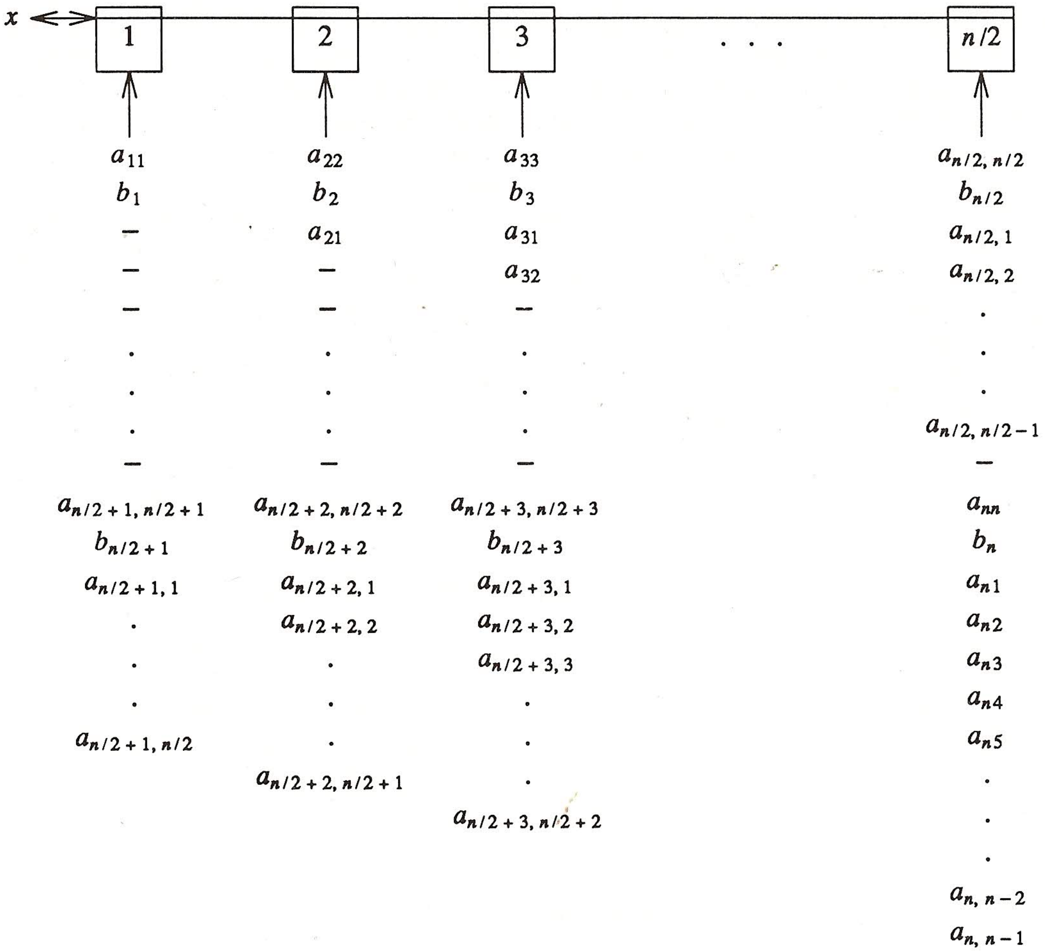


Fig. 8.

$T_C = \frac{3}{2}n$, $T_D = \frac{3}{2}n + 4$, $R_C \sim \frac{3}{2}$, $R_D \sim \frac{3}{2}$ and $R \sim \frac{9}{4}$. This reduction in the R value has been obtained at the expense of T_C and T_D which are now both 50% larger than before.

This idea may be extended to the case of n/k PEs for k any constant. For example when $k = 4$, we get $P = \frac{1}{4}n$, $B = \frac{1}{4}n + 1$, $T_C = \frac{5}{2}n$, $T_D = \frac{5}{2}n + 8$, $R_C \sim \frac{5}{4}$, $R_D \sim \frac{5}{4}$ and $R \sim \frac{25}{16}$. A further reduction in R has occurred at the expense of T_C and T_D! Certainly, when $k = n$, $P = 1$ and the scheme becomes the normal one processor scheme with $R = 1$.

2.6. o(n) throughput with unidirectional data flow

The unidirectional chain of Fig. 2 cannot be modified to produce the n x_i's in o(n) time. This is because it takes o(n) time for x_1 to reach PE(n) and another o(n) time to compute x_n using one PE. While we have seen several architectures that provide for o(n) computation of the x_i's, these either employed a broadcast capability or required data to flow in two directions. Neither of these capabilities is necessary for this.

We may compute the x_i's in o(n) time using $2n - 2$ PEs as in Fig. 9. One PE is assigned to the computation of x_1 and another to that of x_2. Each of the remaining x_i's is computed using two PEs. The upper PE of each such pair computes the terms involving x_i for i odd while the

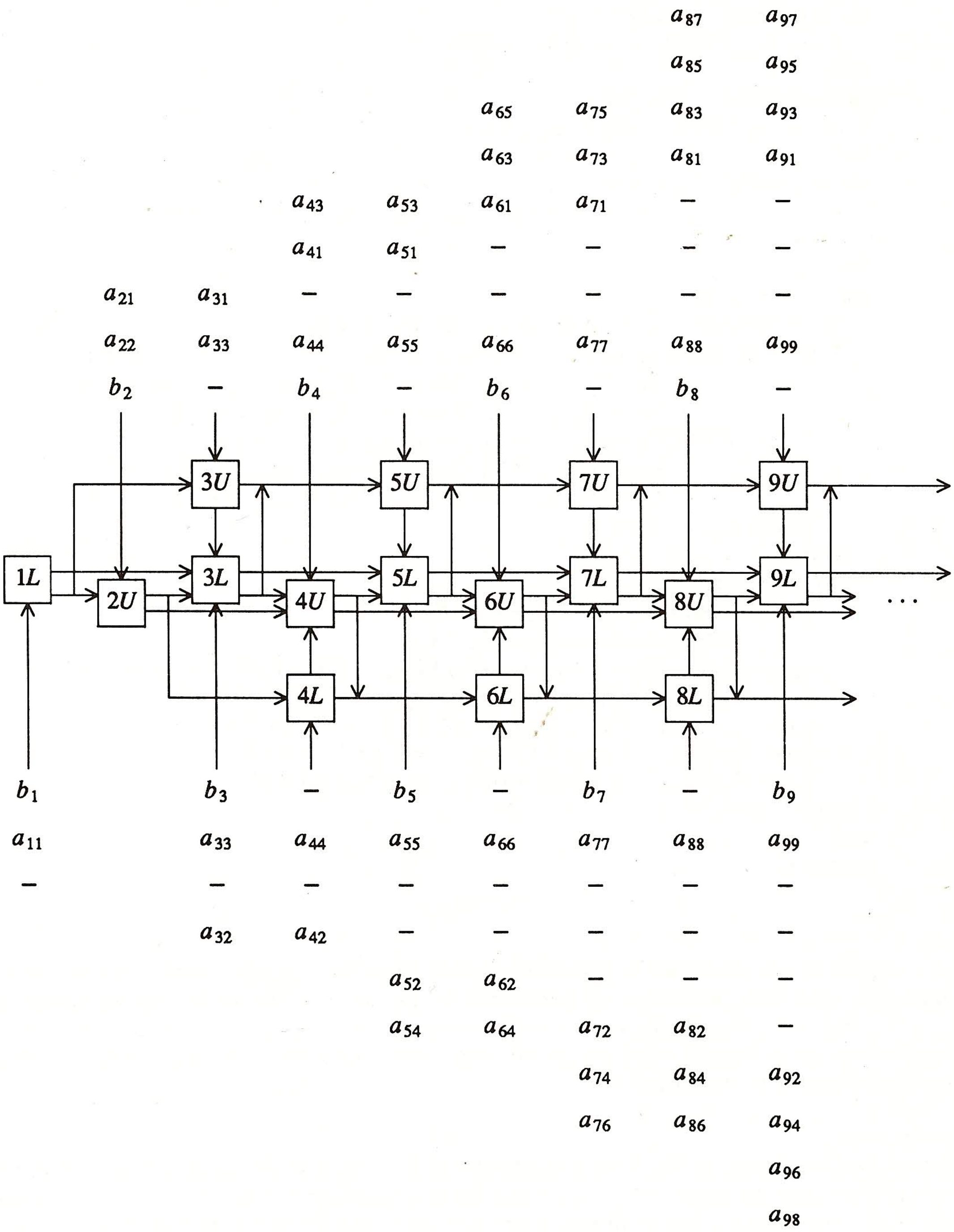

a_{87} a_{97}
a_{85} a_{95}
a_{65} a_{75} a_{83} a_{93}
a_{63} a_{73} a_{81} a_{91}
a_{43} a_{53} a_{61} a_{71} – –
a_{41} a_{51} – – – –
a_{21} a_{31} – – – – – –
a_{22} a_{33} a_{44} a_{55} a_{66} a_{77} a_{88} a_{99}
b_2 – b_4 – b_6 – b_8 –
3U 5U 7U 9U
1L 3L 5L 7L 9L
2U 4U 6U 8U
...
4L 6L 8L
b_1 b_3 – b_5 – b_7 – b_9
a_{11} a_{33} a_{44} a_{55} a_{66} a_{77} a_{88} a_{99}
– – – – – – – –
a_{32} a_{42} – – – – –
a_{52} a_{62} – – –
a_{54} a_{64} a_{72} a_{82} –
a_{74} a_{84} a_{92}
a_{76} a_{86} a_{94}
a_{96}
a_{98}

Fig. 9.

lower PE computes the even terms. Predivision of the a_{ij}'s by a_{ii} as suggested by formula (1) is done by each PE. The upper PE in the pair for x_i computes $U_i = b_i/a_{ii} - \Sigma_{1 \leqslant j < i, j \text{ odd}}(a_{ij}/a_{ii})x_j$ when i is even and $U_i = \Sigma_{1 \leqslant j < i, j \text{ odd}}(a_{ij}/a_{ii})x_j$ when i is odd. The lower PE computes $L_i = b_i/a_{ii} - \Sigma_{1 \leqslant j < i, j \text{ even}}(a_{ij}/a_{ii})x_j$ when i is odd and $L_i = \Sigma_{1 \leqslant j < i, j \text{ even}}(a_{ij}/a_{ii})x_j$ when i is

```
do in parallel   $1 \le i \le n$
  $c_{iU} \leftarrow b_i$,  $i$ even;    $c_{iL} \leftarrow b_i$,  $i$ odd
end
do in parallel   $1 \le i \le n$
  $D_{iU} \leftarrow a_{ii}$,  $i \ne 1$;   $D_{iL} \leftarrow a_{ii}$,  $i \ne 2$
end
do in parallel   $1 \le i \le n$
  $c_{iU} \leftarrow c_{iU}/D_{iU}$,  $c_{iL} \leftarrow 0$,  $i$ even;     $c_{iL} \leftarrow c_{iL}/D_{iL}$,  $c_{iU} \leftarrow 0$,  $i$ odd
  $x_{1L} \leftarrow Z_{1L} \leftarrow 0$
end
for $j \leftarrow 1$ to $n$ do
  do in parallel   $\{l = j + k,\ m = j - 2\lfloor (k-1)/2 \rfloor\}$
    if $j$ odd then [$A_{lU} \leftarrow a_{l,m}$ , $1 \le k \le j + 1$;  $A_{l+1,L} \leftarrow a_{l+1,m-1}$ , $1 \le k \le j - 1$
                     $X_{lL} \leftarrow x_{j-1}$ , $j \ne 1, 0 \le k \le 1$;  $X_{lL} \leftarrow X_{m-2,L}$ , $2 \le k \le j - 2$
                     $X_{lU} \leftarrow X_{m-2,U}$ , $1 \le k \le j - 1$;  $Y_{jL} \leftarrow c_{jU}$ , $j \ne 1$]
               else [$A_{lL} \leftarrow a_{l,m}$ ,  $A_{l+1,U} \leftarrow a_{l+1,m-1}$ ,  $1 \le k \le j$
                     $X_{lU} \leftarrow x_{j-1}$ , $0 \le k \le 1$;  $X_{lU} \leftarrow X_{m-2,U}$ , $2 \le k \le j - 1$
                     $X_{lL} \leftarrow X_{m-2,L}$ , $1 \le k \le j - 2$;  $Y_{jU} \leftarrow c_{jL}$ , $j \ne 2$]
    output $x_{j-1}$,   $j \ne 1$
  end
  do in parallel
    if $j$ odd then $x_j \leftarrow (c_{jL} - Y_{jL}) - Z_{jL} * X_{jL}$
               else $x_j \leftarrow (c_{jU} - Y_{jU}) - Z_{jU} * X_{jU}$
    $c_{mU} \leftarrow c_{mU} - Z_{mU} * X_{mU}$,  $c_{mL} \leftarrow c_{mL} + Z_{mL} * X_{mL}$,  $1 \le k \le \lceil j/2 \rceil - 1$, $m = 2\lceil (j + k)/2 \rceil$
    $c_{mU} \leftarrow c_{mU} + Z_{mU} * X_{mU}$ ,  $1 \le k \le \lfloor j/2 \rfloor$, $m = 2\lceil j/2 + k \rceil - 1$
    $c_{mL} \leftarrow c_{mL} - Z_{mL} * X_{mL}$ ,  $1 \le k \le \lfloor j/2 \rfloor - 1$, $m = 2\lceil j/2 + k \rceil - 1$
    $Z_{lU} \leftarrow A_{lU}/D_{lU}$,   $1 \le k \le 2\lceil j/2 \rceil$,  $l = 2\lfloor j/2 \rfloor + k + 1$
    $Z_{mL} \leftarrow A_{mL}/D_{mL}$,   $1 \le k \le 2\lfloor j/2 \rfloor$,  $m = 2\lceil j/2 \rceil + k$
  end
end
output $x_n$
```

Algorithm 5.

even. L_i and U_i are combined in the lower (upper) PE when i is odd (even). This involves a subtraction and this subtraction is performed in the same cycle as the multiplication of $a_{i,i-1}/a_{ii}$ and x_{i-1}. Hence, the cycle time is $\max\{t_{\text{DIV}}, \max\{t_{\text{MUL}}, t_{\text{SUB}}\} + t_{\text{SUB}}\}$ (assuming the add time to be roughly equal to the subtract time). The working of the systolic system is described formally in Algorithm 5.

Performance

As drawn, the design of Fig. 9 has a bandwidth of $2n$. This is easily reduced to n by delaying the inputs of both the upper row of a_{ii}'s and the row following the lower row on a_{ii}'s by one time unit. This is equivalent to inserting one row of dashes ("-") before row 2 of the upper input and after row 2 of the lower input. The performance figures for this design are $P = 2n - 2$, $B = n$, $T_{\text{C}} = n + 1$, $T_{\text{D}} = n + 4$, $R_{\text{C}} \sim 4$, $R_{\text{D}} \sim 2$ and $R \sim 8$.

2.7. O(1) bandwidth designs

Horowitz [2] presents an n-PE chain with O(1) bandwidth for the back substitution problem. While his original design employs $2n$ PEs, n of these are used solely for input and may be

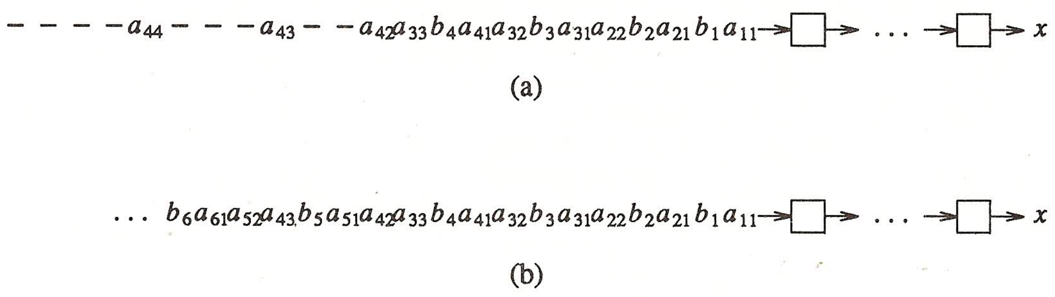

$- - - - a_{44} - - - a_{43} - - a_{42}a_{33}b_4a_{41}a_{32}b_3a_{31}a_{22}b_2a_{21}b_1a_{11}$
...
x
(a)
$\ldots b_6a_{61}a_{52}a_{43}b_5a_{51}a_{42}a_{33}b_4a_{41}a_{32}b_3a_{31}a_{22}b_2a_{21}b_1a_{11}$
...
x
(b)

Fig. 10.

eliminated. Fig. 10(a) shows the input pattern for the case $n = 4$ while Fig. 10(b) shows the general input pattern. PE(i) computes x_i as well as all terms that involve x_i. Hence PE(i) computes $[b_j - \Sigma_{k<i}a_{jk}x_k] - a_{ji}x_i$ for $j > i$. The term $[b_j - \Sigma_{k<i}a_{jk}x_k]$ is transmitted to PE(i) from PE($i-1$) and PE(i) in turn transmits $[b_j - \Sigma_{k<i}a_{jk}x_k] - a_{ji}x_i$ to PE($i+1$). The formal algorithm is given in Algorithm 6. The performance figures for the design of [2] are $P = n$, $B = 1$, $T_C = 2n - 1$, $T_D = n(n+3) - 1$, $R_C \sim 4$, $R_D \sim 2$ and $R \sim 8$.

Improved performance can be obtained using a somewhat simpler algorithm, Algorithm 7, and the data pattern of Fig. 11. This time the chain is a bidirectional chain. The number of PEs used is $n - 1$. The performance figures for this design are $P = n - 1$, $B = 1$, $T_C = 2n - 1$, $T_D = \frac{1}{2}n(n+5)$, $R_C \sim 4$, $R_D \sim 1$ and $R \sim 4$.

Further improvement may be obtained by simulating the $\lceil\frac{1}{2}n\rceil$ PE version of the bidirectional chain algorithm of Section 2.4. The simulation involves sending all the input needed for the next computation rightwards from PE(1) and then performing the computation in all PEs simultaneously. P, T_C and R_C are unchanged. However, now, $B = 1$, $T_D = \frac{1}{2}n(n+5)$, $R_D = 1$ and $R \sim 2$.

An O(1) bandwidth bidirectional chain with improved throughput can also be obtained by simulating the improved throughput design of Section 2.4. The input for each computation step

```
for $i \leftarrow 1$ to $2n-1$ do
   for $k \leftarrow 1$ to $\lceil i/2 \rceil$ do
      do in parallel
         $A_1 \leftarrow a_{\lfloor i/2 \rfloor + k,\ \lceil i/2 \rceil - k + 1}$,     {input}
         $A_j \leftarrow A_{j-1}$,                  $2 \le j \le k$
      end
   end
   do in parallel
      if $i \le n$ then $c_1 \leftarrow b_i$,        {input}
      $c_j \leftarrow c_{j-1}$,        $2 \le j \le \lceil i/2 \rceil$
   end
   do in parallel
      if $i$ odd then [ $x_{\lceil i/2 \rceil} \leftarrow c_{\lceil i/2 \rceil} / A_{\lceil i/2 \rceil}$
                        $c_j \leftarrow c_j - A_j * x_j$,   $j \ne \lceil i/2 \rceil$ ]
               else [ $c_j \leftarrow c_j - A_j * x_j$,   $1 \le j \le \lceil i/2 \rceil$ ]
   end
end
output $x_1, \ldots, x_n$
```

Algorithm 6.

```
$c_1 \leftarrow b_1$          {input}
$A_1 \leftarrow a_{11}$        {input}
$x_1 \leftarrow c_1 / A_1$
output $x_1$
for $i \leftarrow 2$ to $n$ do
    do in parallel
        $c_1 \leftarrow b_{n-i+2}$,   {input}
        $c_j \leftarrow c_{j-1}$,     $2 \le j < n$
    end
end
for $i \leftarrow 1$ to $n-1$ do
    for $k \leftarrow 1$ to $n-i$ do
        do in parallel
            $A_1 \leftarrow a_{n-k+1,i}$,    {input}
            $A_j \leftarrow A_{j-1}$,        $2 \le j \le n-i$
            $x_j \leftarrow x_{j-1}$,        $2 \le j \le n-i$
        end
    end
    $c_j \leftarrow c_j - A_j * x_j$,    $2 \le j \le n-i$
    $A_1 \leftarrow a_{i+1,i+1}$,        {input}
    $x_1 \leftarrow c_1 / A_1$
    do in parallel
        $c_j \leftarrow c_{j+1}$,   $1 \le j \le n-i-1$
        output $x_1$
    end
end
```

Algorithm 7.

is provided through the middle PE. When this is done, we get $P = n$, $B = 1$, $T_C = n + 2$, $T_D = \frac{1}{2}n(n+5)$, $R_C \sim 2$, $R_D = 1$ and $R \sim 2$.

2.8. Summary

The performance figures of the various VLSI architectures for the back substitution problem are summarized in Table 1. Our designs are the first VLSI systems that require less than o(2n) computational steps. This has been accomplished without sacrificing on R. In fact, the R value of our O(1) bandwidth design for the case of $T_C = \text{o}(n)$ is half that of the best designs with $T_C = \text{o}(2n)$.

Finally, we note that the comparisons among the different designs are not entirely fair as some designs require each PE to have a multiplier, a divider, and a subtracter; while other

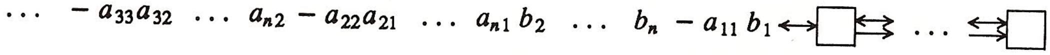

... $-a_{33}a_{32}$ ... a_{n2} $-a_{22}a_{21}$ ... a_{n1} b_2 ... b_n $-a_{11}$ b_1
...

Fig. 11.

Table 1

Performance	Architecture					
	Chain with O(n) bandwidth	Ring with O(n) bandwidth	Bidirectional chain with O(n) bandwidth		Broadcast chain with O(n) bandwidth	
			[7]	Our	n PEs	$n/2$ PEs
P	n	$\lceil n/2 \rceil$	$\lceil n/2 \rceil$	n	n	$n/2$
B	$\lceil n/2 \rceil + 1$	$\lceil n/2 \rceil + 1$	$\lceil n/2 \rceil + 2$	$n+1$	n	$n/2$
T_C	$2n-1$	$2n-1$	$2n-1$	$n+1$	$n+1$	$3n/2$
T_D	$2n$	$2n$	$2n$	$n+3$	$n+3$	$3n/2$
R_C	4	2	2	2	2	$3/2$
R_D	2	2	2	2	2	$3/2$
R	8	4	4	4	4	$9/4$

Performance	Architecture				
	o(n) Throughput unidirectional data flow	O(1) bandwidth designs			
		Unidirectional chain [2]	Bidirectional chain		
			Our	Improved performance	Improved throughput
P	$2n-2$	n	$n-1$	$\lceil n/2 \rceil$	n
B	n	1	1	1	1
T_C	$n+1$	$2n-1$	$2n-1$	$2n-1$	$n+2$
T_D	$n+4$	n^2+3n-1	$n(n+5)/2$	$n(n+5)/2$	$n(n+5)/2$
R_C	4	4	4	2	2
R_D	2	2	1	1	1
R	8	8	4	2	2

$C = n(n+1)/2$, $D = n(n+5)/2$.

designs require two kinds of PEs: one with a multiplier and a subtracter and the other with just a divider.

References

[1] K.H. Cheng and S. Sahni, VLSI architectures for matrix multiplication, in: *Proc. Foundations of Software Technology and Theoretical Computer Science*, Lecture Notes in Computer Science **206** (Springer, Berlin, 1985).

[2] E. Horowitz, VLSI architectures for matrix computations, in: *Proc. IEEE International Conference on Parallel Processing* (1979) 124–127.

[3] K.H. Huang and J.A. Abraham, Efficient parallel algorithms for processor arrays, in: *Proc. IEEE International Conference on Parallel Processing* (1982) 271–279.

[4] H.T. Kung, Let's design algorithms for VLSI systems, in: *Proc. CALTECH Conference on VLSI* (1979) 65–90.

[5] H.T. Kung, A listing of systolic papers, Department of Computer Science, Carnegie-Mellon University, 1984.

[6] H.T. Kung and M. Lam, Wafer scale integration and two level pipelined implementations of systolic arrays, *J. Parallel Distrib. Process.* **1** (1) (1984).

[7] H.T. Kung and C.E. Leiserson, Systolic arrays for VLSI, Department of Computer Science, Carnegie-Mellon University, 1978.

[8] C.E. Leiserson, *Area-Efficient VLSI Computation* (MIT Press, Cambridge, MA, 1983).