

VLSI ARCHITECTURES FOR BACK SUBSTITUTION*

Kam Hoi CHENG⁺ and Sartaj SAHNI
 Computer Science Department, University of Minnesota
 Minneapolis, Minnesota 55455, U.S.A.

VLSI architectures involving unidirectional, bidirectional and broadcast chains as well as unidirectional rings are examined for the back substitution problem. We develop designs with superior performance to earlier designs.

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 {6}. 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. x is obtained using the formula:

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

The x_i s are computed in the order x_1, x_2, \dots, x_n . VLSI architectures for this problem have been proposed earlier in {2}, {4}, {7}, and {8}.

The design of {2} employs a unidirectional chain of processors. The data flow is left to right. The design of Kung and Leiserson ({4}, {5} and {8}) employs a bidirectional chain. Here, data is permitted to flow both from left to right and from right to left. In {7}, a ring architecture. Data can flow in only a single direction (either clockwise or counterclockwise) around the ring.

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. 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 a host processor via 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

$$R_D = B * T_D / D$$

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

+ Author's current address is: Computer Science Department, University of Houston, Houston, Texas, USA.

measures the effectiveness with which the bandwidth B has been used. Clearly, $R_D \geq 1$ for every VLSI design.

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 $R_C \geq 1$ for every VLSI design.

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 {3}. 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: $C = n(n+1)/2 \sim n^2/2$,
 $D = n(n+5)/2 \sim n^2/2$.

2.2. Chain with $O(n)$ Bandwidth

An n PE chain may be used as in Figure 1 (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 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 Figure 1 and Algorithm 1, we see that maximum input/output occurs when $i = n$. At this time, a bandwidth of $\lfloor n/2 \rfloor + 1$ is needed. The performance figures for this scheme are $P = n$, $B = \lfloor n/2 \rfloor + 1$, $T_C = 2n - 1$, $T_D = 2n$, $R_C \sim 4$, $R_D \sim 2$ and $R \sim 8$.

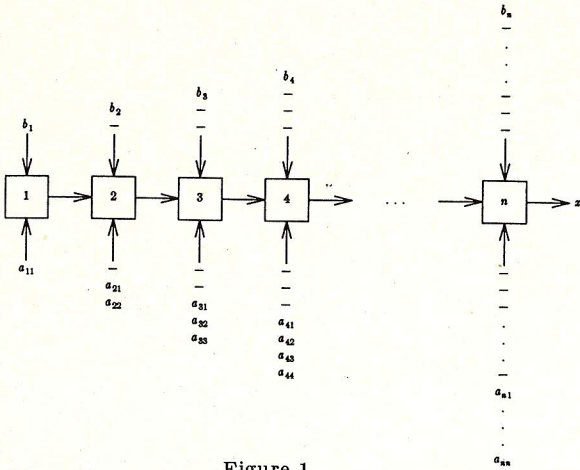


Figure 1

```

for i ← 1 to 2n - 1 do
  do in parallel {Input/Output}
    if i ≤ n then c_i ← b_i
    else output x_n
    A_j ← a_j, i - j + 1, (i + 1)/2 ≤ j ≤ min {i, n}
    x_j ← x_{j-1}, 2 ≤ j ≤ n
  end
  do in parallel
    if i odd then [ x_{[i/2]} ← c_{[i/2]} / A_{[i/2]}
                  c_j ← c_j - A_j * x_j, j ≠ [i/2] ]
    else [ c_j ← c_j - A_j * x_j, 1 ≤ j ≤ n ]
  end
end
output x_n
    
```

Algorithm 1

2.3. Ring with O(n) Bandwidth

An examination of Algorithm 1 reveals that at most $\lceil n/2 \rceil$ PEs do useful work at any time. Hence, we may double up the use of the first $\lceil n/2 \rceil$ PEs by requiring PE(i) to compute both x_i and $x_{i + \lceil n/2 \rceil}$. In case n is odd, PE($\lceil n/2 \rceil$) 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 Figure 2 and the algorithm executed by each PE is given in Algorithm 2. Once again, correctness is easily established. The performance figures are $P = \lceil n/2 \rceil$, $B = \lceil n/2 \rceil + 1$, $T_C = 2n - 1$, $T_D = 2n$, $R_C \sim 2$, $R_D \sim 2$ and $R \sim 4$. So even though only half as many PEs are used, T_C and T_D are unchanged!

```

for i ← 1 to 2n - 1 do
  do in parallel {Input/Output}
    if i ≤ n then c_{(i-1) mod n/2 + 1} ← b_i
    A_j ← a_j, i - j + 1, A_j ← a_j + n/2, i - j - n/2 + 1,
          (i+1)/2 ≤ j + n/2 ≤ min {i, n}
    case
      1 < i ≤ n/2 : x_j ← x_{j-1}, 2 ≤ j < n/2
      n/2 < i ≤ n : x_j ← x_{j-1}, 2 ≤ j < n/2
                   x_1 ← x_{n/2}
      i > n       : x_j ← x_{j-1}, 2 ≤ j < n/2
                   output x_{n/2}
    endcase
  end
  do in parallel
    if i odd then [ k ← ([i/2] - 1) mod n/2 + 1
                  x_k ← c_k / A_k
                  c_j ← c_j - A_j * x_j, j ≠ k ]
    else [ c_j ← c_j - A_j * x_j, 1 ≤ j ≤ n/2 ]
  end
end
output x_{n/2}
    
```

Algorithm 2

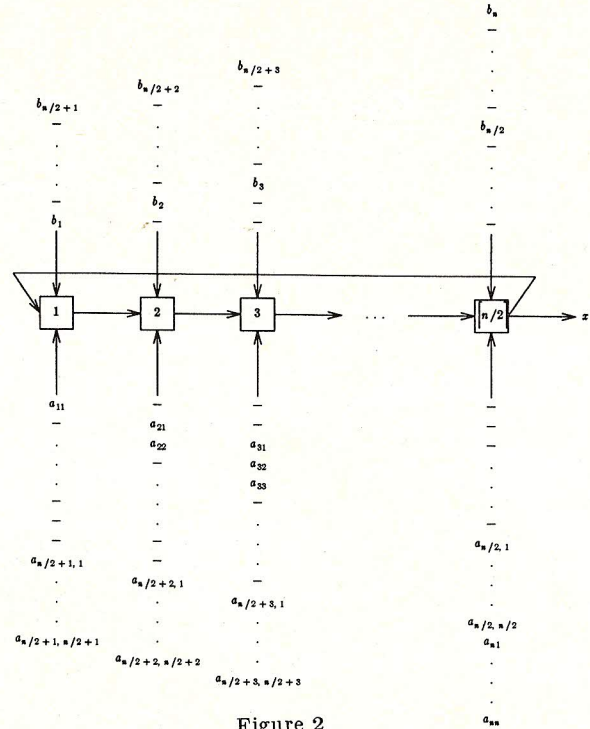


Figure 2

2.4. Bidirectional Chain With O(n) Bandwidth

Kung {4} has proposed an n PE bidirectional chain architecture for the back substitution problem. The performance figures for this architecture are: $P = \lceil n/2 \rceil$, $B = \lceil n/2 \rceil + 2$, $T_C = 2n - 1$, $T_D = 2n$, $R_C \sim 2$, $R_D \sim 2$ 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 Figure 3. 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.

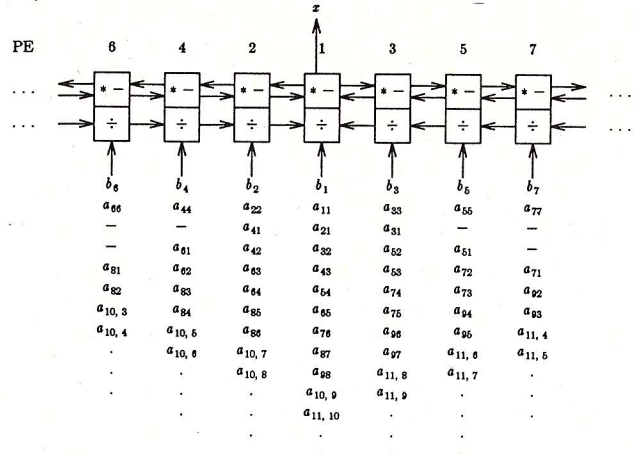


Figure 3

The input data pattern is quite regular. Assume that the PEs are indexed as in Figure 3 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:

