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:

\[
z_i = \left(b_i - \sum_{j=1}^{i-1} a_{ij} z_j \right) / a_{ii}, \quad 1 \leq i \leq n
\]

The \( z_i \) are computed in the order \( z_1, z_2, ..., z_n \). 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 transmission 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 the VLSI system. The ratio \( R_D = B \times T_D / D \) 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

\[
R_C = P \times T_C / C
\]

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 shall regard \( R_C \) 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 \( R = R_C \times R_D \). 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 \( x_j \) 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 \( [n/2] + 1 \) is needed. The performance figures for this scheme are \( P = n \), \( B = [n/2] + 1 \), \( T_C = 2n - 1 \), \( T_D = 2n \), \( R_C \sim 4 \), \( R_D \sim 2 \) and \( R \sim \frac{1}{2} \).
for $i = 1$ to $2n - 1$ do
  do in parallel {Input/Output}
    if $i \leq n$ then $c_i \leftarrow b_i$
    else output $x_{n/2}$
    $A_j \leftarrow a_{j-i-j+2^n/2}, 2 \leq j \leq n$
    $z_j \leftarrow z_{j-n/2}$ if $j \neq \lfloor i/2 \rfloor$
  end
  do in parallel
    if $i$ odd then $\lfloor i/2 \rfloor \times A_{\lfloor i/2 \rfloor} / A_{\lfloor i/2 \rfloor}$
    $c_j \leftarrow c_j - A_j \times z_{\lfloor i/2 \rfloor}, j \neq \lfloor i/2 \rfloor$
    else $c_j \leftarrow c_j - A_j \times z_{j-n/2}, 1 \leq j \leq n$
  end
  output $x_n$

Algorithm 1

2.3. Ring with $O(n)$ Bandwidth

An examination of Algorithm 1 reveals that at most $\lfloor n/2 \rfloor$ PEs do useful work at any time. Hence, we may double up the use of the first $\lfloor n/2 \rfloor$ PEs by requiring PE$(i)$ to compute both $x_i$ and $x_{i+n/2}$. In case $n$ is odd, PE$(\lfloor n/2 \rfloor)$ computes only $x_{n/2}$ 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 $P = \lfloor n/2 \rfloor$, $B = \lfloor n/2 \rfloor + 1$, $T_C = 2n - 1$, $T_D = 2n$, $R_C \sim 2$, $R_D \sim 2$ 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 $i \leq n$ then $c_{i-n/2} \leftarrow b_i$
    $A_j \leftarrow a_{j-i-j+2^n/2}, 2 \leq j \leq n$
    $i-n/2 \leq j+n/2 \leq \min \{i, n/2 \}$
    case
    $1 \leq i \leq n/2$ : $z_j \leftarrow z_{j-n/2}, 2 \leq j \leq n/2$
    $n/2 < i \leq n$ : $z_j \leftarrow z_{j-i-1}, 2 \leq j \leq n/2$
    $z_{i-n/2} \leftarrow x_{i-n/2}$
    endcase
  enddo in parallel
  if $i$ odd then $k \leftarrow \lfloor i/2 \rfloor + 1$ mod $n/2+1$
  $x_k \leftarrow x_{k-i-1}$
  $c_j \leftarrow c_j - A_j \times z_j, j = k$
  else $c_j \leftarrow c_j - A_j \times z_{j-n/2}, 1 \leq j \leq n/2$
  end
  output $x_{n/2}$

Algorithm 2

2.4. Bidirectional Chain With $O(n)$ Bandwidth

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

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 $a_{i-n/2}$. Let $A(i,j)$ denote the $(j-i)$th input to PE$(i)$. Then, $A(1,1) = a_{2n}$, $A(1,2) = a_{2n-2}$, $A(1,3) = a_{2n-4}$, $A(2,1) = a_{4n}$, $A(5,1) = 0$, etc. Formally, we have:
\[ A(i, j) = \begin{cases} a_{j + 1, i} + 1, & i = 1 \\ a_i + 2 \left( (j + 1 - \lceil j/2 \rceil)/2 \right), & j + 1 - \lceil j/2 \rceil, \ i > 1 \end{cases} \]

where \( a_{ii} = 0 \) for \( k < 1 \).

Each \( x_i \) is computed using the formula

\[ x_i = a_{ii} \cdot a_{i1} \cdot x_1 \cdot a_{i2} \cdot x_2 \cdots a_{i,i-1} \cdot x_{i-1} \]

Each term of the form \( a_{ij} \cdot x_j \) is obtained by computing \( a_{ij} / a_{ii} \) in one cycle and then multiplying by \( x_j \) in the next cycle.

The \( c \) registers hold the partial sums for (1); the \( Z \) registers hold the form of \( a_{ij} / a_{ii} \); the \( D \) registers hold the divisors \( a_{ii} \); and the \( e \) registers hold a single term of the form \( a_{ij} \cdot x_j \). The \( e \) registers are needed, 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.

zero all registers
\[ c_i = b_i, \quad 1 \leq i \leq n \]
\[ D_i = a_{ii}, \quad 1 \leq i \leq n \]
\[ \{ \text{Let } U = \{1, 2, 4, 7, 8, 11, \ldots \} \} \]
for \( j = 1 \) to \( n \) do
do in parallel (Input/Output)
\[ A_1 = a_{11} \quad + \quad a_{1, j+1} \quad + \quad a_{1, i+1} \]
\[ i > 1 \]
\[ A_i = a_{ii} \cdot a_{i1} \cdot x_1 \cdot a_{i2} \cdot x_2 \cdots a_{i,i-1} \cdot x_{i-1} \]
\[ z_x = z_x \quad + \quad \{a_{j+1} \cdot x_j \} \quad + \quad z_x \quad + \quad \{a_{i+1} \cdot x_i \} \quad + \quad \{a_{i,i-1} \cdot x_{i-1} \} \]
output \( x_i \)
If \( j \) odd then \( c_i \quad = \quad c_i \quad \quad + \quad \{a_{j+1} \cdot x_j \} \quad + \quad \{a_{i+1} \cdot x_i \} \quad + \quad \{a_{i,i-1} \cdot x_{i-1} \} \]
\[ D_i = D_i \quad + \quad D_i \quad + \quad D_i \]
else \( c_i \quad = \quad c_i \quad \quad + \quad \{a_{j+1} \cdot x_j \} \quad + \quad \{a_{i+1} \cdot x_i \} \quad + \quad \{a_{i,i-1} \cdot x_{i-1} \} \]
\[ D_i = D_i \quad + \quad D_i \quad + \quad D_i \]
end if
end do in parallel
output \( x_i \)

Algorithm 3

From Algorithm 3 and Figure 3, we see that \( P = n, \ B = n + 1, \ C = n + 1, \ T_D = n + 3, \ R_C \sim 2, \ R_D \sim 2 \) and \( R \sim 4 \).

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 \( O(n/2) \) by computing two \( x_i \)'s each cycle using the formula:

\[ x_i = \frac{b_i}{a_{ii}} - \sum_{j=1}^{i-1} \frac{a_{ij}}{a_{ii}} \cdot x_j - \frac{a_{ii} \cdot x_{i-1}}{a_{ii}} \]

(2)

for \( i > 1 \) and odd, and

\[ x_i = \frac{b_i}{a_{ii}} - \frac{a_{ii} \cdot a_{i1} \cdot x_1 \cdot a_{i2} \cdot x_2 \cdots a_{i,i-1} \cdot x_{i-1}}{a_{ii}} \]

(3)

For \( i > 2 \) and even.

First, \( x_1 \) is computed as \( x_1 = b_1 / a_{11} \) and \( x_2 \) as \( x_2 = b_2 / a_{22} \cdot a_{11} / a_{21} \cdot b_1 \). 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 \( x_4 \), \( x_2 \) 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 \( O(24) \), but 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 \( O(n/k) \) for any fixed \( k \).

2.5. Broadcast Chain With \( O(n) \) Bandwidth

The architecture and data pattern are shown in Figure 4. The algorithm used is given in Algorithm 4. The performance figures for the broadcast chain are \( P = n, \ B = n, \ C = n, \ TD = n + 2, \ R_C \sim 2, \ R_D \sim 2 \) and \( R \sim 4 \).

One should note that each computation of Figure 4 takes \( t_{SUB} + t_{MUL} + t_{DIV} \) time, while in all of our previous designs, a computation required only \( t_{DIV} \) when 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 operations. The new design is as in Figure 5. Equation (1) is used to compute the \( x_i \). The dividers are loaded with \( a_{ii} \). Each input to PE\( i \) is divided by \( a_{ii} \) 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
\[ \max \{ t_{DIV}, t_{SUB} + t_{MUL} \} \]
when Figure 5 is used rather
than \[ t_{DIV} + t_{SUB} + t_{MUL} \]. The performance figures
using Figure 5 are \[ P = n, B = n, T_C = n + 1, T_D = n + 3, R_C = 2, R_D = 2 \] and \[ R = 4 \].

\[ D_j \leftarrow a_{ji} \] (input)
\[ c_j \leftarrow b_j \] (input)
\[ x_1 \leftarrow c_1 / D_1 \]
for \( i = 1 \) to \( n - 1 \) do
  do in parallel
    broadcast \( x_i \) to \( x_j, \quad j > i \)
    output \( x_i \)
    \[ A_j \leftarrow a_{ji}, \quad j > i \] (input)
  end
end

\[ x_i \leftarrow (c_j - A_j \times x_j)/D_j, \quad j > i + 1 \]
\[ c_j \leftarrow c_j - A_j \times x_j, \quad j > i + 1 \]

end

Algorithm 4

The \( R \) value of both the preceding designs may be
reduced to \( \approx 9/4 \) by using half as many PEs. For example,
the design of Figure 4 translates into that of Figure 6.
The \( n/2 \) PE broadcast chain is first used to compute
\( x_1, x_2, \ldots, x_{n/2} \) in exactly the same way as Figure 4 is
used when \( A \) is an \( n/2 \times n/2 \) matrix. Next, the
remaining \( n/2 \times 2 \)s are computed by having the \( n/2 \) PEs
behave like the last \( n/2 \) PEs of Figure 4 with the
required \( x_i \)'s being re-input and broadcast. To compute
the first \( n/2 \times 2 \)s, \( n/2 \) compute steps and \( n/2 + 2 \) data
move steps are needed. For the remaining \( n/2 \times 2 \)s, an
additional \( n \) compute steps and \( n + 2 \) data move steps
are needed. Hence, \( P = n/2, B = n + 2 + 1, T_C = 3n/2, T_D = 3n/2 + 4, R_C = 3/2, R_D = 3/2 \)
and \( R = 9/4 \). 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
\[ P = n/4, B = n/4 + 3, T_C = 5n/2, T_D = 5n/2 + 8, R_C = 5/4, R_D = 5/4 \] and
\( R = 25/16 \). 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 \( R = 1 \).

\[ \begin{array}{cccc}
1 & 2 & 3 & 4 \\
\hline
1 & 2 & 3 & 4 \\
2 & & & \\
3 & & & \\
4 & & & \\
\end{array} \]

Figure 5

2.6. \( o(n) \) Throughput With Unidirectional Data Flow

The unidirectional chain of Figure 1 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(0) 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 Figure 7. 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 lower PE computes the even terms. Predi-
vision of the \( a_{ij} \)'s by \( a_{ij} \) as suggested by formula (1) is
done by each PE. The upper PE in the pair for \( x_i \) computes
\[ U_i = b_i - \sum_{1 \leq j < i} a_{ij} \times x_j \]
when \( i \) is even and
\[ U_i = \sum_{1 \leq j < i} a_{ij} \times x_j \]
when \( i \) is odd. The lower PE
computes
\[ L_i = b_i - \sum_{1 \leq j < i} a_{ij} \times x_j \]
when \( i \) is odd and
\[ L_i = \sum_{1 \leq j < i} a_{ij} \times x_j \]
when \( i \) is 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 per-
fomed in the same cycle as the multiplication of \( a_{ij} \)
and \( x_{i-1} \). Hence, the cycle time is
\[ \max \{ t_{DIV}, \max \{ t_{QUL}, t_{SUB} \} \} \] (assuming
the add time to be roughly equal to the subtract time). The
working of the systolic system is described formally in
(6).

As drawn, the design of Figure 7 has a bandwidth of
\( 2n \). This is easily reduced to \( n \) by delaying the inputs of
both the upper row of \( a_{ij} \)'s and the row following the
lower row of \( a_{ij} \)'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 perfor-
amance figures for this design are \( P = 2n - 2, B = n, T_C = n + 1, T_D = n + 4, R_C = 4, R_D = 2 \)
and \( R = 8 \).

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 eliminated. Figure 8(a)
shows the input pattern for the case \( n = 4 \) while Figure
Improved performance can be obtained using a somewhat simpler algorithm, Algorithm 6, and the data pattern of Figure 9. This time the chain is a bidirectional chain. The number of PEs used is \( n - 1 \). The performance figures for this design are \( P = n - 1, B = 1, T_C = 2n - 1, T_D = n(n + 5)/2, R_C \sim 4, R_D \sim 1 \) and \( R \sim 4 \).

\[
\begin{align*}
c_1 &\leftarrow b_1 \quad \text{\{input\}}
A_1 &\leftarrow a_{11} \quad \text{\{input\}}
x_1 &\leftarrow c_1 / A_1 \quad \text{output } x_1
\end{align*}
\]

\[\text{for } i = 2 \text{ to } n \text{ do}
\]
\[\begin{align*}
&\text{do in parallel}
&c_i \leftarrow b_i \pm i - x_i \quad \text{\{input\}}
&c_j \leftarrow c_j - i, \quad 2 \leq j \leq n
&\end{align*}\]
\[\text{end}
\]
\[\text{end}
\]

\[\text{for } i = 1 \text{ to } n - 1 \text{ do}
\]
\[\begin{align*}
&\text{for } k = 1 \text{ to } \lfloor i/2 \rfloor \text{ do}
&\text{do in parallel}
&A_i \leftarrow a_{i, i/2} + k, \lfloor i/2 \rfloor - k + 1 \quad \text{\{input\}}
&A_j \leftarrow A_j - i, \quad 2 \leq j \leq k
&\end{align*}\]
\[\text{end}
\]

\[\text{end}
\]

\[\text{end}
\]

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

2.8. Summary

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 \( T_C = o(n) \) is half that of the best designs with \( T_C = o(2n) \).

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

3. REFERENCES


