Parallel Solution Methods for Triangular Linear Systems of Equations

by

Samuel E. Orcutt

Technical Report No. 77

June, 1974

Digital Systems Laboratory
Stanford Electronics Laboratories
Stanford, California

This work was supported by Bell Laboratories and by the National Science Foundation under grant GJ-41093.

Manuscript documentation unit for:

"Parallel solution Methods for
Triangular Linear Systems of Equations"

by Samuel E. Orcutt

Index terms: linear systems, parallel algorithms,
matrix inversion, recursive doubling.

Preferred address for future correspondence:

S. E. Orcutt
Bell Telephone Laboratories
Naperville, Illinois 60540

Footnotes

Affiliation of the author: This work was done while the author was at Stanford University. The author is now with Bell Laboratories, Naperville, Illinois 60540.

This work was supported by Bell Laboratories and by the National Science Foundation under Grant GJ-41093.

Figure Captions

Fig. 4.1a Initial conditions for reduction

Fig. 4.1b Final conditions after reduction

Fig. 4.2a Form of matrix Z_1

Fig. 4.2b Form of matrix Z_2

Fig. 6.1a jth set of rows before being modified

Fig. 6.1b jth set of rows after being modified

Fig. 7.1a Initial conditions for reduction ($m \leq 2^k$)

Fig. 7.1b Final conditions after reduction ($m \leq 2^k$)

Fig. 7.2a Initial conditions for reduction ($m > 2^k$)

Fig. 7.2b Final conditions after reduction ($m > 2^k$)

Fig. 7.22 Form of matrix E

Fig. 7.3 Initial condition of band triangular system

ABSTRACT

In this paper we consider developing parallel solution methods for triangular linear systems of equations. For a system of N equations in N unknowns the serial method requires $O(N^2)$ steps, and the straightforward parallel method requires O(N) steps and O(N) processors. In this paper we develop methods that require $O(\log^2 N)$ time when used with $O(N^3)$ processors and $O(\sqrt{N} \log N)$ time when used with $O(N^2)$ processors. We also consider solutions to band triangular systems and develop a method that requires $O((\log N)(\log m))$ time and $O(Nm^2)$ processors, where m is the bandwidth of the system.

1. INTRODUCTION

Recent advances in electronic circuitry have brought down the cost of computer components quite dramatically. These reductions have made possible the construction of computers with large numbers of processors capable of operating simultaneously, that is, parallel computers. Probably the best known of these machines is the TLLIAC IV [Barnes, et al., 1968]. The development of these machines has prompted considerable research into structuring algorithms for this type of machine, for example, the log-sum algorithm ['luck, 1968], dynamic programming [Gilmore, 1968], tridiagonal linear systems solver [Stone, 1973], and solutions to linear recurrences [Kogge, 1974]. Most of this research has been toward obtaining maximum possible speedup when using a machine with n processors to solve a problem whose size, in some appropriate measure, is n.

In this paper we consider developing parallel solution methods for triangular linear systems of equations. For a system of N equations in N unknowns the serial method requires $O(N^2)$ steps, and the straightforward parallel method requires O(N) steps when designed for a machine with N processors. The fastest method known to this author is that of Heller [1973]. By applying matrix theoretical arguments to a lower Hessenberg matrix derived from the original triangular matrix, he developed a method which requires

$O(\log^2 N)$* time and $O(N^4)$ processors. By applying recursive doubling arguments, similar to those of Stone [1973] and Kogge [1974], directly to the recurrence relation represented by the triangular system, we also develop a method requiring $O(\log^2 N)$ time. Our method requires only $O(N^3)$ processors to achieve a time requirement identical with Heller's.

In Section 2 we give serial and straightforward parallel algorithms for the solution to triangular linear systems. Section 3 presents the basic principle upon which our method is based, the principle of recursive doubling developed by Stone. In Section 4 we develop our method for solution to triangular linear systems of equations. This problem is of interest as it forms a fundamental step in the solution of general linear systems of equations when using the LU factorization.

The method developed in Section 4 solves triangular systems by considering the individual elements of the matrix rather than the matrix as a whole. In Section 5 we develop an alternative solution method that determines the inverse of a triangular matrix in $O(\log^2 N)$ steps utilizing only operations on the entire matrix. This method also requires $O(N^3)$ processors.

In Section 6 we develop a method that allows triangular systems to be solved in $O(\sqrt{N} \log N)$ time using $O(N^2)$

---

* Throughout this paper all logarithms are taken to the base 2.

processors. Although this time is larger than for the previous algorithms, it requires fewer processors.

An interesting special case of triangular system is the triangular band system. In Section 7 we develop a method for solving these systems in $O((\log N)(\log m))$ time, where m is the bandwidth of the system,. using $O(Nm^2)$ processors.

2. TRIANGULAR LINEAR SYSTEMS OF EQUATIONS

For a computer to be useful, it should be capable of solving more than a single class of problems. We first consider the problem of solving triangular linear systems of equations.

We wish to solve problems of the form

$$M\ y = b$$

where

$$M = \begin{bmatrix} m_{11} & & & & \\ m_{21} & m_{22} & & & \\ m_{31} & m_{32} & m_{33} & & \\ \cdot & \cdot & \cdot & \cdot & \\ m_{n1} & m_{n2} & m_{n3} & \cdot \cdot \cdot & m_{nn} \end{bmatrix}$$

For the purpose of our derivation, we choose to work with an equivalent formulation. Consider evaluating sequences of the form

$$\text{[illegible]} = \sum_{j=0}^{i-1} A(i,j) \cdot y(i) + H(i), \qquad 0 \leq i \leq N.$$

A, H, and N are related to M, b, and n by

$$H(i) = b_{i+1}/m_{i+1,i+1} ;$$

$$A(i,j) = m_{i+1,j+1}/m_{i+1,i+1}, \qquad 0 \leq j \leq i-1 ;$$

$$N = n - 1 .$$

These sequences can be easily evaluated on a serial computer in the following manner.

```
for i: = 0 step 1 until N do y(i): = H(i);
for j: = 0 step 1 until N-1 do
for i: = j+1 step 1 until N do
 y(i): = y(i) + A(i,j) × y(j);
```

This algorithm requires $N\cdot(N+1)/2$ each of additions and multiplications. The algorithm thus requires $O(N^2)$ steps.

From the above serial algorithm we derive the following parallel algorithm for a machine of the ILLIAC IV-type.

```
Y(i): = H(i),     (0 ≤ i ≤ N);
for j: = 0 step 1 until N-l do
y(i): = y(i) + A(i,j) × y(j),     (j+1 ≤ i ≤ N);
```

The (jtl< i ≤ N) following the assignment statement indicates that it is to be done simultaneously for all i in the range between j+1 and N. This algorithm requires N each of addition and multiplication steps, each consisting of up to N operations performed simultaneously in parallel. This yields a speedup of (N+1)/2 as compared with the sequential algorithm.

We would like to obtain further speedup of the algorithm but there appears to be no straightforward way in which this speedup can be obtained. For each value of j, the statement in the for loop requires the values of y from the previous iteration. This situation is quite similar to that encountered in Stone [1973] for tridiagonal systems. On this basis, we apply the techniques of recursive doubling to our problem.

3. THE BASIC PRINCIPLE

The basic principle used in the development of our algorithm is an extension of the technique termed recursive doubling by Stone [1973]. This technique is discussed in detail in Kogge [1974], and the interested reader is referred there for a more thorough discussion. By way of an example,

we now present sufficient background to enable the reader to understand the derivation of the next section.

Consider the problem of evaluating y(i), $0 \leq i \leq N$, where y(i) is defined by the linear recurrence

$$y(0) = y_0 \ ;$$

$$y(i) = A(i) * y(i-1) \ , \ i \geq 1 \ .$$

We proceed in the following manner. Substituting for y(i-1) yields

$$y(i) = A(f) * (A(i-1) * y(i-2))$$

$$= (A(i) * A(i-1)) * y(i-2)$$

$$= A^{(1)}(i) * y(i-2)$$

Similarly, substituting for y(i-2) yields

$$y(i) = A(1)(i) * (A^{(1)}(i-2) * y(i-4))$$

$$= (A^{(1)}(i) * A^{(1)}(i-2)) * y(i-4)$$

$$= A^{(2)}(i) * y(i-4)$$

That is, we proceed by deriving a sequence of equations for y(i) of the following form:

$$y(i) = A^{(k)}(i) * y(i-2^k), \quad A^{(k)}(i) = A^{(k-1)}(i) * A^{(k-1)}(i-2^{k-1}).$$

where

$$A^{(0)}(i) = A(i)$$

These equations, although valid in general, must be modified slightly to account for the boundary conditions in the recurrence that occur at y(0). In this way, we derive a method computing the values y(i), $1 \leq i \leq N$. Let $n = \lceil \log_2 N \rceil$. We evaluate $A^{(n)}(i)$, $1 \leq i \leq N$, according to the above formulas. From the definition of $A^{(n)}(i)$, we know that

$$y(i) = A^{(n)}(i) * y_0, \qquad 1 \leq i \leq N.$$

Thus, after computing the first $\lceil \log_2 N \rceil$ sets of $A^{(k)}(i)$, the values of y(i), $1 \leq i \leq N$, are all available as the result of a single multiplication. On an SIMD (Single Instruction Stream - Multiple Data Stream [Flynn, 1966]) computer with appropriate interconnections, this can be done in O(log N) time and requires O(N) processors. We now proceed with our main development.

4. A DOUBLING FORMULA

Consider evaluating y(i , $0 \leq i \leq N$, where y(i) is defined by the inhomogeneous linear recurrence

$$y(i) = \sum_{j=0}^{i-1} A(i,j) * y(j) \text{ t } H(i \quad , \qquad i \geq 0.$$

We proceed, in the same manner as for our previous example, to derive a sequence of equations for y(i) of the form

$$(*) \qquad y(i) = \sum_{j=0}^{i-2^k} A^{(k)}(i,j) * y(j) + H^{(k)}(i) \ , \qquad i \geq 0 \ .$$

A vacuous sum is interpreted as having the constant value 0. From these equations, we can evaluate y(i) by

$$y(i) = H^{(k)}(i) \ , \ k \geq \lceil \log_2 (i+1) \rceil \ .$$

To evaluate y(i), $0 \leq i \leq N$, we derive the $\lceil \log_2 (N+1) \rceil^{\text{th}}$ set of equations for y(i). We now give an inductive proof of validity of (*) which yields appropriate recurrence relations defining $A^{(k)}(i,j)$ and $H^{(k)}(i)$.

Basis Step: Let

$$A^{(0)}(i,j) = A(i,j) \text{ and } H^{(0)}(i) = H(i) \ .$$

From this we have immediately

$$y(i) = \sum_{j=0}^{i-2^0} A^{(0)}(i,j) * y(j) \text{ t } H^{(0)}(i) \ , \ i \geq 0 \ .$$

Induction Step:

Assume that (*) is valid for k = n-l. We prove that (*) is also valid for k = n. We know that

$$y(i) = \sum_{j=0}^{i-2^{n-1}} A^{(n-1)}(i,j) * y(j) \text{ t } H^{(n-1)}(i) .$$

Using the inductive hypothesis, we substitute for $y(i-2^{n-1},)$, $y(i-2^{n-1}-1), \ldots . y(i-2\%)$. This yields the following recurrence relations defining $A^{(n)}(i,j)$ and $H^{(n)}(i)$.

Case 1: $i \underline{'} 2^n+2^{n-1}-1$:

$$H^{(n)}(i) = H^{(n-1)}(i) + \sum_{j=i-2^n+1}^{i-2^{n-1}} A^{(n-1)}(i,j) * H^{(n-1)}(j) ;$$

$$A^{(n)}(i,j) = \begin{cases} A^{(n-1)}(i,j) + \sum_{k=i-2^n+1}^{i-2^{n-1}} A^{(n-1)}(i,k) * A^{(n-1)}(k,j) , \\ \qquad 0 \leq j \leq i-2^n+1-2^{n-1} , \\ A^{(n-1)}(i,j) + \sum_{k=j+2^{n-1}}^{i-2^{n-1}} A^{(n-1)}(i,k) * A^{(n-1)}(k,j) ; \\ \qquad 1-2^n+2-2^{n-1} < j < i-2^n . \end{cases}$$

Case 2: $2^n \leq i < 2^n+2^{n-1}-1$:

$$H^{(n)}(i) = H^{(n-1)}(i) + \sum_{j=i-2^n+1}^{i-2^{n-1}} A^{(n-1)}(i,j) * H^{(n-1)}(j) ;$$

$$A^{(n)}(i,j) = A^{(n-1)}(i, \cdot\cdot + \sum_{k=j+2^{n-1}}^{i-2^{n-1}} A^{(n-1)}(i,k) * A^{(n-1)}(k,j) ,$$

$$0 \leq j \leq i-2^n .$$

Case 3: $2^{n-1} \leq i < 2^n$:

$$H^{(n)}(i) = H^{(n-1)}(i) + \sum_{j=0}^{i-2^{n-1}} A^{(n-1)}(i,j) * H^{(n-1)}(j).$$

Case 4: $i < 2^{n-1}$:

$$H^{(n)}(i) = H^{(n-1)}(i) .$$

The four cases shown above can be reduced by noting that the differences between the cases are due to different bounds on otherwise identical summations.

The general way in which this method works can be described graphically in terms of the original triangular matrix. We perform operations on the rows of the matrix as

follows. Consider modifying row i where $i \geq 2^{k+1}$. In this case there exists a set of $2^k + 1$ rows of the matrix in the form of Figure 4.1a. The block of 2^k rows in this diagram are rows $i - 2^{k+1} + 1$ through $i - 2^k$. We modify this set of rows into the form of Figure 4.1b. This process corresponds to eliminating 2^k variables from the equation represented to the i^{th} row. The values of M_1' and b_1' are given by

$$M_1' = M_1 \times Z_1 - (M_1 \times Z_2) \times M_2 ,$$

$$b_1' = b_1 - (M_1 \times Z_2) \times b .$$

Z_1 and Z_2 are matrices such that $M_1 \times Z_1$ is a matrix consisting of the first $m - 2^k$ columns of M_1 and $M_1 \times Z_2$ is a matrix consisting of the last 2^k columns of Ml. These are shown schematically in Figure 4.2. We use a somewhat liberalized definition of matrix product. In particular, if you multiply an $m \times n$ matrix A times a $p \times q$ matrix B, you obtain an $m \times q$ matrix C with

$$c_{ij} = \sum_{k=1}^{\min(n,p)} A(i,k) \times B(k,j) .$$

For the rows $2^k \leq i < 2^{k+1}$ the situations is similar to that presented except that the matrices M_2 and b_2 are no longer

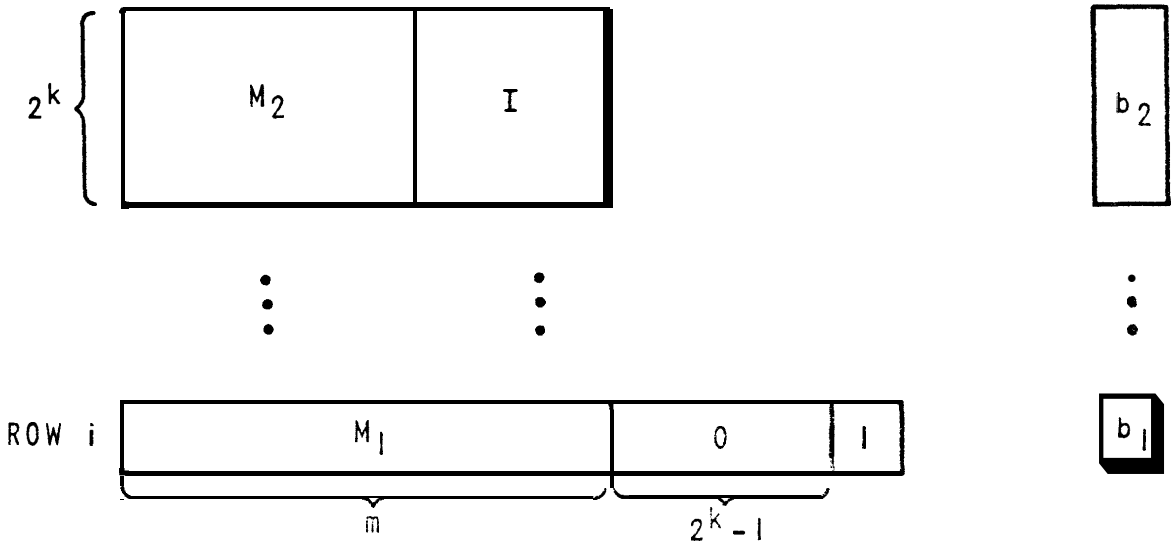

FIGURE 4.1a
INITIAL CONDITIONS FOR REDUCTION

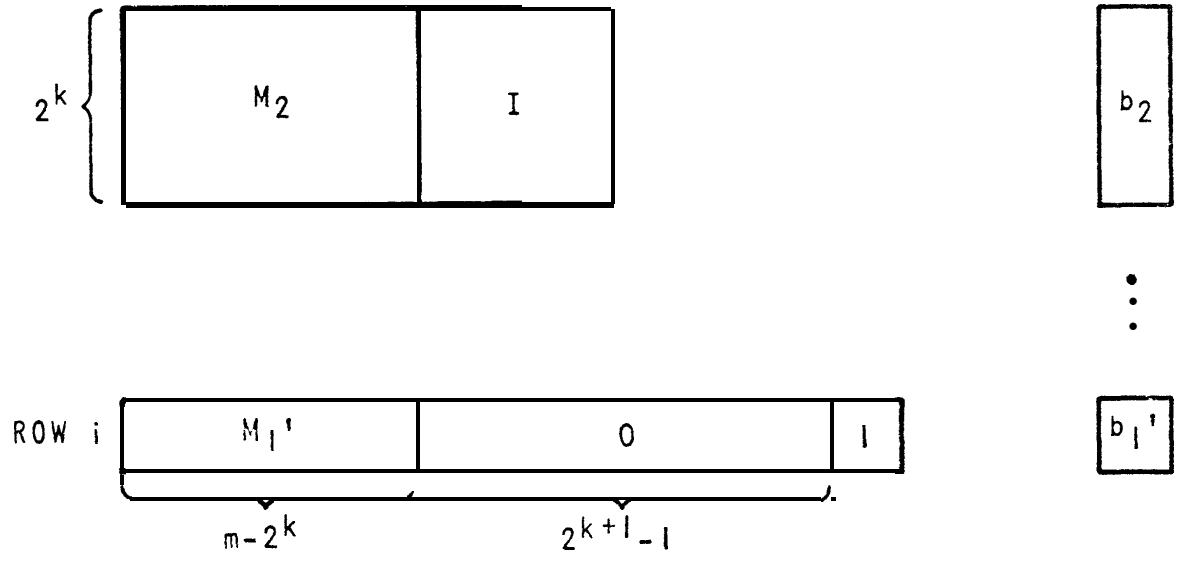

FIGURE 4.1 b
FINAL CONDITIGNS AFTER REDUCTION

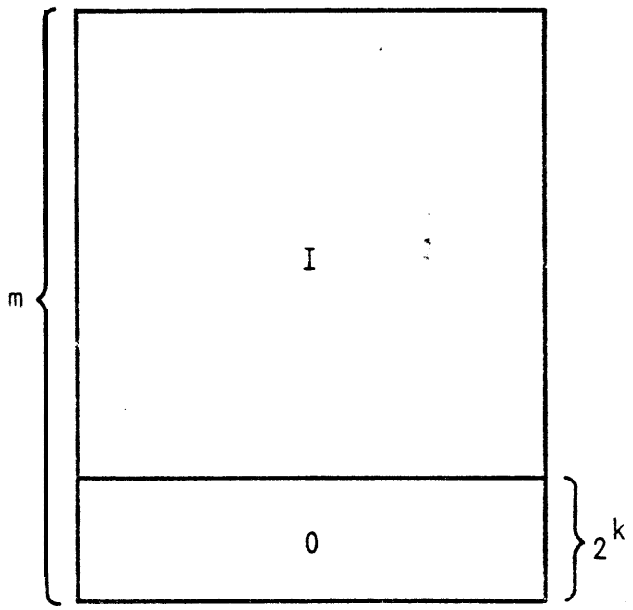

FIGURE 4.2a
FORM OF MATRIX Z_1

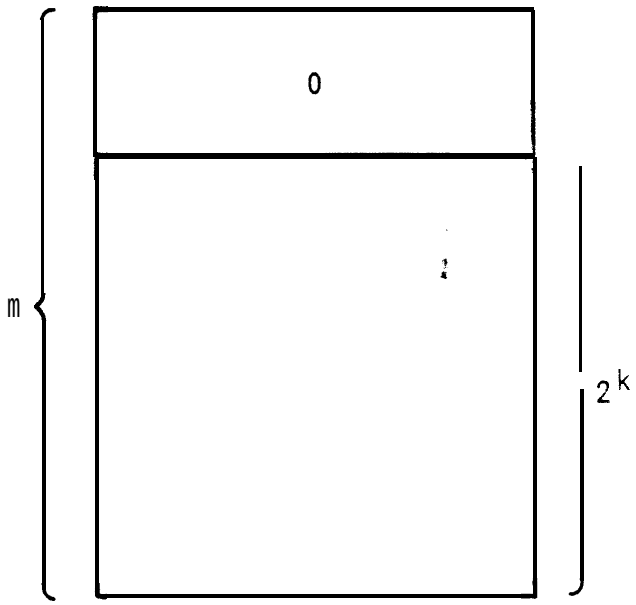

FIGURE 4.2b

FORM OF MATRIX Z_2

2^k rows long. We perform the changes just described to all rows i, $i \geq 2^k$. In this way many new zero elements are introduced into the matrix. These elements are introduced in a manner corresponding to the elimination of 2^k diagonals of the matrix. In this way, in the first step we eliminate the first subdiagonal, the second step eliminates the next two subdiagonals, then the next four, and so on until there is only an identity matrix remaining. This process is shown in Figure 4.3 for the case n = 5. At this point the solution vector is easily determined as it is identical to the modified right-hand side.

The execution time and process requirements of this method can also be determined. Eliminating all the subdiagonals of the matrix requires $\lceil \log_2 n \rceil$ applications of the recurrence. Each of these applications requires the computation of inner products of vectors of length at most N/2 which requires at most $\lceil \log_2 N \rceil$. From this, the time required is seen to be $O(\log^2 N)$. From similar considerations, the processor requirement can be seen to be $O(N^3)$.

$$\left[\begin{array}{ccccc|c} 1 & & & & & 10 \\ 3 & & & & & 14 \\ 2 & & 1 & & & 12 \\ 6 & & 3 & 1 & & 3 \\ 1 & & 1 & 5 & 1 & 6 \end{array}\right]$$

$$\left[\begin{array}{ccccc|c} 1 & & & & & 10 \\ 0 & 1 & & & & -16 \\ -4 & 0 & 1 & & & -16 \\ 0 & -2 & 0 & 1 & & -33 \\ -29 & -18 & -14 & 0 & 1 & -9 \end{array}\right]$$

$$\left[\begin{array}{ccccc|c} -1 & & & & & 10 \\ 0 & 1 & & & & -16 \\ 0 & 0 & 1 & & & 24 \\ 0 & 0 & 0 & 1 & & -65 \\ -85 & 0 & 0 & 0 & 1 & -521 \end{array}\right]$$

$$\left[\begin{array}{ccccc|c} 1 & & & & & 10 \\ 0 & 1 & & & & -16 \\ 0 & 0 & 1 & & & 24 \\ 0 & 0 & 0 & 1 & & -65 \\ 0 & 0 & 0 & 0 & 1 & 329 \end{array}\right]$$

Figure 4.3

Numerical Example for N = 5

5. AN ALTERNATIVE TRIANGULAR SYSTEM SOLVER

In a previous section we developed an algorithm for solving triangular linear systems of equations in $O(\log^2 N)$ time for a system of N equations and N unknowns. This algori'thm was developed by applying the principle cf recursive doubling [Stone, 1973; Kogge, 1974] to the recurrence relation equivalent to the linear system. This approach is not entirely satisfying from the matrix theory point of view. In this section we present an algorithm for computing the inverse of a triangular matrix in $O(\log^2 N)$ time using only matrix operations rather than operations on the individual elements of the matrix. Heller [1974] presented, without proof, the same solution method. The derivation we present here was developed independently and shows the validity of this method. To proceed with this development we first prove two lemmas.

Lemma 1 [Ostrowski, 1936]:

For any N × N matrix X, we have

$$(I-X)^{-1} = (I-X^{2^k})^{-1}\,(1+X)\,(I+X^2)(I+X^4)\ldots(I+X^{2^{k-1}})$$

where I is an N × N identity matrix.

Proof:

We start with the ident

$$(I-X)^{-1} = (I-X)^{-1}\ .$$

The right-hand side of this equation is multiplied on the right by

$$I = (I+X)^{-1}(I+X^2)^{-1}(I+X^4)^{-1}\ldots(I+X^{2^{k-l}})^{-1}(I+X^{2^{k-l}})\ldots(I+X^2)(I+X) .$$

We reduce this by applying reductions similar to

$$(I-X^k)^{-1}\,(I+X^k)^{-1} = (I-X^{2k})^{-1}$$

until only a single term involves an inverse. Since the terms $(I-t-X^k)$ are polynomials in the matrix X, they commute. Using this commutativity, we reorder the terms into the order given in the statement of the lemma. QED

Lemma 2:

If A and B are arbitrary N × N matrices with $a_{ij} = 0$ for all i < jfkl and $b_{ij} = 0$ for all $i < j+k_2$, then C = A·B is a matrix with $c_{ij} = 0$ for all $i < j+k_1+k_2$.

Proof:

For c_{ij} to possibly be nonzero, we must have

$$\sum_{\ell=1}^{n} a_{i\ell}\cdot b_{\ell j} \neq 0.$$

This implies that

$$\exists\, \ell \mid i \geq \ell+K_1 \text{ and } \ell \geq j+k_2$$

or equivalently

$$i \geq j+k_1+k_2 .$$

From this it is easily seen that $c_{ij} = 0$ for $i < j+k_1+k_2$. QED

With these two lemmas we can derive an equation for A^{-1} where A is lower triangular with unit diagonal elements. To proceed, we let X in Lemma 1 be replaced by (I-A). This yields

$$A^{-1} = (I-(I-A))-1 = (I-(I-A)^{2^n})^{-1}(I+(I-A))\ (I+(I-A)^2)\ldots(I+(I-A)^{2^{n-1}}).$$

Let $n = \lceil \log_2 N \rceil$. This means that $2^n \geq N$. The matrix (I-A) is such that $(I-A)_{ij} = 0$ for all $i < j+1$. Applying Lemma 2 repeatedly, we see that $((I-A)^{2^n})_{ij} = 0$ for all $i < j+2^n$. Since $2^n \geq N$, this is also true for all $i < j+N$ which includes all elements of the matrix. From this we determine that

$$(I-(I-A)^{2^n})^{-1} = (I-0)^{-1} = I^{-1} = I$$

where 0 is an N × N zero matrix. Substituting into the original equation for A^{-1}, we obtain

$$A^{-1} = (I+(I-A))\,(I+(I-A)^2)(I+(I-A)^4)\ .\ .\ .\ (I+(I-A)^{2^{n-1}}) .$$

We can determine the required time for this method fairly easily. All the required terms

$$(I-A)^k$$

can be determined by repeated squarings. This requires $O(\log N)$ steps each taking $O(\log N)$ time for a total time of $O(\log^2 N)$. The terms

$$(I+(I-A)^k)$$

can now all be computed with $O(\log N)$ matrix additions each requiring $O(1)$ steps for a total of $O(\log N)$ steps. The final value of A^{-1} is obtained by another $O(\log N)$ matrix products. The total overall time required is thus $O(\log^2 N)$. As for the processor requirements, the operations that are used are matrix products and sums requiring $O(N^3)$ and $O(N^2)$ processor:, respectively. This implies a processor requirement of $O(N^3)$.

6. A TRIANGULAR SYSTEMS SOLVER FOR $O(N^2)$ PROCESSORS

The solution methods presented in the preceding two sections required $O(N^3)$ processors for their execution. This rate of growth is quite rapid as N becomes large. In this section we briefly describe a method that makes a trade-off between reducing the number of processors required and increasing the execution time. More precisely, the

method we present requires only $O(N^2)$ processors to attain an execution time of $O(\sqrt{N} \log N)$.

This method proceeds in two stages. In the first stage we consider the rows of the matrix in blocks of $\sqrt{N}$ consecutive rows. All of these blocks are treated identically and in parallel. A typical block is shown in Figure 6.1a. M_j is a full matrix and L_j is a lower triangular matrix with unit diagonal elements. Proceeding by rows, we eliminate the subdiagonal elements of L_j. This yields the situation shown in Figure 6.1b where

$$M_j' = L_j^{-1} M_j$$

$$b_j' = L_j^{-1} b_j$$

L_j^{-1} is never actually formed but the equivalent effect is obtained by the row eliminatfcns. The execution time and processor requirements for this stage are easily determined. At each step we modify a row of at most N elements in each of $\sqrt{N}$ blocks. Updating each element requires computation of inner products of a- ost $\sqrt{N}$ elements. This requires a total of $O(N)$ $O(\sqrt{N})$ $O(\sqrt{N}) = O(N^2)$ processors. To determine the execution time requirements, we note that this process requires $\sqrt{N} - 1$ steps each involving inner products of at

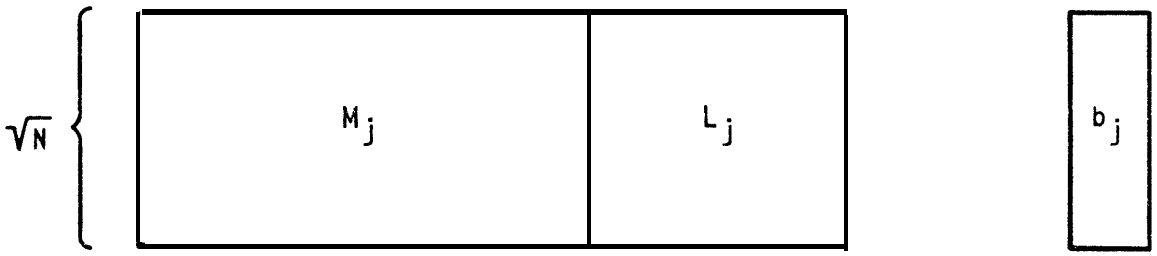

FIGURE 6.1a
j^{th} SET OF ROWS BEFORE BEING MODIFIED

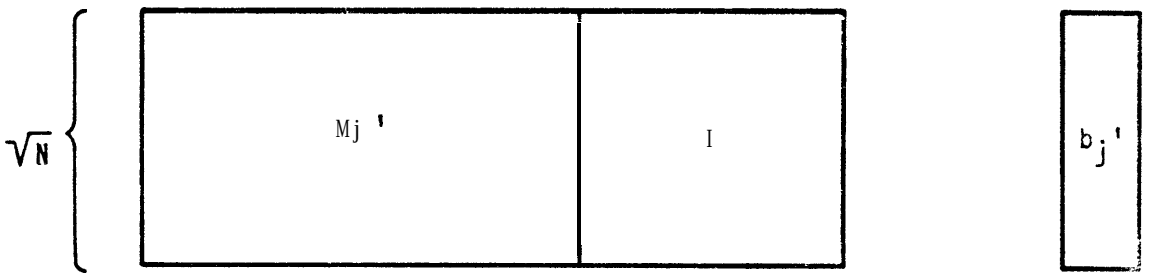

FIGURE 6.1 b
j^{th} SET OF ROWS AFTER BEING MODIFIED

most $\sqrt{N}$ elements yielding an execution time of $O(\sqrt{N} \log N)$. The execution of this stage of the process converts the matrix from lower triangular form with unit diagonal elements into block lower triangular form with identity matrices on the diagonal.

The second half of the method proceeds in a manner logically equivalent to the first half except that the elements involved are $\sqrt{N} \times \sqrt{N}$ matrices rather than scalars. The modifications performed are those of Figure 6.1a with M_j a null matrix. The time and processor requirements are again easy to determine. At each step, we compute at most $\sqrt{N}$ inner products each of at most N elements. This requires $O(N^{3/2})$ Processors. For the time requirements, there are $\sqrt{N} - 1$ steps each requiring the computation of inner products of at most N elements. This yields a time of $O(\sqrt{N} \log N)$.

To compute the overall requirements, we take the maximum of the processor requirements and the sum of the time requirements. This yields a total time of $O(\sqrt{N} \log N)$ and a total processor requirement of $O(N^2)$. Figure 6.2 gives an example of this method for $N = 9$.

$$\begin{bmatrix} 1 & & & & & & & & \\ -3 & 1 & & & & & & & \\ -3 & -1 & 1 & & & & & & \\ 3 & -4 & -1 & 1 & & & & & \\ -5 & -3 & -1 & -5 & 1 & & & & \\ 1 & -3 & -3 & -3 & -2 & 1 & & & \\ -4 & 0 & -3 & -5 & -2 & 4 & 1 & & \\ -2 & -2 & -1 & -3 & -2 & 3 & 1 & 1 & \\ 3 & -4 & -3 & 3 & 3 & 3 & -3 & 4 & 1 \end{bmatrix} \quad \begin{bmatrix} 1 \\ -7 \\ 4 \\ 11 \\ 32 \\ 8 \\ -16 \\ -9 \\ -21 \end{bmatrix}$$

$$\begin{bmatrix} 1 & & & & & & & & \\ 0 & 1 & & & & & & & \\ -3 & -1 & 1 & & & & & & \\ 3 & -4 & -1 & 1 & & & & & \\ 10 & -23 & -6 & 0 & 1 & & & & \\ 1 & -3 & -3 & -3 & -2 & 1 & & & \\ -4 & 0 & -3 & -5 & -2 & 4 & 1 & & \\ 2 & -2 & 2 & 2 & 0 & -1 & 0 & 1 & \\ 3 & -4 & -3 & 3 & 3 & 3 & -3 & 4 & 1 \end{bmatrix} \quad \begin{bmatrix} 1 \\ -4 \\ 4 \\ 11 \\ 87 \\ 8 \\ -16 \\ 7 \\ -21 \end{bmatrix}$$

$$\begin{bmatrix} 1 & & & & & & & & \\ 0 & 1 & & & & & & & \\ 0 & 0 & 1 & & & & & & \\ 3 & -4 & -1 & 1 & & & & & \\ 10 & -23 & -6 & 0 & 1 & & & & \\ 30 & -61 & -18 & 0 & 0 & 1 & & & \\ -4 & 0 & -3 & -5 & -2 & 4 & 1 & & \\ 2 & -2 & 2 & 2 & 0 & -1 & 0 & 1 & \\ -17 & 4 & -20 & -20 & -3 & 19 & 0 & 0 & 1 \end{bmatrix} \quad \begin{bmatrix} 1 \\ -4 \\ 3 \\ 11 \\ 87 \\ 215 \\ -16 \\ 7 \\ -97 \end{bmatrix}$$

$$\begin{bmatrix} 1 & & & & & & & & \\ 0 & 1 & & & & & & & \\ 0 & 0 & 1 & & & & & & \\ 0 & 0 & 0 & 1 & & & & & \\ 0 & 0 & 0 & 0 & i & & & & \\ 0 & 0 & 0 & 0 & 0 & 1 & & & \\ -4 & 0 & -3 & -5 & -2 & 4 & 1 & & \\ 2 & -2 & 2 & 2 & 0 & -1 & 0 & 1 & \\ -17 & 4 & -20 & -20 & -3 & 19 & 0 & 0 & 1 \end{bmatrix} \quad \begin{bmatrix} 1 \\ -4 \\ 3 \\ -5 \\ 3 \\ -5 \\ -16 \\ 7 \\ -97 \end{bmatrix}$$

$$\begin{bmatrix} 1 & & & & & & & & \\ 0 & 1 & & & & & & & \\ 0 & 0 & 1 & & & & & & \\ 0 & 0 & 0 & 1 & & & & & \\ 0 & 0 & 0 & 0 & 1 & & & & \\ 0 & 0 & 0 & 0 & 0 & 1 & & & \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & & \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \quad \begin{bmatrix} 1 \\ -4 \\ 3 \\ -5 \\ 3 \\ -5 \\ -2 \\ -4 \\ 0 \end{bmatrix}$$

Figure 6.2

Numerical Example of Procedure for N = 9

7. BAND TRIANGULAR SYSTEMS

Although the general triangular system solution methods can also be used to solve band triangular systems, it would be desirable to utilize the special form of the matrix to reduce the time and processor requirements. In this section we develop a method that accomplishes this.

Before presenting the mathematical formulation of our method, we motivate the basic concept used in a graphical manner. Consider the problem of solving

$$A \times y = b$$

where A is an $N \times N$ upper triangular matrix with unit diagonal elements and bandwidth m. We consider performing operations on the rows of A in the following manner. There are two cases to consider. In Figure 7.1 we have $2^k \geq m$. When this is the case, we transform the situation shown in 7.1a into that shown in 7.1b where

$$M_1' = -M_1 \times M_2 ,$$

$$b_1' = b_1 - M_1 \times b_2 ,$$

In the second case, $m > 2^k$, the situation is complicated somewhat by the overlapping columns of M_1 and M_2. In this case, we transform the situation shown in Figure 7.2a into that shown in 7.2b where

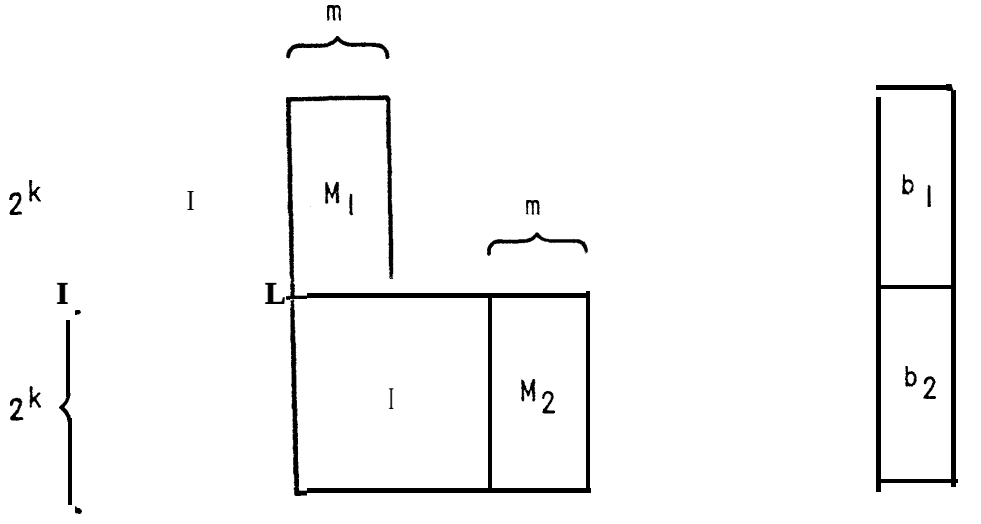

FIGURE 7.1a
INITIAL CONDITIONS FOR REDUCTION ($m < 2^k$)

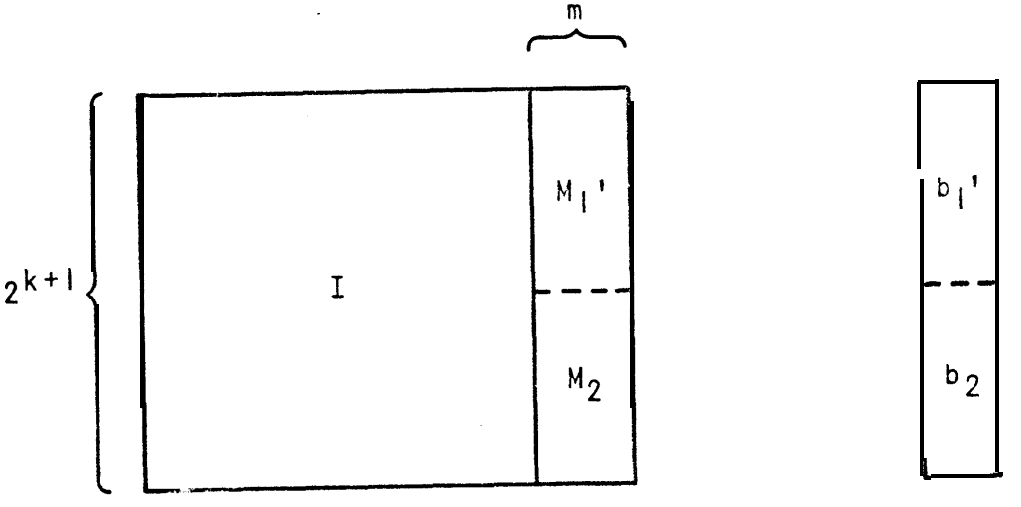

FIGURE 7.1 b
FINAL CONDITIONS AFTER REDUCTION ($m < 2^k$)

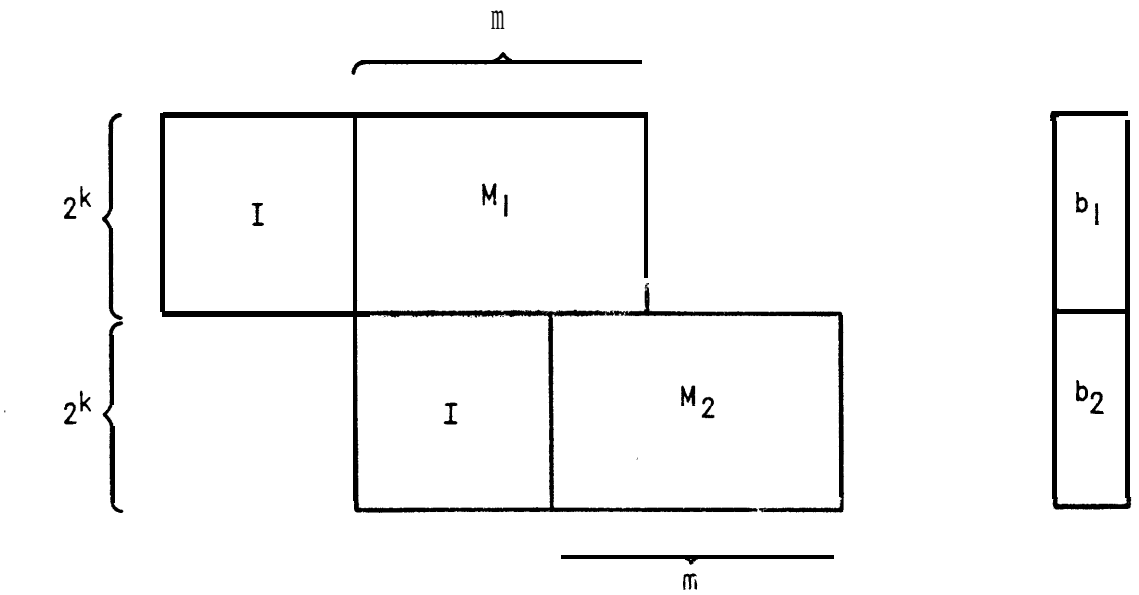

FIGURE 7.2a
INITIAL CONDITIONS FOR REDUCTION ($m > 2^k$)

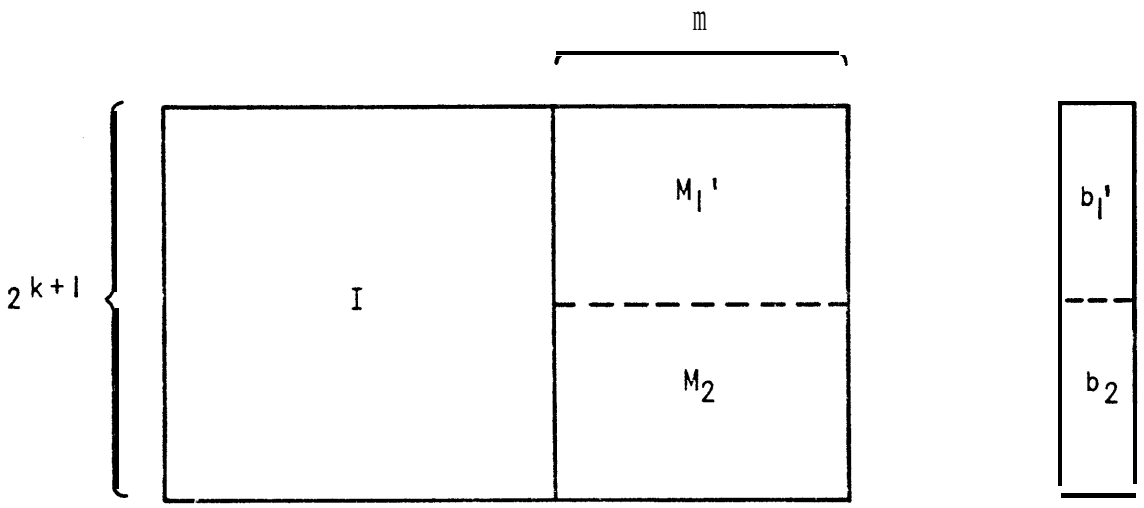

FIGURE 7.2b
FINAL CONDITIONS AFTER REDUCTION ($m > 2^k$)

$$M_1' = M_1 \times E - M_1 \times M_2,$$

$$b_1' = b1 - M_1 \times b_2,$$

and E is a matrix that shifts the columns of Ml 2^k columns to the left and inserts columns of zeroes on the right. E is shown in Figure 7.2c. These operations are equivalent to eliminating variables from the equation represented by the row of the matrix. In this way, we can reduce a pair of adjacent blocks of 2^k rows each into a single block of 2^{k+1} rows and still maintain the same general form for the rows.

Considering again the initial problem, we see that the rows of the matrix are initially in the form shown in Figure 7.3. This is precisely the form which we require in the case that k = 0. From this we see that the matrix can be reduced by applying the transformations just described. We first combine all odd - even pairs of rows to get blocks of size 2. Odd - even pairs of these blocks are then combined to yield blocks of size 4. This process Is continued until only a single block remains. At this time, the block must be an identity matrix and the solution is immediately available from the value of the modified right-hand side.

To put things on a more concrete foundation, we now present a mathematical formulation of the method we just

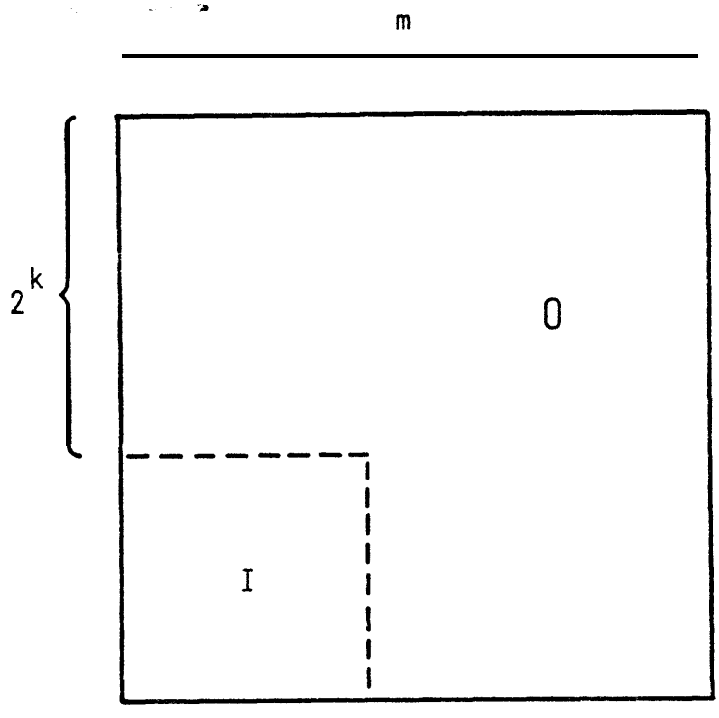

FIGURE 7.2c

FORM OF MATRIX E

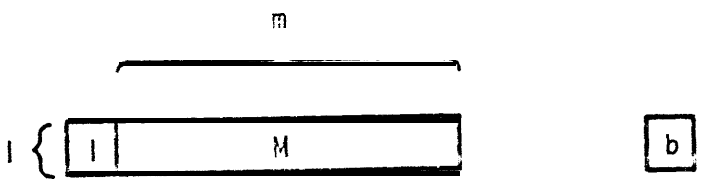

FIGURE 7.3

INITIAL CONDITION OF BAND TRIANGULAR SYSTEM

described. As for the general triangular system solver, we choose to work with the equivalent recurrence relation rather than the matrix formulation. In particular we consider solving

$$X(i) = \sum_{j=1}^{m} A(i,j)\cdot X(i-j) + b(i) ,$$

$$1 \leq i \leq N.$$

On the basis of the graphical description, we postulate that

$$X(i) = \sum_{j=1}^{m} A^{(k)}(i,j)\cdot X(i - (i-1) \bmod 2^k - j) + b^{(k)}(i) ,$$

$$1 < i \leq N .$$

Using this as our inductive hypothesis, we now show that it is valid. The proof yields the required recurrence relations for $A^{(k)}$ and $b^{(k)}$.

Basis Step:

$$X(i) = \sum_{j=1}^{m} A(i,j)\cdot X(i-j) + b(i)$$

$$= \sum_{j=1}^{m} A^{(0)}(i,j)\cdot X(i - (i-1) \bmod 2^0 - j) + b^{(0)}(i) .$$

Induction Step:

Case 1: $(i-1) \bmod 2^{k+1} = (i-1) \bmod 2^k$:

$$A^{(k+1)}(i,j) = A^{(k)}(i,j) ,$$

$$b^{(k+1)}(i) = b^{(k)}(i) .$$

Case 2: $(i-1) \bmod 2^{k+1} = (i-1) \bmod 2^k + 2^k$:

2a) $2^k < m$:

$$X(i) = \sum_{j=1}^{2^k} A^{(k)}(i,j) \cdot X(i - (i-1) \bmod 2^k - j)$$

$$+ \sum_{j=2^k+1}^{m} A^{(k)}(i,j) \cdot X(i - (i-1) \bmod 2^k - j) + b^{(k)}(i) .$$

Using the inductive hypothesis to substitute for the X's in the first summation, we are able to derive the following recurrence relations between the $A^{(k)}$ and $b^{(k)}$.

$$A^{(k+1)}(i,j) = \begin{cases} A^{(k)}(i,j+2^k) + \sum_{\ell=1}^{2^k} A^{(k)}(i,\ell)\cdot A^{(k)}(i-(i-1) \bmod 2^k - \ell, j), \\ \qquad 1 \le j \le m-2^k ; \\ \sum_{\ell=1}^{2^k} A^{(k)}(i,\ell)\cdot A^{(k)}(i-(i-1) \bmod 2^k - \ell, j), \\ \qquad m-2^k+1 \le j \le m ; \end{cases}$$

$$b^{(k+1)}(i) = b^{(k)}(i) + \sum_{\ell=1}^{2^k} A^{(k)}(i,\ell)\cdot b^{(k)}(i-(i-1) \bmod 2^k - \ell) .$$

2b) $2^k \ge m$:

$$X(i) = \sum_{j=1}^{m} A^{(k)}(i,j)\cdot X(i-(i-1) \bmod 2^k - j) + b^{(k)}(i)$$

We again use the inductive hypothesis to substitute for the X's in the summation. This yields the following recurrence relations.

$$A^{(k+1)}(i,j) = \sum_{\ell=1}^{m} A^{(k)}(i,\ell)\cdot A^{(k)}(i-(i-1) \bmod 2^k - \ell, j) ,$$

$$b^{(k+1)}(i) = b^{(k)}(i) + \sum_{\ell=1}^{m} A^{(k)}(i,\ell)\cdot b^{(k)}(i-(I-1) \bmod 2^k - \ell).$$

From this we see that our inductive hypothesis was in fact valid and we determine the recurrence relations defining the $A^{(k)}$ and $b^{(k)}$.

Utilizing the recurrence relations, we can solve our problem in the following way. We derive the $\lceil \log_2 N \rceil^{\text{th}}$ set of equations for $A^{(k)}$ and $b^{(k)}$. From the nature of these equations, it is easily seen tnat each X is given as a summation involving only Xs with nonpositive indices and a constant term. The mapping between the original triangular system and the recurrence relation assigns all Xs with non-positive indices a value of 0. This implies that the value of X(i) is b(i) in the $\lceil \log_2 N \rceil^{\text{th}}$ set of equations. The function of this algorithm is easily shown graphically. Figure 7.4 gives a numerical example for N = 6 and m = 2.

To determine the execution time and processor requirements for this method, we consider the time and number of processors involved in a single step of the method. At each step we compute at most Nm/2 inner products each product being of vectors of length at most m, giving a processor requirement of $O(m^2N)$. Since each step requires O(log m) time and there are O(log N) steps, the total time required by the method is O((log m)(log N)).

$$\begin{bmatrix} 1 & 3 & 2 & 0 & 0 & 0 \\ & 1 & 4 & 1 & 0 & 0 \\ & & 1 & 6 & 2 & 0 \\ & & & 1 & 1 & 3 \\ & & & & 1 & 4 \\ & & & & & 1 \end{bmatrix} \quad \begin{bmatrix} 8 \\ 4 \\ 1 \\ 6 \\ 9 \\ 7 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 0 & -10 & -3 & 0 & 0 \\ & 1 & 4 & 1 & 0 & 0 \\ & & 1 & 0 & -4 & -18 \\ & & & 1 & 1 & 3 \\ & & & & 1 & 0 \\ & & & & & 1 \end{bmatrix} \quad \begin{bmatrix} -4 \\ 4 \\ -35 \\ 6 \\ -19 \\ 7 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 0 & -10 & -3 & 0 & 0 \\ & 1 & 4 & 1 & 0 & 0 \\ & & 1 & 0 & 0 & 0 \\ & & & 1 & 0 & 0 \\ & & & & 1 & 0 \\ & & & & & 1 \end{bmatrix} \quad \begin{bmatrix} -4 \\ 4 \\ 15 \\ 4 \\ -19 \\ 7 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ & 1 & 0 & 0 & 0 & 0 \\ & & 1 & 0 & 0 & 0 \\ & & & 1 & 0 & 0 \\ & & & & 1 & 0 \\ & & & & & 1 \end{bmatrix} \quad \begin{bmatrix} 158 \\ -60 \\ 15 \\ 4 \\ -19 \\ 7 \end{bmatrix}$$

Figure 7.4

Numerical Example for N = 6 and m = 2

8. CONCLUSIONS

In this paper we have considered the solution to triangular linear systems of equations. For a general triangular system, we derived two methods for solution both of which require $O(\log^2 N)$ time and $O(N^3)$ processors, and a solution method which requires $O(\sqrt{N} \log N)$ time but only $O(N^2)$ processors. Table 8.1 summarizes the execution time and processor requirements for various triangular linear systems solvers. One of these methods is developed by applying the principle of recursive doubling to a recurrence relation equivalent to the original triangular system and another is developed by applying matrix operations to the original matrix. The best prior method known to this author is that of Heller [1973] which required $O(\log^2 N)$ time and $O(N^4)$ processors.

We also considered the solution to band triangular systems. By considering the equivalent recurrence relation, we derived a method requiring $O((\log N)(\log m))$ time and $O(Nm^2)$ processors, where m is the bandwidth of the matrix. Algorithms with different processor and execution time requirements also exist for this problem but are not discussed here.

Recent work by Hyafil and Kung [1974a, 1974b] has resulted in the development of a class of algorithms for the solution of this problem. In particular, they describe

an algorithm using $O(n^2)$ processors that requires only $O(n^{1/3} \log^2 n)$ time. They also develop a more unified method of reducing the numbers of processors used in an algorithm. The method we used in Section 6 seems to be a hybrid of the two methods they propose.

Table 8.1

Solution Methods for Full Triangular Systems

Method	Processors	Time
Serial	1	N^2
Straightforward Parallel	N	N
Section 6	N^2	$\sqrt{N} \log N$
Sections 4 and 5	N^3	$\log^2 N$

9. ACKNOWLEDGMENT

The author expresses his appreciation to his dissertation advisor, Professor Harold S. Stone, for his suggestions and criticism, and to Bell Laboratories for the financial support which made this work possible.

BIBLIOGRAPHY

Barnes, G. H., et al., 1968. "The ILLIAC IV Computer," IEEE Transactions on Computers, Vol. C-17, No. 8, pp. 746-757, August 1968.

Flynn, M. J., 1966. "Very High-Speed Computing Systems," Proceedings of the IEEE, Vol. 54, No. 12, pp. 1901-1909, December 1966.

Gilmore, P. A., 1968. "Structuring of Parallel Algorithms," Journal of the ACM, Vol. 15, No. 2, pp. 176-192, April 1968.

Heller, D., 1973. "A Determinant Theorem with Applications to Parallel Algorithms," Department of Computer Science, Carnegie-Mellon University, Pittsburgh, Pennsylvania, March 1973.

Heller, D., 1974. "On the Efficient Computation of Recurrence Relations," ICASE Report, June 13, 1974.

Hyafil, L. and Kung, H. T., 1974a. "The Complexity of Parallel Evaluation of Linear Recurrences," Department of Computer Science, Carnegie-Mellon University, November 1974.

Hyafil, L. and Kung, H. T., 1974b. "Parallel Algorithms for Solving Triangular Systems with Small Parallelism," Department of Computer Science, Carnegie-Mellon University, December 1974.

Kogge, P. M., 1974. "Parallel Algorithms for the Efficient Solution 0f Recurrence Problems," IBM Journal of Research and Development, Vol. 1.8, No. 2, pp. 138-148, March 1974.

Kuck, D., 1968. "ILLIAC IV Software and Applications Programming," IEEE Transactions on Computers, Vol. C-17, No. 8, pp. 758-770, August 1968.

Ostrowski, A., 1936. "Sur une Transformation de la Série de Liouville-Neumann," Comptes Rendes, Vol. 203, pp. 602-604, 1936. See also A. Ostrowski, "On Trends and Problems in Numerical Approximation," in On Numerical Approximation, R. E. Langer, University of Wisconsin Press, 1959.

Stone, H. S., 1973. "An Efficient Parallel Algorithm for the Solution of a Tridiagonal Linear System of Equations," Journal of the ACM, Vol. 20, No. 1, pp. 27-38, January 1973.