MATHEMATICS OF COMPUTATION, VOLUME 33, NUMBER 145
JANUARY 1979, PAGES 185–199

A Parallel Algorithm for Solving General Tridiagonal Equations

By Paul N. Swarztrauber*

Abstract. A parallel algorithm for the solution of the general tridiagonal system is presented. The method is based on an efficient implementation of Cramer's rule, in which the only divisions are by the determinant of the matrix. Therefore, the algorithm is defined without pivoting for any nonsingular system. $O(n)$ storage is required for n equations and $O(\log n)$ operations are required on a parallel computer with n processors. $O(n)$ operations are required on a sequential computer. Experimental results are presented from both the CDC 7600 and CRAY-1 computers.

1. Introduction. Currently, several parallel algorithms exist for a wide class of tridiagonal systems. Golub developed the method of cyclic reduction for solving large block tridiagonal systems which resulted from the discretization of Poisson's equation on a rectangle. Although the algorithm was unstable for this application, it could be used to solve the scalar constant-coefficient tridiagonal systems; and it could be efficiently implemented on a parallel computer [5], [6]. Later, Buneman stabilized the Golub algorithm [1], [2] which provided another algorithm for solving the constant-coefficient tridiagonal system. In [10], [11], Stone describes his "recursive doubling" algorithm and generalizes the cyclic reduction algorithm so that both can be used to solve the nonconstant coefficient problem. In [11] Stone compares the arithmetic complexity of all three algorithms. In [12], Swarztrauber presents an algorithm for solving block tridiagonal systems that result from separable elliptic partial differential equations; and its scalar analogue provides yet another algorithm for solving the nonconstant coefficient system.

Each of the algorithms discussed so far is direct in the sense that if exact arithmetic were used then an exact solution would be obtained with a finite number of operations. Traub [3], [13] describes a parallel method for improving an approximate solution which can be applied repeatedly to obtain the desired accuracy. Lambiotte and Voigt [8] discuss the implementation of Traub's method on the Control Data STAR-100, together with Gaussian elimination, cyclic reduction, recursive doubling, successive relaxation, and Jacobi's method.

As Stone notes in [11], each of the algorithms mentioned above requires the system to be diagonally dominant. If the system does not have this structure, then

Received June 8, 1978.
AMS (MOS) subject classifications (1970). Primary 65F05, 68A20; Secondary 15A15.
Key words and phrases. Tridiagonal matrices, parallel algorithms, linear equations.
*National Center for Atmospheric Research, Boulder, Colorado 80307, which is sponsored by the National Science Foundation.

0025-5718/79/0000-0012/$05.00

the algorithm may not be stable, or it may not be defined if a division by zero occurs. The general tridiagonal system can be solved using Gaussian elimination with partial pivoting. However, as Lambiotte and Voigt observe, there is currently no known way for incorporating a pivot strategy into existing parallel algorithms [8].

In this paper a parallel algorithm is presented which is based on an efficient implementation of Cramer's rule. The only division is by the determinant of the matrix, with the result that the algorithm is defined without pivoting for any nonsingular system. $O(n)$ storage is required for n equations, and $O(\log n)$ operations are required on a parallel computer with n processors. $O(n)$ operations are required on a sequential computer. The formal development of the algorithm is given in Section 2; and certain computational aspects are considered in Section 3 including complexity, accuracy, scaling, timing and storage.

2. **The Algorithm.** We wish to solve a linear system of equations

$$\mathbf{Ax} = \mathbf{y}, \tag{1}$$

where the matrix $\mathbf{A}$ is tridiagonal

$$\mathbf{A} = \begin{bmatrix} b_1 & c_1 & & & \\ a_2 & \cdot & & & \\ & \cdot & & & \\ & & \cdot & c_{n-1} & \\ & & a_n & b_n & \end{bmatrix}, \quad \text{and} \quad \mathbf{y} = \begin{bmatrix} y_1 \\ \cdot \\ \cdot \\ \cdot \\ y_n \end{bmatrix}, \quad \mathbf{x} = \begin{bmatrix} x_1 \\ \cdot \\ \cdot \\ \cdot \\ x_n \end{bmatrix}. \tag{2}$$

Since the algorithm is based on Cramer's rule, we begin by defining the following determinants and related quantities which will be used in the theorem given below.

$$b_\nu^{(\mu)} = \begin{vmatrix} b_\nu & c_\nu & & & \\ a_{\nu+1} & \cdot & & & \\ & \cdot & & & \\ & & \cdot & c_{\mu-1} & \\ & & a_\mu & b_\mu & \end{vmatrix}, \tag{3}$$

$$d_i = b_1^{(i)}, \tag{4}$$

$$g_i = -a_i b_{i+1}^{(n)}, \tag{5}$$

$$(6) \qquad s_i = \begin{vmatrix} b_1 & c_1 & & & y_1 \\ a_2 & \cdot & & & \cdot \\ & \cdot & & & \cdot \\ & & & \cdot & y_{i-1} \\ & & & a_i & y_i \end{vmatrix},$$

$$(7) \qquad u_i = \begin{vmatrix} y_i & c_i & & & \\ y_{i+1} & \cdot & & & \\ \cdot & & \cdot & & \\ \cdot & & & \cdot & c_{n-1} \\ y_n & & & a_n & b_n \end{vmatrix}.$$

We now state and prove the theorem which is fundamental to the algorithm.

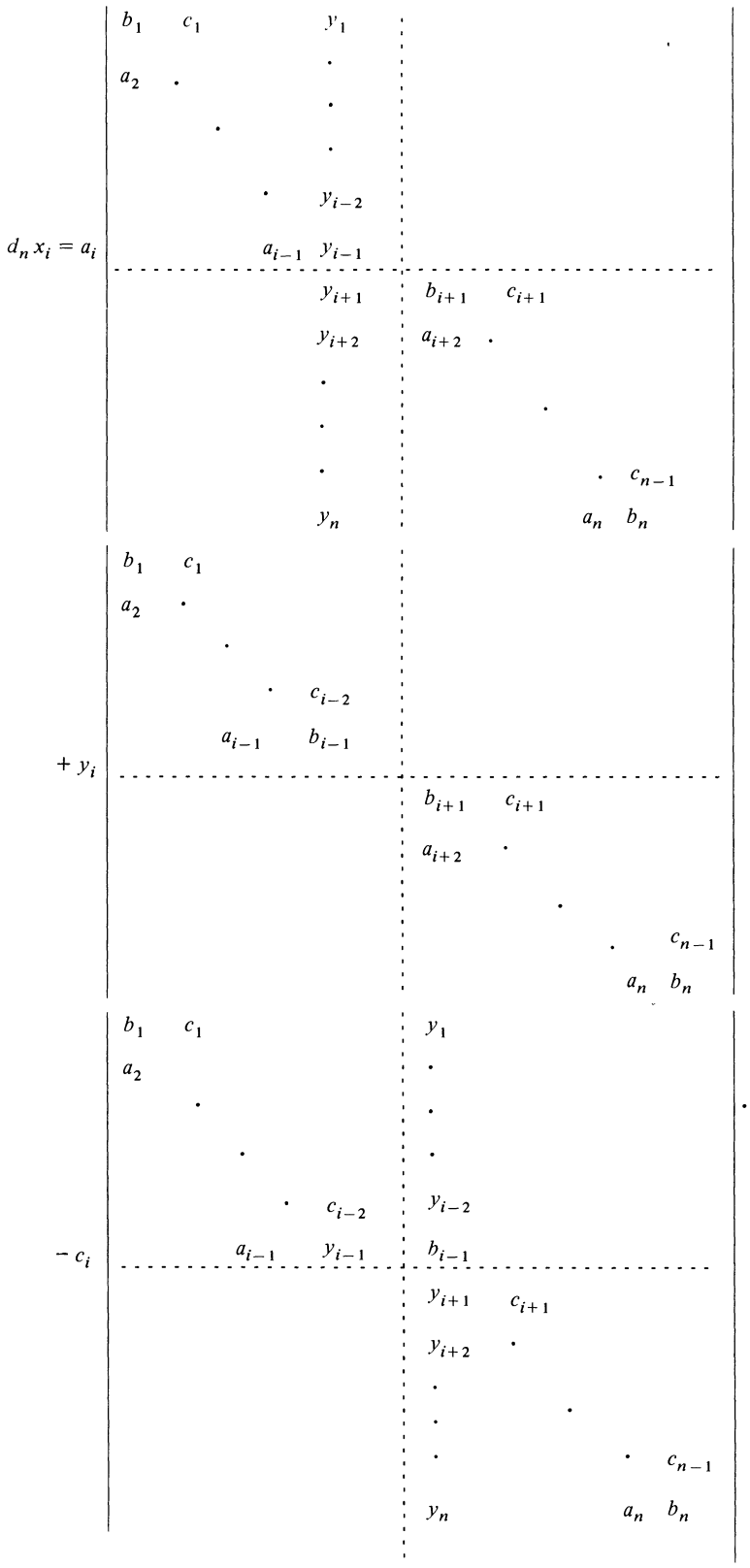
THEOREM. *Given the sequences d_i, g_i, s_i and u_i defined above, then the solution x_i is given by*

$$(8) \qquad x_i = d_n^{-1}(g_i s_{i-1} + d_{i-1} u_i).$$

Proof. From Cramer's rule

$$d_n x_i = \begin{vmatrix} b_1 & c_1 & & & y_1 & & & \\ & & & & \cdot & & & \\ a_2 & & & & \cdot & & & \\ & \cdot & & & \cdot & & & \\ & & \cdot & & \cdot & & & \\ & & & \cdot & y_{i-1} & & & \\ & & & a_i & y_i & c_i & & \\ & & & & y_{i+1} & \cdot & & \\ & & & & \cdot & & \cdot & \\ & & & & \cdot & & & \\ & & & & \cdot & & & \cdot \\ & & & & & & & c_{n-1} \\ & & & & y_n & & a_n & b_n \end{vmatrix}.$$

Expanding by the ith row,


b_1 c_1 y_1
a_2
y_{i-2}
$d_n x_i = a_i$ a_{i-1} y_{i-1}
y_{i+1} b_{i+1} c_{i+1}
y_{i+2} a_{i+2}
c_{n-1}
y_n a_n b_n
b_1 c_1
a_2
c_{i-2}
a_{i-1} b_{i-1}
$+ y_i$
b_{i+1} c_{i+1}
a_{i+2}
c_{n-1}
a_n b_n
b_1 c_1 y_1
a_2
c_{i-2} y_{i-2}
$- c_i$ a_{i-1} y_{i-1} b_{i-1}
y_{i+1} c_{i+1}
y_{i+2}
c_{n-1}
y_n a_n b_n

Since the determinant of a block triangular matrix is the product of the determinants of the blocks on the diagonal,

$$d_n x_i = - a_i s_{i-1} b_{i+1}^{(n)} + y_i d_{i-1} b_{i+1}^{(n)} - c_i d_{i-1} u_{i+1}, \tag{9}$$

$$d_n x_i = g_i s_{i-1} + d_{i-1} (y_i b_{i+1}^{(n)} - c_i u_{i+1}). \tag{10}$$

Expanding u_i, given in (7), in terms of its first row, we obtain

$$u_i = - c_i u_{i+1} + b_{i+1}^{(n)} y_i, \tag{11}$$

which substituted into (10), yields the desired result (8) and completes the proof of the theorem.

The algorithm consists of two parts which have two and three steps, respectively.

I. Initialization

Ia. Reduction

Ib. Backsubstitution

II. Solution

IIa. Reduction

IIb. Backsubstitution

IIc. Combination

Each of these steps is developed throughout the remainder of this section, and examples for the case $n = 8$ are given. The complete algorithm is summarized at the end of the section. Part I consists of calculations which are independent of the right side, y_i, including the sequences d_i and g_i. Part II consists of all calculations which depend on y_i, including the sequences s_i and u_i.

We begin with part I and, in particular, with the calculation of d_i. Expanding d_i, given by (4) in terms of its ith row we obtain

$$d_i = b_i d_{i-1} - a_i c_{i-1} d_{i-2}, \tag{12}$$

which, together with $d_0 = 1$ and $d_1 = b_1$, determines the sequence d_i. This recurrence is not suitable for parallel computations; and, thus, we could consider using one of several related methods [4], [7], [11] which are available for the parallel computation of d_i. However, the direct application of these methods would be inefficient and it will be shown that the sequences d_i and g_i can both be determined with about the same amount of computation that would be required to determine either one separately. Writing (12) in matrix form, we obtain

$$\begin{bmatrix} d_i \\ d_{i-1} \end{bmatrix} = \begin{bmatrix} b_i & - a_i c_{i-1} \\ 1 & 0 \end{bmatrix} \begin{bmatrix} d_{i-1} \\ d_{i-2} \end{bmatrix}. \tag{13}$$

If we define the sequence

$$e_i = c_i d_{i-1} \tag{14}$$

and substitute into (13), then

$$\begin{bmatrix} d_i \\ e_i \end{bmatrix} = \begin{bmatrix} b_i & -a_i \\ c_i & 0 \end{bmatrix} \begin{bmatrix} d_{i+1} \\ e_{i+1} \end{bmatrix}. \tag{15}$$

In order to determine a similar matrix recurrence for g_i we must first define

$$f_i = b_i^{(n)}. \tag{16}$$

Referring to definition (3), we can expand f_i in terms of a_i, b_i and c_i,

$$f_i = b_i f_{i+1} - a_{i+1}\, c_i f_{i+2}, \tag{17}$$

which has the matrix form:

$$\begin{bmatrix} f_i \\ f_{i+1} \end{bmatrix} = \begin{bmatrix} b_i & -a_{i+1}\, c_i \\ 1 & 0 \end{bmatrix} \begin{bmatrix} f_{i+1} \\ f_{i+2} \end{bmatrix}. \tag{18}$$

From (5) and (16) we see that $g_i = -a_i f_{i+1}$ which substituted into (8) yields

$$\begin{bmatrix} f_i \\ g_i \end{bmatrix} = \begin{bmatrix} b_i & c_i \\ -a_i & 0 \end{bmatrix} \begin{bmatrix} f_{i+1} \\ g_{i+1} \end{bmatrix}. \tag{19}$$

Further, if we define

$$\mathbf{Q}_i = \begin{bmatrix} b_i & c_i \\ -a_i & 0 \end{bmatrix}, \quad \mathbf{e} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \tag{20}$$

$$\mathbf{w}_i = \begin{bmatrix} d_i \\ e_i \end{bmatrix}, \quad \mathbf{z}_i = \begin{bmatrix} f_i \\ g_i \end{bmatrix}; \tag{21}$$

then from (15)

$$\mathbf{w}_i = \mathbf{Q}_i^T \mathbf{Q}_{i-1}^T \cdots \mathbf{Q}_1^T \mathbf{e}, \tag{22}$$

or

$$\mathbf{w}_i^T = \mathbf{e}^T \mathbf{Q}_1 \mathbf{Q}_2 \cdots \mathbf{Q}_i \tag{23}$$

and from (19)

$$\mathbf{z}_i = \mathbf{Q}_i \mathbf{Q}_{i+1} \cdots \mathbf{Q}_n \mathbf{e}. \tag{24}$$

Once the vectors $\mathbf{w}_i$ and $\mathbf{z}_i$ are computed then the sequences d_i and g_i are available for computing the solution using (8). The matrix recurrence expressions (15) and (18) appear to be quite independent of one another since (15) would be computed in the direction of increasing index i and (19) would be computed in the direction of decreasing index i. However, in both (23) and (24) the matrix factors occur in the same order, and therefore, the products that are required to compute $\mathbf{w}_i$ can also be used to compute $\mathbf{z}_i$. To evaluate these products we use the method of recursive doubling due to Stone [11]. We will illustrate the parallel algorithm for computing $\mathbf{w}_i$ and $\mathbf{z}_i$ for the case $n = 8$. For arbitrary ν and μ we first define

$$\mathbf{Q}_\nu^{(\mu)} = \mathbf{Q}_\nu \mathbf{Q}_{\nu+1} \cdots \mathbf{Q}_\mu. \tag{25}$$

In the algorithm given below, the quantities in each cycle can be computed in parallel.

I. *Initialization.*

Ia. *Reduction (example).*

Cycle 1 $\mathbf{Q}_1^{(2)} = \mathbf{Q}_1\mathbf{Q}_2$; $\mathbf{Q}_3^{(4)} = \mathbf{Q}_3\mathbf{Q}_4$; $\mathbf{Q}_5^{(6)} = \mathbf{Q}_5\mathbf{Q}_6$; $\mathbf{Q}_7^{(8)} = \mathbf{Q}_7\mathbf{Q}_8$,

Cycle 2 $\mathbf{Q}_1^{(4)} = \mathbf{Q}_1^{(2)}\mathbf{Q}_3^{(4)}$; $\mathbf{Q}_5^{(8)} = \mathbf{Q}_5^{(6)}\mathbf{Q}_7^{(8)}$,

Cycle 3 $\mathbf{Q}_1^{(8)} = \mathbf{Q}_1^{(4)}\mathbf{Q}_5^{(8)}$.

The following elements in the sequences $\mathbf{w}_i$ and $\mathbf{z}_i$ are directly available:

$$\mathbf{w}_1^T = \mathbf{e}^T\mathbf{Q}_1; \quad \mathbf{w}_2^T = \mathbf{e}^T\mathbf{Q}_1^{(2)}; \quad \mathbf{w}_4^T = \mathbf{e}^T\mathbf{Q}_1^{(4)}; \quad \mathbf{w}_8^T = \mathbf{e}^T\mathbf{Q}_1^{(8)},$$

$$\mathbf{z}_8 = \mathbf{Q}_8\mathbf{e}; \quad \mathbf{z}_7 = \mathbf{Q}_7^{(8)}\mathbf{e}; \quad \mathbf{z}_5 = \mathbf{Q}_5^{(8)}\mathbf{e}; \quad \mathbf{z}_1 = \mathbf{Q}_1^{(8)}\mathbf{e}.$$

The remaining values are computed from the expressions below which make extensive use of the relations

$$\mathbf{w}_\mu^T = \mathbf{w}_\delta^T\mathbf{Q}_{\delta+1}^{(\mu)}, \qquad \nu < \delta < \mu, \tag{26}$$

$$\mathbf{z}_\nu = \mathbf{Q}_\nu^{(\delta-1)}\mathbf{z}_\delta, \tag{27}$$

which are obtained from (23), (24), and the associative property of matrix multiplication.

Ib. *Backsubstitution (example).*

Cycle 1 $\mathbf{w}_6^T = \mathbf{w}_4^T\mathbf{Q}_5^{(6)}$; $\mathbf{z}_3 = \mathbf{Q}_3^{(4)}\mathbf{z}_5$,

Cycle 2 $\mathbf{w}_3^T = \mathbf{w}_2^T\mathbf{Q}_3$; $\mathbf{w}_5^T = \mathbf{w}_4^T\mathbf{Q}_5$; $\mathbf{w}_7^T = \mathbf{w}_6^T\mathbf{Q}_7$,

$\mathbf{z}_6 = \mathbf{Q}_6\mathbf{z}_7$; $\mathbf{z}_4 = \mathbf{Q}_4\mathbf{z}_5$; $\mathbf{z}_2 = \mathbf{Q}_2\mathbf{z}_3$.

The sequences d_i and g_i, $i = 1, \ldots, 8$, are now available from the components of $\mathbf{w}_i$ and $\mathbf{z}_i$, respectively, for use in (8).

We now direct our attention to part II of the algorithm and the calculation of the remaining quantities on the right side of (8), namely the sequences s_i and u_i. If we expand s_i, given by (6) in terms of a_i and y_i, we obtain

$$s_i = -a_i s_{i-1} + d_{i-1} y_i. \tag{28}$$

If we define $\alpha_{i+1}^{(i)} = \lambda_{i+1}^{(i)} = 1$; $\alpha_i^{(i)} = -a_i$; $\lambda_i^{(i)} = -c_i$; and

$$\alpha_\nu^{(\mu)} = \prod_{j=\nu}^{\mu} a_j, \quad \lambda_\nu^{(\mu)} = \prod_{j=\nu}^{\mu} c_j, \tag{29}$$

then the solution of (28) is

$$s_i = \sum_{j=1}^{i} (-1)^{i-j} \alpha_{j+1}^{(i)} d_{j-1} y_j. \tag{30}$$

However, this form is not suitable for parallel computation. If we define

$$s_\nu^{(\mu)} = \sum_{j=\nu}^{\mu} (-1)^{\mu-j} \alpha_{j+1}^{(\mu)} d_{j-1} y_j, \tag{31}$$

then for any $\nu < \delta < \mu$,

$$s_\nu^{(\mu)} = \sum_{j=\nu}^{\delta} (-1)^{\mu-j} \alpha_{j+1}^{(\mu)} d_{j-1} y_j + \sum_{j=\delta+1}^{\mu} (-1)^{\mu-j} \alpha_{j+1}^{(\mu)} d_{j-1} y_j. \tag{32}$$

$$s_\nu^{(\mu)} = (-1)^{\mu-\delta} \alpha_{\delta+1}^{(\mu)} \sum_{j=\nu}^{\delta} (-1)^{\delta-j} \alpha_{j+1}^{(\delta)} d_{j-1} y_j + s_{\delta+1}^{(\mu)}. \tag{33}$$

$$s_\nu^{(\mu)} = (-1)^{\mu-\delta} \alpha_{\delta+1}^{(\mu)} s_\nu^{(\delta)} + s_{\delta+1}^{(\mu)}. \tag{34}$$

This result is used extensively in the parallel calculation of s_i, which is described below. The calculation of u_i is similar to that of s_i. Starting with (11), which has the solution

$$u_i = \sum_{j=i}^{n} (-1)^{j-1} \lambda_i^{(j-1)} f_{j+1} y_j, \tag{35}$$

then we define

$$u_\nu^{(\mu)} = \sum_{j=\nu}^{\mu} (-1)^{j-\nu} \lambda_\nu^{(j-1)} f_{j+1} y_j, \tag{36}$$

from which we obtain

$$u_\nu^{(\mu)} = u_\nu^{(\delta-1)} + (-1)^{\delta-\nu} \lambda_\nu^{(\delta-1)} u_\delta^{(\mu)}. \tag{37}$$

The algorithm for computing s_i and u_i will be illustrated for the case $n = 8$. It is similar to the calculation of $\mathbf{w}_i$ and $\mathbf{z}_i$ in the sense that it has two steps, namely,

reduction and backsubstitution. The goal is to compute $s_1^{(i)}$ and $u_i^{(n)}$, which from (30), (31), (35), and (36) are just s_i and u_i, respectively. The starting values $s_i^{(i)} = d_{i-1}y_i$ and $u_i^{(i)} = f_{i+1}y_i$ are given by (31) and (36), respectively. The expressions in each cycle are derived from (34) and (37) and can be computed in parallel.

II. *Solution.*

IIa. *Reduction (example).*

$$\begin{aligned}
\text{Cycle 1}\quad & s_1^{(2)} = s_2^{(2)} - a_2 s_1^{(1)}; \quad u_1^{(2)} = u_i^{(1)} - c_1 u_2^{(2)},\\
& s_3^{(4)} = s_4^{(4)} - a_4 s_3^{(3)}; \quad u_3^{(4)} = u_3^{(3)} - c_3 u_4^{(4)},\\
& s_5^{(6)} = s_6^{(6)} - a_6 s_5^{(5)}; \quad u_5^{(6)} = u_5^{(5)} - c_5 u_6^{(6)},\\
& s_7^{(8)} = s_8^{(8)} - a_8 s_7^{(7)}; \quad u_7^{(8)} = u_7^{(7)} - c_7 u_8^{(8)}.\\
\text{Cycle 2}\quad & s_1^{(4)} = s_3^{(4)} + \alpha_3^{(4)} s_1^{(2)}; \quad u_1^{(4)} = u_1^{(2)} + \lambda_1^{(2)} u_3^{(4)},\\
& s_5^{(8)} = s_7^{(8)} + \alpha_7^{(8)} s_5^{(6)}; \quad u_5^{(8)} = u_5^{(6)} + \lambda_5^{(6)} u_7^{(8)}.\\
\text{Cycle 3}\quad & s_1^{(8)} = s_5^{(8)} + \alpha_5^{(8)} s_1^{(4)}; \quad u_1^{(8)} = u_1^{(4)} + \lambda_1^{(4)} u_5^{(8)}.
\end{aligned}$$

The elements $s_1 = s_1^{(1)}$, $s_2 = s_1^{(2)}$, $s_4 = s_1^{(4)}$, $s_8 = s_1^{(8)}$, $u_8 = u_8^{(8)}$, $u_7 = u_7^{(8)}$, $u_5 = u_5^{(8)}$, and $u_1 = u_1^{(8)}$ are available from the reduction step IIa. The remaining elements are computed in the backsubstitution step IIb.

IIb. *Backsubstitution (example).*

$$\begin{aligned}
\text{Cycle 1}\quad & s_1^{(6)} = s_5^{(6)} + \alpha_5^{(6)} s_1^{(4)}; \quad u_3^{(8)} = u_3^{(4)} + \lambda_3^{(4)} u_5^{(8)},\\
& s_1^{(3)} = s_3^{(3)} - a_3 s_1^{(2)}; \quad u_2^{(8)} = u_2^{(2)} - c_2 u_3^{(8)},\\
& s_1^{(5)} = s_5^{(5)} - a_5 s_1^{(4)}; \quad u_4^{(8)} = u_4^{(4)} - c_4 u_5^{(8)},\\
& s_1^{(7)} = s_7^{(7)} - a_7 s_1^{(6)}; \quad u_6^{(8)} = u_6^{(6)} - c_6 u_7^{(8)}.
\end{aligned}$$

Once the sequences g_i, d_i, s_i, and u_i are computed, then the solution x_i can be computed from (8).

The algorithm for general n is given below.

I. *Initialization.*

Ia. *Reduction.* Starting with

$$(38)\qquad \mathbf{Q}_i^{(i)} = \begin{bmatrix} b_i & c_i \\ & \\ -a_i & 0 \end{bmatrix}; \quad \alpha_i^{(i)} = -a_i; \quad \lambda_i^{(i)} = -c_i,$$

then for $j = 1, \ldots, k = \log_2 n$ and $i = 2^j, 2 \cdot 2^j, \ldots, n - 2^j$ compute

(39a) $$\alpha_{i+1}^{(i+2l)} = \alpha_{i+1}^{(i+l)}\alpha_{i+l+1}^{(i+2l)},$$

(39b) $$\lambda_{i+1}^{(i+2l)} = \lambda_{i+1}^{(i+l)}\lambda_{i+l+1}^{(i+2l)}, \qquad l = 2^j,$$

(39c) $$\mathbf{Q}_{i+1}^{(i+2l)} = \mathbf{Q}_{i+1}^{(i+l)}\mathbf{Q}_{i+l+1}^{(i+2l)}.$$

Ib. *Backsubstitution.* Starting with

(40a) $$\mathbf{w}_l^T = \mathbf{e}^T\mathbf{Q}_1^{(l)}, \qquad l = 2^j,\ j = 0, \ldots, k,$$

(40b) $$\mathbf{z}_{n-l+1} = \mathbf{Q}_{n-l+1}^{(n)}\mathbf{e},$$

then for $j = k - 1, \ldots, 1$ and $i = 2^j, 2 \cdot 2^j, \ldots, n - 2^j$ compute

(41a) $$\mathbf{w}_{i+l}^T = \mathbf{w}_i^T\mathbf{Q}_{i+1}^{(i+l)},$$

(41b) $$\mathbf{z}_{i-l+1} = \mathbf{Q}_{i-l+1}^{(i)}\mathbf{z}_{i+1}.$$

II. *Solution.*

IIa. *Reduction.* Starting with $s_i^{(i)} = d_{i-1}y_i$ and $u_i^{(i)} = f_{i+1}y_i$, then for $j = 1, \ldots, k$ and $i = 2^j, 2 \cdot 2^j, \ldots, n - 2^j$ compute

(42a) $$s_{i+1}^{(i+2l)} = s_{i+l+1}^{(i+2l)} + \alpha_{i+l+1}^{(i+2l)}s_{i+1}^{(i+l)}, \qquad l = 2^j,$$

(42b) $$u_{i+1}^{(i+2l)} = u_{i+1}^{(i+l)} + \lambda_{i+1}^{(i+l)}u_{i+l+1}^{(i+2l)}.$$

IIb. *Backsubstitution.* For $j = k - 1, \ldots, 1$ and $i = 2^j, 2 \cdot 2^j, \ldots, n - 2^j$

(43a) $$s_1^{(i+1)} = s_{i+1}^{(i+l)} + \alpha_{i+1}^{(i+l)}s_1^{(i)},$$

(43b) $$u_{i-l+1}^{(n)} = u_{i-l+1}^{(i)} + \lambda_{i-l+1}^{(i)}u_{i+1}^{(n)}.$$

IIIb. *Combination.*

(44) $$x_i = (d_n)^{-1}(g_i s_{i-1} + d_{i-1}u_i).$$

where $d_i = \mathbf{w}_i^T\mathbf{e}$, $g_i = (0, 1)\mathbf{z}_i$, $x_1 = u_1/d_n$ and $x_n = s_n/d_n$.

3. Computational Notes

A. *Operation Counts.* The total number of operations for each step is given first, which provides an estimate of the computational effort required on a sequential or vector machine. Asymptotic operation counts are given, which include only those contributions that are proportional to n. The letters M, A, and D refer to multiplications, additions, and divisions, respectively. Thus, for example, the number of operations in Step Ia is $8.5n$ multiplications plus $2.5n$ additions.

Step Ia	$n(8.5M + 2.5A)$	Initialization $n(15.5M + 5.5A)$	Total $n(23.5M + 11.5A + D)$
Step IB	$n(7M + 3A)$		
Step IIa	$n(4M + 2A)$	Solution $n(8M + 5A + D)$	
Step IIb	$n(2M + 2A)$		
Step IIc	$n(2M + A + D)$		

The initialization requires $n(15.5M + 5.5A)$ operations. If the matrix **A** remains unchanged, then additional solutions corresponding to different right-hand sides **y** can be obtained with $n(8M + 5A + D)$ operations. The total number of operations is $n(23.5M + 8.5A + D)$. This number can be compared with the counts that Stone [11] gives: $n(13M + 6A + D)$, $n(15M + 10A + 2D)$, and $n(22M + 11A + 4D)$ for cyclic reduction, Buneman's algorithm, and recursive doubling, respectively.

Now, recalling that $k = \log_2 n$, on a parallel computer with $n/2$ processors the operation counts are

Step Ia	$k(10M + 4A)$	Initialization $k(18M + 8A)$	Total $k(26M + 14A + 2D)$
Step Ib	$k(8M + 4A)$		
Step IIa	$k(2M + 2A)$	Solution $k(8M + 6A + 2D)$	
Step IIb	$k(2M + 2A)$		
Step IIc	$2k(2M + A + D)$		

Asymptotic operation counts are given which include the terms that are proportional to k. The operation count can be reduced further by increasing the number of processors, since additional computations can be done in parallel. For example, all ten multiplications in Step Ia could be done simultaneously. With $5n$ processors, the total operation count would be $k(5M + 5A + D)$.

B. *Accuracy.* Because of the well-known accuracy problems with both cyclic reduction and Cramer's rule for solving 2×2 systems [9], the accuracy of this algorithm is of considerable interest. In this subsection the accuracy will be examined empirically via several numerical experiments which suggest that it is quite stable. In the course of experimentation it was observed that in certain cases the accuracy could be improved by the double precision summation of the scalar products in Step Ia only. In what follows we will refer to the algorithm with this addition as the *modified* algorithm. Although satisfactory results are obtained without this modification, it may nevertheless be desirable for applications in which the matrix remains unchanged and only the right side varies. In Table 1 below, the error is given for both the modified and the unmodified algorithm.

Results are presented for five problems; the first four solve the system with $a_i = c_i = -1$ and $b_i = b = 0, .5, 1, 2,$ and 4. The determinants d_i are periodic as a function of i for $b < 2$, linear in i for $b = 2$, the exponential for $b > 2$. Hence, the problems which have been selected are representative. For the fifth

problem a solution is obtained with the coefficient matrix given below, which will be referred to as

Matrix A.

$$b_1 = 2(4n^2 + 6n - 7),$$

$$c_1 = -(2n - 3)(2n + 6),$$

$$a_i = -[2(n + i) + 1][2(n - i) + 2],$$

$$c_i = -[2(n + i) + 4][2(n - i) - 1], \qquad i = 2, \ldots, n - 1,$$

$$b_i = -(a_i + c_i - 4),$$

$$a_n = -8n - 2,$$

$$b_n = 4n + 2.$$

These coefficients originate in the process of computing the Fourier coefficients of the Legendre polynomials. The system differs from the system considered above since it is not symmetric, and the coefficients vary over three orders of magnitude for $n = 1024$.

In all cases the coefficients a_i, b_i, and c_i are perturbed randomly in about the 12th decimal digit to induce roundoff error. This is necessary since the coefficients are integers, and only multiplications and additions occur in part I. This is particularly true for the case $a_i = c_i = -1$ and $b_i = 2$, which if not perturbed, yields quite accurate but unrepresentative results. All experiments were performed on the NCAR Control Data 7600, which has a relative accuracy of 7.1×10^{-15}.

The entries in Table 1 are the relative maximum errors given by

$$e = \frac{\max_i |\hat{x}_i - x_i|}{\max_i |x_i|},$$

where $\hat{x}_i$ is the computed solution and x_i is the "exact" (double precision) solution.

TABLE 1

The Accuracy of the Parallel, Modified Parallel, and Gaussian Elimination Algorithms; $n = 1024$

	Parallel Tridiagonal	Modified Parallel	Gaussian Elimination
$b = 0$	4.0×10^{-12}	1.1×10^{-12}	1.6×10^{-12}
$p = .5$	4.3×10^{-12}	1.6×10^{-12}	7.1×10^{-12}
$b = 1.$	1.2×10^{-12}	1.7×10^{-13}	2.8×10^{-12}
$b = 2.$	3.3×10^{-8}	4.5×10^{-10}	1.7×10^{-10}
$b = 4.$	4.0×10^{-14}	6.5×10^{-14}	2.3×10^{-14}
Matrix A	3.9×10^{-8}	5.9×10^{-10}	1.3×10^{-9}

The error was also determined for a number of other systems when the coefficients were computed randomly. In general, the accuracy of the unmodified parallel algorithm was comparable to that of Gauss elimination with partial pivoting, and in the majority of the cases the modified parallel algorithm was superior to Gauss elimination

C. *Scaling.* It is possible, even for matrices of modest order, for the determinant to either under- or overflow the computer, depending on the magnitude of the coefficients. Therefore, scaling may be necessary, either implicitly in the formulation of the problem or explicitly at the time of solution. At the current state of the art and for most applications, the number of equations is sufficiently small that scaling could be performed before solution. However, for large systems or for any general-purpose software an adaptive scaling procedure would be desirable. A scaling procedure was implemented, and its effect on the computing time is given in Table 2. The scaling can be done in the initialization part, so that the time required to obtain a solution is the same as the unscaled algorithm.

D. *Timing.* In Table 2 below, the times for the parallel algorithm and its variants are compared with Gaussian elimination with partial pivoting. All programs are written in FORTRAN with the exception of two function subprograms which extract and modify the exponent in the program which includes scaling. The modified parallel algorithm is all in FORTRAN, and the calculations in Step Ia are performed in double precision but with single precision storage.

The times given in Tables 2 and 3 for the parallel algorithm are not representative of the times which would be obtained on a parallel computer with a sufficient number of processors to demonstrate the $O(\log n)$ dependence of computing time on n. However, they do provide a benchmark from which the performance of the algorithm on a parallel computer can be estimated.

TABLE 2

CDC 7600 Computation Time in Milliseconds for a Tridiagonal System With Order $n = 1024$

	Initialization	Solution	Total
Gauss elimination with partial pivoting	7.7	5.1	12.8
Parallel Algorithm	8.4	6.0	14.4
Modified Parallel	12.6	6.0	18.6
Scaled Parallel	16.2	6.0	22.2
Modified Parallel with scaling	20.4	6.0	26.4

In Table 3 below, the parallel algorithm is compared with Gauss elimination

with partial pivoting on the CRAY-1 computer. Both programs were written in FORTRAN; however, the parallel program can vectorized by the CRAY-1 computer. The CRAY-1 is not a parallel computer; and hence, the computing time for the parallel algorithm exceeds $O(\log n)$. However, the vectorization is sufficient to make it more efficient than the Gauss algorithm for n greater than 32. The figures include both the initialization and solution of the system.

TABLE 3

CRAY-1 Computation Times in Milliseconds for the Parallel and Gauss Elimination Algorithms

n	Parallel	Gauss	G/P
32	.24	.21	.87
64	.32	.43	1.34
128	.44	.86	1.95
256	.64	1.71	2.67
512	1.01	3.43	3.40
1024	1.71	6.85	4.01
2048	3.09	13.71	4.43

E. *Storage.* The implementation at NCAR of the algorithm required $6n$ locations without using the $4n$ locations required by the tridiagonal matrix and its right side. The storage was allocated in the following manner. In Step Ia the matrices $\mathbf{Q}_{i+1}^{(i+2l)}$ required $4n$ locations and the $\alpha_{i+1}^{(i+2l)}$, and $\lambda_{i+1}^{(i+2l)}$ required $2n$ locations. No additional storage is required in Step Ib where the vectors $\mathbf{w}_i$ and $\mathbf{z}_i$ overwrite the matrices computed in Ia. In Step IIa, $s_{i+1}^{(i+2l)}$ and $u_{(i+1)}^{(i+2l)}$ overwrite the e_i and x_i arrays; and finally, in step IIb the s_i and u_i overwrite the storage used in IIa. The program with scaling required an additional $3n$ locations.

National Center for Atmospheric Research
Boulder, Colorado 80307

1. O. BUNEMAN, *A Compact Non-Iterative Poisson Solver,* Rep. 294, Stanford Univ. Institute for Plasma Research, Stanford, Calif., 1969.

2. B. L. BUZBEE, G. H. GOLUB & C. W. NIELSON, "On direct methods for solving Poisson's equations," *SIAM J. Numer. Anal.,* v. 7, 1970, pp. 627–656.

3. D. E. HELLER, D. K. STEVENSON & J. F. TRAUB, *Accelerated Iterative Methods for the Solution of Tridiagonal Systems on Parallel Computers,* Dept. Computer Sci. Rep., Carnegie-Mellon Univ., Pittsburgh, Pa., 1974.

4. D. E. HELLER, "A determinant theorem with applications to parallel algorithms," *SIAM J. Numer. Anal.,* v. 11, 1974, pp. 559–568.

5. R. W. HOCKNEY, "A fast direct solution of Poisson's equation using Fourier analysis," *J. Assoc. Comput. Mach.,* v. 8, 1965, pp. 95–113.

6. R. W. HOCKNEY, "The potential calculation and some applications," *Methods of Computational Physics,* B. Adler, S. Fernback and M. Rotenberg (Eds.), Academic Press, New York, 1969, pp. 136–211.

7. P. M. KOGGE, *Parallel Algorithms for the Efficient Solution of Recurrence Problems,* Rep. 43, Digital Systems Laboratory, Stanford Univ., Stanford, Calif., 1972.

8. J. R. LAMBIOTTE, JR. & R. G. VOIGT, "The solution of tridiagonal linear systems on the CDC STAR-100 computer," *ACM Trans. Math. Software.*, v. 1, 1975, pp. 308–329.

9. C. B. MOLER, "Cramer's rule on 2-by-2 systems," *ACM SIGNUM Newsletter,* v. 9, 1974, pp. 13–14.

10. H. S. STONE, "An efficient parallel algorithm for the solution of a tridiagonal linear system of equations," *J. Assoc. Comput. Mach.*, v. 20, 1973, pp. 27–38.

11. H. S. STONE, "Parallel tridiagonal equation solvers," *ACM Trans. Math. Software.*, v. 1, 1975, pp. 289–307.

12. P. N. SWARZTRAUBER, "A direct method for the discrete solution of separable elliptic equations," *SIAM J. Numer. Anal.*, v. 11, 1974, pp. 1136–1150.

13. J. F. TRAUB, "Iterative solution of tridiagonal systems on parallel or vector computers," *Complexity of Sequential and Parallel Numerical Algorithms,* J. F. Traub (Ed.), Academic Press, New York, 1973, pp. 49–82.