NASA TECHNICAL
MEMORANDUM

NASA TM X-62,370

NASA TM X-62,370

PARALLEL TRIDIAGONAL EQUATION SOLVERS

Harold S. Stone

Ames Research Center
Moffett Field, Calif. 94035
and
Digital Systems Laboratory
Departments of Electrical Engineering & Computer Science
Stanford University
Stanford, California

April 1974

CS 422
SEL TR 79

NASA TMX-62,370

PARALLEL TRIDIAGONAL EQUATION SOLVERS

by

Harold S. Stone

Ames Research Center
Moffett Field, Calif. 94305

and

Digital Systems Laboratory

Departments of Electrical Engineering and Computer Science

Stanford University

Stanford, California

April 1974

Parallel Tridiagonal Equation Solvers

by Harold S. Stone

ABSTRACT

This paper compares three parallel algorithms for the direct solution of tridiagonal linear systems of equations. The algorithms are suitable for computers such as ILLIAC IV and CDC STAR.

For array computers similar to ILLIAC IV, cyclic odd-even reduction has the least operation count for highly structured sets of equations, and recursive doubling has the least count for relatively unstructured sets of equations. Since the difference in operation counts for these two algorithms is not substantial, their relative running times may be more related to overhead operations, which are not measured in this paper. The third algorithm, based on Buneman's Poisson solver, has more arithmetic operations than the others, and appears to be the least favorable. For pipeline computers similar to CDC STAR, cyclic odd-even reduction appears to be the most preferable algorithm for all cases.

When the tridiagonal system satisfies a strong diagonal dominance condition, the intermediate values computed by cyclic odd-even reduction form a rapidly convergent sequence, and thus the algorithm can be terminated early when values are correct to within machine accuracy. The convergence is linear until off diagonal terms fall below 1/3 the magnitude of the diagonal element, at which point the convergence becomes quadratic. The quadratic convergence is superior to the linear convergence reported by Traub for several parallel iterative tridiagonal solvers.

Parallel Tridiagonal Equation Solvers

by Harold S. Stone

I. Introduction

Within the last few years new techniques have appeared for solving tridiagonal systems of equations efficiently with a parallel processor. Cyclic odd-even reduction [Buzbee et al., 1970] is an algorithm for the direct solution of two-dimensional Poisson problems. It can be applied to one-dimensional Poisson problems as well, which are nothing more than tridiagonal linear systems of a special form. Buzbee *et al.* [1970] also describe an algorithm attributed to Buneman which solves two-dimensional Poisson problems directly. Like cyclic odd-even reduction, the Buneman algorithm can be applied to one-dimensional problems, and thus can also solve tridiagonal systems. Both of these algorithms differ from conventional tridiagonal solvers in that they are suitable for parallel computers. Stone [1973] reported a tridiagonal solver based on a recursive doubling technique and specifically designed for the ILLIAC IV. The computation time for a system of dimension N for all three algorithms is proportional to $\log_2 N$ on a parallel array computer with N processors, so that relative speed differences depend very strongly on the number of operations per iteration. A conventional tridiagonal solver requires time proportional to N on a serial machine, and cannot be run faster on a parallel machine because of the highly serial nature of the computation.

In this paper we measure the number of arithmetic operations per iteration for each of the three algorithms when applied to tridiagonal systems with varying amounts of structure. For array computers cyclic odd-even reduction and recursive doubling are relatively close in arithmetic operation count with the former preferred for highly structured systems and the latter preferred for unstructured systems. The relative preferences may change if we take into account the overhead operations such as indexing, routing, and memory accesses. Buneman's algorithm requires substantially more arithmetic operations per iteration and is the least preferable of the three algorithms for this problem.

The analysis is slightly different for pipeline computers of the CDC STAR class. For these computers computation time depends not only on the number of vector arithmetic operations but also on the total number of individual arithmetic operations. Since the recursive doubling algorithm requires $O(N \log_2 N)$ arithmetic operations as compared to $O(N)$ for cyclic odd-even reduction and Buneman's algorithm, it is quite unattractive for pipeline computers as originally formulated. In this paper we present a modification of the recursive doubling algorithm suitable for pipeline machines in that the number of arithmetic operations is reduced to $O(N)$. Nevertheless, the operation count for cyclic odd-even reduction is less for both the modified recursive doubling and Buneman algorithms.

Under conditions of diagonal dominance, cyclic odd-even reduction and Buneman's algorithm both compute intermediate values that converge to a solution, so both may be used as iterative rather than as direct methods. Again cyclic odd-even reduction appears to be superior to Buneman's algorithm. We show that convergence is linear then changes to quadratic when the magnitude of off-diagonal terms becomes much less than the magnitude of the diagonal terms. The parallel iterative methods studied by Traub [1973] appear to have inherent linear convergence and thus are likely to be uncompetitive with cyclic odd-even reduction for quadratically convergent cases.

As a final point of comparison, we investigate the additional cost of solving a tridiagonal system for a new right-hand side, after having a solution for a first right-hand side. The number of arithmetic operations is reduced by about 2/3 under these circumstances, so it is worthwhile to take advantage of this reduction when circumstances permit. As before, recursive doubling has a slightly lower count for the general case, and cyclic odd-even reduction has a lower count for highly structured cases. The counts are very close, however, so the comparisons are inconclusive in obtaining measures of the relative speeds.

In Section II of this paper we examine the three algorithms in their most general form. The symmetric constant diagonal case, which is the most familiar for cyclic odd-even reduction and Buneman's algorithm, is analyzed in Section III. Section IV contains the analysis for the CDC STAR class of computers. Convergence rates are compared in Section V, and Section VI examines the additional computation required to solve a set of equations with a new right-hand side.

I. II. The tridiagonal equation solvers

In this section we examine the tridiagonal equation solvers in their most general forms, and obtain the arithmetic operation counts. The cyclic odd-even reduction and Buneman algorithms described here are direct generalizations of the algorithms given in Buzbee *et al.* [1970], and have not appeared before in this form. The recursive doubling algorithm is taken from Stone [1973].

We wish to solve a set of N linear equations of the form A x = y where A is the tridiagonal matrix

$$A = \begin{bmatrix} d_1 & f_1 & & & & \\ e_2 & d_2 & f_2 & & & \\ & e_3 & d_3 & f_3 & & \\ & & \ddots & \ddots & \ddots & \\ & & & e_{N-1} & d_{N-1} & f_{N-1} \\ & & & & e_N & d_N \end{bmatrix}$$

For cyclic odd-even reduction and the Buneman algorithm, it is most convenient if $N = 2^m - 1$, while for recursive doubling we should have $N = 2^m$ for greatest efficiency. In this discussion we assume $N = 2^m - 1$ and ignore the slight inefficiencies introduced in the recursive doubling algorithm.

The general scheme of cyclic odd-even reduction and the Buneman algorithm are similar. Consider the i th row of A, for i even. This row has the form $(\ldots.0, e_i, d_i, f_i, 0 \ldots)$. Multiples of row i+1 and row i-1 are added to a multiple of this row to form the new row whose form is $(\ldots.e_i', 0, d_i', 0, f_i', \ldots)$. This operation creates a tridiagonal system from the $2^{m-1}-1$ even rows of A. Although the odd rows have been eliminated, the odd unknowns can be obtained from the even unknowns by back substitution. Given the system of $2^{m-1}-1$ equations involving just the even unknowns, we can eliminate every other row by repeating the process above, leaving a set of $2^{m-2}-1$ equations involving unknowns whose subscripts are multiples of 4. This process is repeated until we obtain a single equation for $x_{2^{m-1}}$, which can be readily solved. Then by back substitution, we can compute the eliminated unknowns in the reverse of the order in which they were eliminated.

To describe the algorithm, let $(\ldots.0, e, d, f, 0 \ldots.)$ be the i th row whose new values will be $(\ldots.e', 0, d', 0, f', \ldots.)$, and let the rows below and above which are added to this row be respectively, $(\ldots.e^+, d^+, f^+ \ldots.)$ and $(\ldots.e^-, d^-, f^- \ldots)$. For the first iteration, these rows have index i+1 and i-1, respectively, and for the k th iteration they have index

$i+2^{k-1}$ and $i-2^{k-1}$.

The inner loop of the reduction process of cyclic odd-even reduction then becomes:

$$\begin{aligned} d' &= d^{+}ef^{-} + d^{-}fe^{+} - d^{-}d^{+}d \\ e' &= d^{+}ee^{-} \\ f' &= d^{-}ff^{+} \\ y' &= d^{+}ey^{-} + d^{-}fy^{+} - d^{-}d^{+}y \end{aligned} \tag{1}$$

These equations are obtained by adding ed^{+} times the equation above, and fd^{-} times the equation below to $-d^{-}d^{+}$ times the middle equation.

Back substitution requires the solution of equations of the form

$$ex^{-} + dx + fx^{+} = y$$

for x when both x^{-} and x^{+} are known. Thus the inner loop of the back substitution operation has the form

$$x = (y - ex^{-} - fx^{+})/d \tag{2}$$

where e, d, and f represent intermediate rather than initial values of the variables.

For the reduction and back substitution the first and last equations are special cases because only one row is combined with these rows. For a parallel computer, they must be processed by the same vector instructions as the interior equations. This is usually done with the aid of masks or other artifices to obtain the correct answers. There is essentially no time lost or gained in processing the boundary equations, so we ignore these special conditions in the remainder of this paper.

To evaluate (1) efficiently, we suggest that d^-d^+, d^+e, and d^-f be computed first, then used where needed to compute the new variables. This gives 11 multiplications and four additions per iteration to compute (1). When we account for the back substitution (2) and note that $\lceil \log_2 N \rceil - 1$ iterations of (1) and (2) are required, we find the total number of operations for cyclic odd-even reduction to be as shown in Table I.

Turning now to the Buneman algorithm, the derivation for it comes from (1) where we write for y'

$$\begin{aligned} y' &= d^+ey^- + d^-fy^+ - d^-d^+y \\ &= (d^+ef^- + d^-fe^+ - dd^-d^+)\, y/d \\ &\quad + d^+ ey^- + d^-fy^+ - (y/d)\,(d^+ef^- + d^-fe^+) \end{aligned} \tag{3}$$

Now we introduce quantities p and q such that

$$y = dp + q \tag{4}$$

and similarly we write

$$\begin{aligned} y^+ &= d^+p^+ + q^+ \\ y^- &= d^-p^- + q^- \end{aligned} \tag{5}$$

Placing (3) into the form of (4) we find

$$y' = d'p' + q' \tag{6}$$

Table I

Operation counts for parallel tridiagonal solvers

(Array computer)

Equation type	Cyclic odd-even reduction	Buneman's algorithm	Recursive doubling
$(\ldots e_i, d_i, f_i, \ldots)$	$K(13M + 6A + D)$	$K(15M + 10A + 2D)$	$K(12M + 5A) + 2M + A + 4D$
$(\ldots e_i, 1, f_i \ldots)$	---	---	$K(11M + 5A) + M + A + 4D$ $[+3D]$
$(\ldots e_i, d_i, 1/e_{i+1} \ldots)$	---	---	$K(9M + 5A) + M + A + 4D$ $[+ KM + 3D]$
$(\ldots e_i, 1, e_i \ldots)$	---	---	$K(11M + 5A) + M + A + 4D$ $[+2D]$
$(\ldots 1, d_i, 1 \ldots)$	---	---	$K(9M + 5A) + M + A + 4D$ $[+2D]$
$(\ldots e, d, e \ldots)$	$K(5M + 6A + D)$	$K(6M + 10A + 2D)$	---
$(\ldots 1, d, 1 \ldots)$	$K(2M + 5A + D)$ $[+D]$	$K(M + 9A + 2D)$ $[+D]$	---

$K = \lceil \log_2 N \rceil - 1$

M = Multiplications

A = Additions

D = Divisions

Bracketed expressions show the number of operations required to normalize into the given form.

where

$$\begin{aligned} d' &= d^+ef^- + d^-fe^+ - dd^-d^+ \\ e' &= d^+ee^- \\ f' &= d^-ff^+ \\ p' &= p + (q - ep^- - fp^+)/d \\ q' &= d^+eq^- + d^-fq^+ - (d^+ef^- + d^-fe^+)p' \end{aligned} \tag{7}$$

The inner loop of the reduction process for the Buneman algorithm consists of repeating (7) where the primed variables have subscripts of the form $i \cdot 2^k$ in the kth iteration, until, at the last iteration, we obtain a single equation for $x_{2^{m-1}}$. Back substitution proceeds as with cyclic odd-even reduction by solving the equation

$$ex^- + dx + fx^+ = y = dp + q$$

or

$$x = p + (q - ex^- - fx^+)/d. \tag{8}$$

Here the variable x has an odd subscript, and x^- and x^+ have even subscripts in the reduced set of equations and are known from the previous iteration of the back substitution process. As before, variables are recovered in the reverse order in which they are eliminated.

To evaluate (7) and (8), the best method appears to be to compute d^+e, d^-f, and then $(d^+ef^- + d^-fe^+)$. These quantities appear at least twice in (7). From this we obtain 13 multiplications per iteration for (7). The total number of operations for (7) and (8) combined appears in Table I.

The last of three tridiagonal solvers, recursive doubling, is described in detail in Stone[73]. The algorithm has three parts, namely, the computation of the LU decomposition of A, a forward sweep through a lower bidiagonal system, and a backward sweep through an upper bidiagonal system. In the LU decomposition phase, during each iteration we update the values of variables r_i, s_i, and t_i, $1 \le i \le N$, from their previous values. Let $v_i = e_i f_{i-1}$ for $2 \le i \le N$, and note that the v_i vector can be computed by a single multiplication before we perform the iteration below. During the k^{th} iteration, we compute the updated values r_i', s_i', and t_i' of r_i, s_i and t_i by the formulas:

$$\begin{aligned} r_i' &= s_i s_{i-2^k+1} - v_{i-2^k+2}\, r_i r_{i-2^k} \\ s_i' &= t_i s_{i-2^k} - v_{i-2^k+1} s_i r_{i-2^k-1} \\ t_i' &= d_i s_{i-1}' - v_i r_{i-2}'. \end{aligned} \qquad (9)$$

Here the subscript expressions with terms of the form 2^k are written out explicitly, whereas they are implicit in the superscript notation of (1) to (8).

The forward and backward sweeps have identical form, as indicated for the i^{th} equations below:

$$\begin{aligned} y' &= y + y^- m \\ m' &= m m^- \end{aligned} \qquad (10)$$

Here the minus sign superscript denotes a variable with subscript $i-2^k$ in the k^{th} iteration for the forward sweep. Equation (10) describes the forward sweep. The backward sweep is obtained by replacing y^- and m^- by y^+ and m^+. In addition to the operations that occur in loops described by (9) and (10), there are two multiplications, one addition, and four divisions required to initialize loops. The operation count for recursive doubling is summarized in Table I. Note that we assume the array computer has N processors and the system is of order N in obtaining these counts.

Recursive doubling has an advantage not shared by the other two algorithms in that it lends itself to solving normalized equations. In (9), either the v_i's or the d_i's can be normalized to unity with a consequent saving in multiplications. The d_i's can be normalized by dividing the i^{th} equation by d_i, and d_i's remain normalized throughout the algorithm. To normalize equations for which each d_i is nonzero on a parallel computer with N processors we simply perform three divisions by d_i to compute the normalized values of e_i, f_i, and y_i, respectively. The operation count for this solution is shown in Table I for the problem labeled $(\ldots e_i, 1, f_i \ldots)$.

Note that when division is much longer than multiplication, there may be no real gain from this normalization unless N is very large. However, a different normalization is possible that is more likely to result in a gain in speed. For the second normalization we set $v_i = 1$ for $2 \leq i \leq N-1$. We let primes indicate the normalized values of e_i and f_i, and we set out to make $v_i = e_i'f_{i-1}' = 1$ by setting $f_{i-1}' = 1/e_i'$. By dividing equation N-1 by $e_N f_{N-1}$ we obtain $f_{N-1}' = 1/e_N$, so $v_N = 1$. Similarly, dividing equation N-2 by $e_{N-1}' f_{N-2} = (e_{N-1}f_{N-2})/(e_N f_{N-1})$ yields $v_{N-1} = 1$. Fortunately we can compute all of the normalization divisors in parallel using recursive doubling. If u_i is the divisor of the i^{th} equation, then we have

$$u_N = e_N f_{N-1}$$

and

$$u_i = (e_i f_{i-1})/u_{i+1} \qquad \text{for } 1 \le i \le N-1$$

Note that we can compute u_i from u_{i+2} by substituting for u_{i+1} above, and we find:

$$u_i = u_{i+2} \cdot (e_i f_{i-1})/(e_{i+1} f_i) \qquad \text{for } 1 \le i \le N-2$$

Recursive doubling can be applied directly to this form of the recurrence, to yield an efficient parallel method [Stone, 1973]. The N-2 constants of the form $(e_i f_{i-1})/(e_{i+1} f_i)$ can be computed with a single parallel multiplication and a single parallel division, and then from these constants all of the divisors u_i can be computed in $\lceil \log_2 N \rceil - 2$ parallel multiplications. The normalization itself requires two divisions to compute normalized values of d_i and y_i. The operation count for this method is shown in Table I in the column labeled $(\ldots e_i, d_i, 1/e_{i+1} \ldots)$. For both normalization methods the overhead for normalization is shown in brackets, and the unbracketed terms indicate the operation count for the normalized solutions.

While on the subject of normalization, we should also treat the symmetric case for which $e_i = f_i$ for $2 \le i \le N-1$. Again, the cyclic odd-even reduction and Buneman algorithms have no specific advantage for this case because both the symmetry and the unit coefficients are destroyed after one iteration. For recursive doubling, the symmetric case can be normalized either into the form $(\ldots e_i, 1, e_i \ldots)$ or into the form $(\ldots 1, d_i, 1 \ldots)$ depending on how we select the normalizing constants. The former case is solved in the same manner as the unsymmetric case $(\ldots e_i, 1, f_i \ldots)$ except that two, rather than three, divisions are required to create the normalized form since $e_i = f_i$. In the latter case, after normalization the solution is identical to the solution for the case $(\ldots e_i, d_i, 1/e_{i+1} \ldots)$, but the normalization cost is reduced from three divisions and $\lceil \log_2 N \rceil - 1$ multiplications to just two divisions, since the normalization constant for the i^{th} equation is just e_i. The two symmetric cases with normalization are also listed in Table I.

To summarize the results of Table I, for the general form $(\ldots e_i, d_i, f_i \ldots)$ the algorithm with the least operation count per iteration is recursive doubling, with 17 operations per iteration, and cyclic odd-even reduction trails slightly behind at 20 operations per iteration. Buneman's algorithm has the highest count with 25 operations per iteration, and is particularly at a disadvantage on a machine with relatively slow division since it has two divisions per iteration as compared to one for odd-even reduction and zero for recursive doubling. Using normalization in the recursive doubling algorithm may result in a slightly faster algorithm for large N, but the gain may be insignificant for N=63 and N=127. For the general case, recursive doubling appears to be slightly preferrable to cyclic odd-even reduction and quite preferable to Buneman's algorithm. The comparison with odd-even reduction is so close as to be inconclusive, since careful consideration of overhead computations such as memory fetching, indexing, and routing can change the relative speed estimates. For symmetric matrices with nonconstant diagonals, the analysis indicates a much stronger preferance for recursive doubling, with 14 operations per iteration as compared to 20, which suggests a speed differential of 30% or more might exist in this case. Again a more careful analysis of the overhead operations is required because the true speed differential may be much smaller than indicated here.

III. The symmetric constant-diagonal case

In this section we examine the tridiagonal equation solvers when operating on matrices of the form

$$A = \begin{bmatrix} d & e & & & & & \\ e & d & e & & & & \\ & e & d & e & & & \\ & & e & d & e & & \\ & & & \ddots & \ddots & \ddots & \\ & & & & e & d & e \\ & & & & & e & d \end{bmatrix}$$

which in shorthand notation we denote as the (...e, d, e ...) case. For this case the cyclic odd-even reduction and Buneman algorithms are greatly simplified. Consider the three adjacent rows of A

$$\begin{matrix} e & d & e & & \\ & e & d & e & \\ & & e & d & e \end{matrix}$$

and note that the middle row can be changed from the form (...0, e, d, e, 0...) into the form (...e', 0, d', 0, e'...) by subtracting d times it from e times the sum of the first and third rows. This gives the following iteration:

$$\begin{aligned} d' &= 2e^2 - d^2 \\ e' &= e^2 \\ y' &= e(y^+ + y^-) - dy \end{aligned} \qquad (11)$$

As before, the odd indexed equations are eliminated in the first iteration, and at each subsequent iteration the odd equations of the reduced tridiagonal system are eliminated. This is exactly the iteration described by Buzbee et al. [1970] for the solution to two-dimensional Poisson problems. The back substitution involves substituting for x^+ and x^- in the equation

$$ex^- + dx + ex^+ = y$$

or

$$x = (y - e(x + x^+))/d \qquad (12)$$

Counting multiplication by a constant 2 as an addition, we find the cost in arithmetic operations per iteration of (11) and (12) is five multiplications, six additions and one division.

Buneman's algorithm uses intermediate variables p and q such that

$$y = dp + q.$$

With appropriate modifications to (11) we find the reduction iteration to be

$$\begin{aligned} d' &= 2e^2 - d^2 \\ e' &= e^2 \\ p' &= p + [q - e(p^+ + p^-)]/d \qquad (13) \\ q' &= e(q^+ + q^-) - 2e^2p' \end{aligned}$$

Back substitution involves solution of the equation

$$e(x^+ + x^-) + dx = y = dp + q$$

which yields

$$x = p + [q - e(x^+ + x^-)]/d \qquad (14)$$

Again counting multiplication by 2 as an addition, we find the per iteration cost of (13) and (14) to be six multiplications, 10 additions, and two divisions.

The cost for these two algorithms is summarized in Table I in the column labeled (...e,d,e...). To compare recursive doubling to these algorithms, use the analysis for the (...1,d_i,1...) case for recursive doubling since the presence of constant diagonals does not change the recursive doubling algorithm. The best algorithm for the symmetric constant diagonal case appears to be cyclic odd-even reduction. The Buneman algorithm has a higher iteration count than recursive doubling for the (...1,d_i,1...) case, so it again appears to be uncompetitive with the two other algorithms.

Both the Buneman and the cyclic odd-even reduction algorithms enjoy a significant speed increase when the e coefficients are initially equal to unity. For cyclic odd-even reduction, they remain unity throughout the computation, and thereby reduce the number of multiplications and additions per iteration to two and five, respectively. The number of operations for this case is summarized in Table I in the column labeled (... 1,d,1...). Likewise, the number of operations for the Buneman algorithm can be reduced as indicated in Table I. Note that the (...e,d,e...) case can be normalized into the (...1,d,1...) form by two divisions to normalize d and y. On most parallel computers the normalized values of d and y can be computed simultaneously, since d is a constant. It can be treated as an additional of y for the normalization, and then can be broadcast in a separate operation to the vector storage area for d. Consequently, the cost of normalization is given as one division rather than two.

For both algorithms the multiplications in each iteration have almost been eliminated and the operation counts are significantly reduced. Thus the symmetric constant diagonal case almost certainly should be normalized into the (...1,d,1...) form to obtain greater speed.

Note that for the present case these two algorithms should be compared to the $(\ldots 1, d_i, 1 \ldots)$ case for recursive doubling as indicated before, and the comparison shows that cyclic odd-even reduction is the most preferable with eight operations per iteration as compared to 12 for Buneman's algorithm and 14 for recursive doubling. The fact that division is usually more time consuming than addition or multiplication makes the cyclic odd-even reduction algorithm a little less attractive than indicated here, but its operation count is so much better than the other two algorithms that it is almost certainly the fastest of the three for this case. For the ILLIAC IV computer, for which $A \approx M \approx D/5$, recursive doubling is likely to be as fast or faster than the Buneman algorithm.

The conclusions to be drawn from Table I suggest that both cyclic odd-even reduction and recursive doubling are attractive for implementation, with recursive doubling slightly favored for the general tridiagonal system, and a much stronger preferance for cyclic odd-even reduction for symmetric, constant diagonal systems. The operation counts are sufficiently close in most cases to make our estimates of relative speed subject to error due to failure to consider the cost of overhead operations. It is worthwhile to mention that all of the cases shown in Table I can be solved efficiently by implementing just two different subroutines. A single recursive doubling subroutine can treat all of the cases in the first five rows without change. The subroutine should test to see if the matrix is symmetric and if so, it should normalize the matrix into the $(\ldots 1, d_i, 1 \ldots)$ form. Otherwise the matrix should be normalized into the $(\ldots e_i, d_i, 1/e_i \ldots)$ form. A single recursive doubling subroutine solves both of these forms, and thus is suitable for each of the first five cases of Table I when normalization is used. The last two cases can be solved with a single cyclic odd-even reduction subroutine. Thus it is easy to take advantage of the special form of certain tridiagonal matrices to increase the speed of computation.

While the differences in computation speed between the recursive doubling and odd-even reduction algorithms are not substantial in most cases, the Buneman algorithm does appear to be uniformly slower than either of the other two algorithms. Its value, of course, lies in its application to two-dimensional problems and for the solution of the corresponding block-tridiagonal systems, primarily because of its good numerical stability.

IV. The solution of tridiagonal systems on pipeline computers

Under the assumptions of the previous analysis, the computer is a vector processor that can perform up to N identical operations simultaneously, and we have carefully restricted the problems to be of order N. This is a reasonable model for the ILLIAC IV computer where $N = 64$ and $N = 128$ in double and single precision modes, respectively. On pipeline machines our analysis is not accurate because computation time on such machines depends not only on the number of vector instructions executed, but on the total number of elementary arithmetic operations as well. The Buneman and cyclic odd-even reduction algorithms both require $O(N)$ elementary arithmetic operations, while the recursive doubling algorithm requires $O(N \log_2 N)$ elementary arithmetic operations, so that for sufficiently large N it is guaranteed to be slower than the other algorithms. In this section we present a variation of the recursive doubling algorithm with $O(N)$ arithmetic operations, and compare the three algorithms when executing on a pipeline computer.

Let V indicate the number of vector arithmetic instructions issued during the execution of a program, and let T denote the total number of elementary arithmetic operations performed by these instructions. That is, if the i^{th} vector arithmetic instruction operates on n_i pairs of operands to produce n_i results, then $T = \Sigma n_i$. The computation time of a program on a pipeline processor such as CDC STAR is then approximated by the expression $c_1V + c_2T$ where c_1 and c_2 are constants. For the CDC STAR c_1 is the larger constant, about 100 to 200 times larger than C_2 depending on factors related to the storage of the data and type of operation. For the problem at hand V is $O(\log_2 N)$ and T is $O(N)$ so that the first term in the expression dominates for small N and the second for large N, with a cross-over point somewhere in the region $64 \le N \le 1024$, depending on factors not treated here. Our previous analysis has measured V, and in this section we measure T, since it clearly contributes to the computation time, and in many cases is the dominant term.

We begin by deriving a variation of the recursive doubling algorithm in Stone [1973] that requires only $O(N)$ arithmetic operations. Then we compare the three tridiagonal solvers. The major issue concerns the number of operations required to solve recurrence relations with associative operators. As a typical example, consider the solution of the recurrence

$$\begin{aligned} x_i &= a_i + x_{i-1} \qquad & 2 \le i \le N \\ &= a_1 & i = 1 \end{aligned} \tag{15}$$

when the coefficients a_i are given. This can obviously be solved sequentially to obtain x_i for $2 \le i \le N$ with $N - 1$ additions. Recursive doubling yields an algorithm to compute all of the x_i's with $\lceil \log_2 N \rceil$ vector operations, but the number of additions increases to $O(N \log_2 N)$. The increase comes about because for each vector operation, at least $N/2$ additions occur.

The goal is to compute all of the x_i's in $O(\log_2 N)$ vector operations while holding the number of additions to $O(N)$. This turns out to be very easy to do given the capabilities of a pipeline computer like the CDC STAR. The algorithm that we describe here is suitable for the CDC STAR, but because of the differences between STAR and ILLIAC, it has no particular advantage for the ILLIAC.

For convenience in this discussion we shall assume that $N = 2^m$. The algorithm to solve (15) involves the computation of a sequence of vectors $X^{(k)}$, $1 \le k \le \log_2 N$, where $X^{(k)}$ has length $N/2^k$. When we obtain $X^{(m)}$, we then sweep backwards through the vectors, updating them so that after the last step of the computation the updated vector $X^{(1)}$ contains the values of x_i.

The algorithm is given below in a ALGOL-like notation. Here the vector a[i] contains the coefficients a_i. The parenthesized expressions appearing after assignment statements give the values of the index variable i for which the vector operation is performed. The algorithm makes use of vectors $X^{(k)}$, $1 \le k \le m$, with $X^{(k)}$ of length $N/2^{k-1}$.

```
$X^{(1)}[i] := a[i], (1 \le i \le N)$;
comment forward sweep. $\log_2 N = m$;
for k := 1 step 1 until m-1 do
  begin
    $X^{(k+1)}[i] := X^{(k)}[2i] + X^{(k)}[2i-1], (1 \le i \le N/2^k)$;
  end;
comment backward sweep;
for k := m-1 step -1 until 1 do
  begin
    $X^{(k)}[i] := X^{(k+1)}[i/2]$, (i even, $2 \le i \le N/2^{k-1}$);
    $X^{(k)}[i] := X^{(k+1)}[(i-1)/2] + X^{(k)}[i]$, (i odd, $3 \le i \le N/2^{k-1}$);
  end;
```

An example of the algorithm for N = 8 appears in Fig. 1.

i =	1	2	3	4	5	6	7	8
$x^{(1)}[i]$	1	2	3	4	5	6	7	8
$x^{(2)}[i]$	1-2	3-4	5-6	7-8				
$x^{(3)}[i]$	1-4	5-6						
$x^{(4)}[i]$	1-8							
$x^{(3)}[i]$	1-4	1-8						
$x^{(2)}[i]$	1-2	1-4	1-6	1-8				
$x^{(1)}[i]$	1	1-2	1-3	1-4	1-5	1-6	1-7	1-8

Each entry indicates the range of subscripts of the a_i's in a partial summation.

Fig. 1. A sample execution of the modified recursive doubling algorithm.

The figure shows that the algorithm proceeds in the forward sweep in the familiar way, that is, it computes sums of adjacent pairs, then of adjacent quadruples, and then obtains the sum of all elements. The backward sweep simply uses the stored information to compute the final values of x_i, first for $i = 4$, then for $i = 2$ and 6, and finally for i odd. The strategy in this algorithm is essentially the same strategy followed by cyclic odd-even reduction and the Buneman algorithm. We give no formal proof of correctness here because the algorithm is quite straightforward.

We should mention that the number of vector arithmetic operations for this algorithm is $2\lceil \log_2 N \rceil$ rather than $\lceil \log_2 N \rceil$ which holds for the more usual recursive doubling algorithm. The overhead for this algorithm is also quite high, and does not appear in this analysis. Consequently, for small N, the new algorithm is not recommended.

To modify the recursive doubling tridiagonal solver into the new form, we simply note that the addition in (15) can be replaced by any associative operator. The solution of tridiagonal equations involves the solution recurrences of the form of (15) in which the variables x_i are vectors of length 2, the variables a_i are 2×2 matrices, and the operation is matrix multiplication. The recurrences appear once in the calculation of the LU decomposition, once in the forward sweep of a bidiagonal system, and once in the backward sweep of a bidiagonal system. Thus these three recurrence systems can each be evaluated by the algorithm given here, provided we replace each $X^{(k)}[i]$ by a 2×2 matrix, and we change the addition in the algorithm to matrix multiplication.

The cost in arithmetic operations per iteration for the modified recursive doubling algorithm is 50% larger than the cost for the unmodified algorithm. The matrix multiplication for the LU decomposition requires eight multiplications and four additions for the forward sweep, but only half of this for the backward sweep. The matrices for the two bidiagonal systems have the special form:

$$\begin{bmatrix} a & b \\ 0 & 1 \end{bmatrix}$$

Multiplication of matrices in this form requires only two multiplications and one addition. This is done twice for each of the bidiagonal systems to yield a total of eight multiplications and four additions. Special forms of the tridiagonal matrix do not appear to offer a reduction in arithmetic operations in the LU decomposition computation for this algorithm.

To compare the three algorithms, we can use the data from Table I, and we discover that the relative costs of the algorithms are not significantly different for the pipeline computer except that recursive doubling is less attractive, particularly when division has a high cost. The total number of arithmetic operations for all three algorithms is $O(N)$ times the number of arithmetic operations per iteration, even though the number of iterations is $O(\log_2 N)$. This follows because the lengths of vectors treated are of the form $2^i - 1$, $1 \le i \le m$, which when summed yields a total of $2^{m+1} - 1 - m = O(N)$ when $m = \lceil \log_2 N \rceil$.

Table II summarizes the count of the total number of arithmetic operations for each of the algorithms. In each case terms of order less $O(N)$ are omitted.

Table II

Operation counts for tridiagonal solvers

(Pipeline computer)

Equation type	Cyclic odd-even reduction	Buneman's algorithm	Recursive doubling
$(\ldots e_i, d_i, f_i \ldots)$	N(13M + 6A + D)	N(15M + 10A + 2D)	N(22M + 11A + 4D)
$(\ldots e, d, e \ldots)$	N(5M + 6A + D)	N(6M + 10A + 2D)	---
$(\ldots 1, d, 1 \ldots)$	N(2M + 5A + D) [+ND]	N(M + 9A + 2D) [+ND]	---

Only terms of order N are shown.

The terms of lower order may be significant in many cases, and tend to make cyclic odd-even reduction even more attractive than it appears to be in Table II. Since recursive doubling requires four vector divisions as part of its overhead, these operations contribute substantially to its operation count. The number of vector instructions executed in each case is not shown in the figure, but for every case this number is $O(\log_2 N)$, and can be ignored for large N. For small values of N, the relative costs of the algorithms tends toward the data given in Table I since this accounts for the number of vector instructions.

Table II shows that cyclic odd-even reduction has a very large advantage over the other algorithms in operation count, and is therefore likely to be the fastest of the three algorithms. Again we must account for computational overhead which may alter relative desirability somewhat but is unlikely to change the general conclusions.

V. Semi-direct methods

The three algorithms under investigation are direct methods for the solution of tridiagonal equations, but under special conditions the Buneman and cyclic odd-even reduction algorithms behave like iterative methods. The values of intermediate quantities converge to final values, and may reach the final values to within machine accuracy well before the full number of iterations have been done. The convergence can be tested, and the algorithms can be terminated early when full machine accuracy is attained.

Traub [1973] discusses several iterative algorithms for the parallel solution of tridiagonal equations. One is a direct method in the sense that in the absence of round-off error it terminates with the exact solution in a fixed number of iterations. However, it is clearly iterative in intent, since it is designed to obtain a convergent sequence of intermediate solutions, with early termination of the algorithm when convergence is reached. And indeed, when used as an iterative algorithm, there is a substantial improvement in computation speed. Consequently, the algorithm analyzed by Traub, like cyclic odd-even reduction and Buneman's algorithm, is *semi-direct* (or perhaps, should be called *semi-iterative*).

In this section we review the convergence rates of the iterative and semi-direct algorithms proposed by Traub and compare them to the convergence of the Buneman and cyclic odd-even reduction algorithms. We find the convergence rates of the Traub algorithms are linear, whereas the convergence rates of the other algorithms are linear when diagonal dominance is small, but change rapidly to quadratic as dominance increases. Thus, cyclic odd-even reduction and the Buneman algorithm appear to be quite efficient as iterative algorithms when conditions permit.

Traub analyzes four parallel algorithms based on well-known serial algorithms for the solution of tridiagonal systems. They are respectively called parallel Gauss, Jacobi, parallel Gauss-Seidel, and parallel optimal SOR. He also reports a parallel iterative algorithm based on the LU decomposition method, whose convergence rate is that of the parallel Gauss algorithm. We do not treat the LU decomposition algorithm separately here.

For an algorithm to be convergent, some dominance conditions must hold for the tridiagonal system of equations. Most often it is convenient to assume the system is diagonally dominant. For our purposes we assume a particularly strong form of dominance, and we note our assumptions are sufficient for convergence, but not necessary. The assumptions greatly simplify the comparison of several different algorithms. Specifically we assume:

$$|e_i|,\ |f_i| < |d_i/2| \qquad 1 \le i \le N$$

When these assumptions hold, we can normalize into the form $(\ldots,e_i,1,f_i\ldots)$ and bound the convergence by calculating the convergence of the more slowly convergent constant diagonal system $(\ldots 1,1/e,1\ldots)$ where $e = \max_i(e_i,f_i)$ of the normalized system.

These assumptions are at least as strong as the assumptions of Traub, so that all of the algorithms he describes are convergent under these assumptions.

To briefly summarize his results, an upper bound on the number of iterations required to reduce the error in an initial approximation by an amount 2^{-b} grows linearly in b for the parallel Gauss and Jacobi algorithms, and grows as the square of b for the parallel Gauss-Seidel and parallel optimal SOR algorithms. Thus, if the bounds on convergence rates are accurate estimates, the best convergence rates from this class of algorithms might be linear in order. Traub reports numerical experiments that tend to confirm that the parallel Gauss and Jacobi algorithms have convergence rates predicted by the bounds.

Now we show that the Buneman and cyclic odd-even reduction algorithms converge linearly or quadratically under the stated assumptions. Since the two algorithms compute equivalent, but not identical, quantities during a computation, it is sufficient to show that either of the algorithms converges, for if one does, then both do. In using the algorithms as semi-direct algorithms, the strategy is to compute a sequence of tridiagonal systems, one in each iteration, and to check the diagonal dominance of each new system as it is computed. Under convergent conditions the ratios $|e_i/d_i|$ and $|f_i/d_i|$ eventually become less than 2^{-b} where b is the machine accuracy. At this time the tridiagonal system is declared to be a diagonal system that can be solved directly, and whose solutions can be used immediately in the backward sweep of the back substitution. This idea has been used successfully to solve two-dimensional Poisson problems. [Hockney, 1965].

The convergence of the cyclic odd-even reduction algorithm is easily discovered by examining (11). To find a lower bound on convergence of the system $(\ldots,e_i,d_i,f_i,\ldots)$ we mentioned above that we can use the more slowly convergent system $(\ldots,1,1/e,1,\ldots)$ where e is the maximum normalized off-diagonal element. Thus it is sufficient to consider the symmetric constant diagonal case $(\ldots,1,d,1,\ldots)$. When $|d| > 2$, convergence occurs when the diagonal elements overflow, that is when $|d| \geq 2^b$, since the off-diagonal elements remain at unity throughout the computation, and thus $|e/d| \leq 2^{-b}$ at overflow. Convergence in this iteration is essentially linear or quadratic depending on whether $|d|$ is respectively less than or greater than 3. The transition between linear and quadratic behavior occurs when $|d| \approx 3$.

To determine the convergence rate, we define $\varepsilon^{(k)} = |d^{(k)}| - 2 \geq 0$ where $d^{(k)}$ is the diagonal element computed during the kth iteration. We attempt to measure how fast $|d^{(k)}|$ grows as a function of k, as it grows toward 2^b. From (11) we have:

$$d^{(k+1)} = 2 - [d^{(k)}]^2$$

so that

$$\begin{aligned} |d^{(k+1)}| &= |2 - [2 + \varepsilon^{(k)}]^2| \\ &\geq 2 + 4\varepsilon^{(k)} + [\varepsilon^{(k)}]^2 \end{aligned}$$

Thus

$$\varepsilon^{(k+1)} \geq 4\varepsilon^{(k)} + [\varepsilon^{(k)}]^2 \qquad (16)$$

For $\varepsilon^{(k)} < 1$ we have

$$\varepsilon^{(k+1)} \geq 4\varepsilon^{(k)} \qquad (17)$$

and the rate of growth is at least first order or better. For $\varepsilon^{(k)} > 1$, we have

$$\varepsilon^{(k+1)} \geq [\varepsilon^{(k)}]^2 \qquad (18)$$

at which point the rate of growth is quadratic or better. Since $\varepsilon = |d| - 2$, the breakpoint comes roughly at $|d| = 3$ as indicated above.

The number of iterations required to increase $|d^{(0)}|$ from $2 + \varepsilon^{(0)}$ to 3 is the number of iterations, k, such that

$$\varepsilon^{(k)} \geq 1$$

but from (16) it follows that

$$\varepsilon^{(k)} \geq 4^k \varepsilon^{(0)} \qquad (19)$$

Solving the inequality $4^k \varepsilon^{(0)} \geq 1$, for k yields

$$k = \left\lceil - \frac{\log_2 \varepsilon^{(0)}}{2} \right\rceil \qquad (20)$$

is sufficient to insure that $|d^{(k)}| \geq 3$. Note that if $\varepsilon^{(0)}$ is of size 2^{-t}, then the number of iterations is roughly t/2. The number of iterations to raise $|d^{(0)}|$ from 3 to full machine size of 2^b is obtained from (18) to be the least k such that

$$\varepsilon^{(k)} \geq [\varepsilon^{(0)}]^{2^k} \geq 2^b$$

or

$$k \geq \log_2 b - \log_2 \log_2 \varepsilon^{(0)} \qquad (21)$$

Thus when dominance is sufficiently great in a tridiagonal system, (21) guarantees extremely fast convergence of the direct methods. The convergence rate for less dominance as indicated by (20) is of the same order as the convergence found by Traub, so that at least in this region the direct methods still converge as rapidly but not dramatically faster than the algorithms described by Traub.

The very good convergence of cyclic odd-even reduction and Buneman algorithm has a simple intuitive explanation. The solution of a tri-diagonal system requires that every equation influence every other equation. For diagonally dominant systems the influence diminishes with the distance between equations, and in fact, for strong dominance, the influence of one equation is neglible many equations away. In the two direct algorithms, during the k^{th} iteration, each equation spreads its influence over equations up to 2^k rows away in each direction. Thus, when the system is strongly diagonally dominant, the algorithms can terminate early when each equation has spread its influence to all equations within the range of its influence.

The algorithms studied by Traub are structured so that during each successive iteration the sphere of influence of each equation increases by at most one equation in each direction, rather than doubles in size as is the case for the algorithms above. Thus the number of iterations required to spread the influence of each equation sufficiently far is greater than the number of iterations required for the direct algorithms.

We mention in closing that the algorithms studied by Traub are particularly amenable to contexts in which a good initial guess is available. This occurs frequently when a sequence of slightly perturbed equations are solved. The two direct methods do not make use of initial guesses, and are likely to be slower when good initial guesses are available.

VI. Repeated solutions with new right-hand sides

In some contexts, particularly in the solution of Poisson's equation on a rectangle, one system of tridiagonal equations must be solved repeatedly with different right-hand sides. In these contexts it is possible to reduce computation time substantially by taking advantage of intermediate results produced during the first solution of a set of equations. The classic example is the repeated use of the LU decomposition of a system. Since the recursive doubling algorithm computes the LU decomposition, it offers some advantage when equations are solved repeatedly. The cyclic odd-even reduction algorithm also offers some benefit. In this section we reexamine these algorithms and compare their relative costs for repeated solutions.

From the description of the cyclic odd-even reduction algorithm in Section II, it is clear that we need not repeat the computations for d', e', and f' in (1) when we solve a system with a new right-hand side. We must save the intermediate variables, and then apply the equation for y' in (1) repeatedly, $\log_2 N$ times. At this point we can do the normal back substitution process of (2), and find the new solution. If we assume that we have saved the values of $d^- d^+$, $d^+ e$, and $d^- f$, then at most three multiplications and two additions per iteration are required in the forward process, and the backward substitution remains unchanged with two multiplications, one division, and two subtractions. These results are summarized in Table III. The analysis of the various special cases is similar to our previous analysis and is not repeated here. Golub [1974] has made this

Table III

Operation counts for each additional solution after first solution.

	Cyclic odd-even reduction	Buneman's algorithm	Recursive doubling
$(\ldots e_i, d_i, f_i \ldots)$	K(5M + 4A + D)	K(7M + 8A + 2D)	K(4M + 2A) + D
$(\ldots e, d, e \ldots)$	K(3M + 4A + D)	K(4M + 8A + 2D)	---
$(\ldots 1, d, 1 \ldots)$	K(M + 4A + D) [+D]	K(9A + 2D) [+D]	---

algorithm explicit by exhibiting a matrix factorization for the cyclic odd-even reduction algorithm that is roughly analgous to an LU decomposition.

Buneman's algorithm also lends itself to repeated solutions of one set of equations, but the savings is relatively less than obtained for cyclic odd-even reduction. We summarize its costs as obtained from (7) and (8) in Table III. There is negligible savings except for the interesting special cases.

Recursive doubling is reduced in complexity by roughly 2/3 when the LU decomposition of (9) is saved and not recomputed. The summary of the analysis for recursive doubling appears in Table III.

Table III follows the general trend set by the previous tables. For the general case, recursive doubling has the least operation count. For the special case of constant symmetric diagonals, cyclic odd-even reduction is relatively efficient and is approximately equal in operation count to recursive doubling. To establish the fastest algorithm for this case, it is essential to account for overhead computations and other factors besides the arithmetic operations enumerated here.

We should mention that the machine model assumed for Table III is an array processor. For a pipeline processor the number of arithmetic operations for recursive doubling is approximately twice that shown in parenthesis in Table III, and thus cyclic odd-even reduction is likely to be uniformly better than the other algorithms when executed on a pipeline computer.

VII. Summary and conclusions

In comparing the three algorithms, the operation count for recursive doubling tends to make it the most attractive when the system of equations has no particular special structure. Cyclic odd-even reduction has the least operation count of the three when the system is symmetric with constant diagonals. Cyclic odd-even reduction appears to be the most efficient algorithm for a pipeline computer in virtually all cases. Moreover, when the system of equations permits cyclic odd-even reduction to be used as an iterative algorithm, it converges as fast or faster than other parallel iterative methods proposed recently.

The analysis focuses on arithmetic operations and does not enumerate the number of memory references, shifts, index calculations, and other overhead. In many instances, the algorithms are sufficiently close in operation count to make a timing evaluation inconclusive, because the overhead computations have not been taken into account. We have recently been informed of a report that does in fact make the careful analysis of overhead to provide relative timings for the CDC STAR computer [Lambiotte and Voigt, 1974]. That report generally substantiates our conclusion that cyclic odd-even reduction is the most desirable algorithm of the ones studied here.

We have also not considered the relative stability of the algorithms. If a system is not diagonally dominant, then the computation of the LU decomposition in the recursive doubling algorithm might fail. Likewise the stability of cyclic odd-even reduction and Buneman's algorithm is in question under these conditions, with Buneman's algorithm possibly having greater stability than cyclic odd-even reduction [Buzbee *et al.*, 1970]. We have omitted all questions of relative stability in this analysis and leave such questions for future research.

Acknowledgement

The author is deeply indebted to Prof. Gene Golub of Stanford University for many discussions, comments, and suggestions regarding this paper. We also acknowledge helpful suggestions and conversations with Prof. Joseph Traub of Carnegie-Mellon University, Dr. Robert G. Voigt of ICASE and Dr. Jules J. Lambiotte of NASA Langley Research Center.

References

Buzbee, B. L., G. H. Golub and C. W. Nielson [1970]. "On direct method for solving Poisson's equations," SIAM J. Numer. Anal., Vol. 7, No. 4, pp. 627-656, Dec. 1970.

Golub, G. H. [1973]. Private communications.

Hockney, R. W. [1965]. "A fast direct solution of Poisson's equation using Fourier analysis," J. ACM, Vol. 12, No. 1, pp. 95-113, Jan. 1965.

Lambiotte, J. J., and R. G. Voigt [1974]. "The solution of tridiagonal linear systems on the CDC STAR-100 computer," ICASE Report, June 1974.

Stone, H. S. [1973]. "An efficient parallel algorithm for the solution of a tridiagonal linear system of equations," J. ACM, Vol. 20, No. 1, pp. 27-38, January 1973.

Traub, J. F. [1973]. "Iterative solution of tridiagonal systems on parallel or vector computers," in Complexity of Sequential and Parallel Numerical Algorithms, J. F. Traub, ed., Academic Press, New York, 1973.