27
8-10-81
Jek
24D NTIS
B6297

(1)

UCID- 19085

MASTER

THE SOLUTION OF SINGLE LINEAR TRIDIAGONAL
SYSTEMS AND VECTORIZATION OF THE ICCG
ALGORITHM ON THE CRAY 1

David S. Kershaw

June 25, 1981

Lawrence
Livermore
Laboratory

This is an informal report intended primarily for internal or limited external distribution. The opinions and conclusions stated are those of the author and may or may not be those of the Laboratory.

Work performed under the auspices of the U.S. Department of Energy by the Lawrence Livermore Laboratory under Contract W-7405-Eng-48.

DISTRIBUTION OF THIS DOCUMENT IS UNLIMITED

DISCLAIMER

This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor any agency Thereof, nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof.

DISCLAIMER

Portions of this document may be illegible in electronic image products. Images are produced from the best available original document.

THE SOLUTION OF SINGLE LINEAR TRIDIAGONAL SYSTEMS AND
VECTORIZATION OF THE ICCG ALGORITHM ON THE CRAY 1

David S. Kershaw

Lawrence Livermore National Laboratory

DISCLAIMER

This book was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor any agency thereof, nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof.

DISTRIBUTION OF THIS DOCUMENT IS UNLIMITED

Ray

ABSTRACT

When we examine the numerical algorithms used to solve the physics equations in codes which model laser fusion we find that a large number of subroutines require the solution of tridiagonal linear systems of equations. One dimensional radiation transport, thermal and suprathermal electron transport, ion thermal conduction, charged particle and neutron transport, all require the solution of tridiagonal systems of equations. The standard algorithm that has been used in the past on CDC 7600's will not vectorize and so cannot take advantage of the large speed increases possible on the Cray-1 through vectorization. There is, however, an alternate algorithm for solving tridiagonal systems, called cyclic reduction, which allows for vectorization, and which is optimal for the Cray-1. Software based on this algorithm is now being used in LASNEX to solve tridiagonal linear systems in the subroutines mentioned above. The new algorithm runs as much as five times faster than the standard algorithm on the Cray-1.

In LASNEX, the two-dimensional radiation transport, electron thermal conduction, ion thermal conduction and neutron transport all use ICCG to solve the transport equations. We have been using the ICCG method² to solve the diffusion equation with a nine-point coupling³ scheme on the CDC 7600. In going from the CDC 7600 to the Cray-1, a large part of the algorithm consists of solving tridiagonal linear systems on each L line

of the Lagrangian mesh in a manner which is not vectorizable. Therefore a direct translation from the 7600 to the Cray would not give much increase in running speed because the vectorization potential of the Cray cannot be used. We have developed an alternate ICCG algorithm for the Cray-1 which utilizes a block form of the cyclic reduction algorithm. This new algorithm allows full vectorization and runs as much as five times faster than the old algorithm on the Cray-1. It is now being used in Cray LASNEX to solve the two-dimensional diffusion equation in all the physics subroutines mentioned above.

*Work performed under the auspices of the U.S. Dept. of energy by the Lawrence Livermore National Laboratory under contract W-7405-ENG-48.

TABLE OF CONTENTS

I.	A Vector Algorithm for Tridiagonal Linear Systems	
	A. Basic Algorithm	
	B. Implementation on the Cray 1	
	C. Performance	
II.	A Vector ICCG Algorithm	
	A. Basic Algorithm	
	B. Array Storage	
	C. Minimization of Bandwidth	
	D. Incomplete Cyclic Reduction	
	E. Performance	
III.	Cyclic Reduction on Future Machines	

PART I

SOLVING TRIDIAGONAL LINEAR SYSTEMS ON THE
CRAY-I COMPUTER AT VECTOR SPEEDS

When we examine the numerical algorithms used to solve the physics equations in codes which model laser fusion we find that a large number of subroutines require the solution of tridiagonal linear systems of equations. Radiation transport, thermal- and suprathermal-electron transport, ion thermal conduction, charged-particle and neutron transport, all require the solution of tridiagonal systems of equations. The standard algorithm that has been used in the past on CDC 7600's will not vectorize and so cannot take advantage of the large speed increases possible on the Cray-1 through vectorization. There is, however, an alternate algorithm for solving tridiagonal systems, called cyclic reduction, which allows for vectorization, and which is optimal for the Cray-1. Software based on this algorithm is now being used in LASNEX to solve tridiagonal linear systems in the subroutines mentioned above. The new algorithm runs as much as five times faster than the standard algorithm on the Cray-1.

A. The Basic Algorithm.

Consider a tridiagonal linear system of equations

$$b_{i-1}x_{i-1} + a_i x_i + c_i x_{i+1} = y_i, \quad i = 1, 2, \dots, n, \quad (1)$$

with $b_0 = c_n = 0$. The standard algorithm for solving these equations is

$$d_i = (a_i - b_{i-1}c_{i-1})^{-1} \quad \text{for } i = 1, 2, \dots, n \quad (2)$$

and

$$l_i = b_i d_i \quad \text{for } i = 1, 2, \dots, n-1, \quad (3)$$

followed by

$$w_i = y_i - l_{i-1} w_{i-1} \quad i = 1, 2, \dots, n \quad (4)$$

and

$$x_i = d_i (w_i - c_i x_{i+1}), \quad i = n, (n-1), \dots, 3, 2, 1. \quad (5)$$

This algorithm is recursive and cannot be vectorized, and therefore will only run about twice as fast on the Cray-1 as it did on the CDC 7600.

Note that the standard scalar algorithm is just LU decomposition.¹

If our original equation is written in matrix notation as $MX = Y$, with

$$M = \begin{pmatrix} a_1 & c_1 & 0 & 0 \\ b_1 & a_2 & c_2 & 0 \\ 0 & b_2 & a_3 & c_3 \\ 0 & 0 & b_3 & a_4 \end{pmatrix}, \quad (6)$$

then we may decompose M as a lower-triangular-matrix L with unit diagonal elements and an upper-triangular-matrix U , i.e., $M = LU$. Here

$$L = \begin{pmatrix} 1 & 0 & 0 & 0 \\ l_1 & 1 & 0 & 0 \\ 0 & l_2 & 1 & 0 \\ 0 & 0 & l_3 & 1 \end{pmatrix} \quad (7)$$

and

$$U = \begin{pmatrix} d_1^{-1} & c_1 & 0 & 0 \\ 0 & d_2^{-1} & c_2 & 0 \\ 0 & 0 & d_3^{-1} & c_3 \\ 0 & 0 & 0 & d_4^{-1} \end{pmatrix}. \quad (8)$$

Then we solve $LW = Y$ and $UX = W$ to find X .

To obtain an algorithm which allows some vectorization we reorder the rows and columns of our matrix so that first we take all the odd multiples of 1, then all the odd multiples of 2, then all the odd multiples of 2^2 , etc. Thus we apply a permutation P which takes the original ordering, $1, 2, 3, \dots, n$ into the new ordering

$$1, 3, 5, 7, \dots, 2, 6, 10, 14, \dots, 4, 12, 20, 28, \dots, 8, 24, 40, 56, \dots, \\ 2^q, 3 \cdot 2^q, 5 \cdot 2^q, 7 \cdot 2^q, \dots, 2^p. \quad (9)$$

Here p is the highest power of 2 with $2^p \leq n$.

Our original matrix equation now becomes

$$(PMP^{-1})(PX) = (PY). \quad (10)$$

If we perform LU decomposition on this reordered matrix we obtain a new algorithm called cyclic reduction. This algorithm is vectorizable because we are eliminating all the odd variables first (which are not coupled to each other), then all the odd-multiple-of-2 variables (which are not coupled to

each other), and so on. Furthermore, the fact that the algorithm is just LU decomposition on a permuted matrix tells us that it is just as stable as the standard scalar algorithm, a virtue not shared by other methods for vectorizing tridiagonal systems such as recursive doubling.

Applying the standard formulas for LU decomposition, and eliminating those terms which are zero because of the particular sparsity pattern of our reordered matrix, we obtain

$$\tilde{x}_i^0 = b_i, \tilde{u}_i^0 = c_i \text{ and } \tilde{d}_i^0 = a_i, i = 1, \dots, (n-1), n. \quad (11)$$

Then for each $q = 0, 1, 2, 3, \dots, p$ we take

$$\begin{aligned} d_{(2i-1)}^q &= 1/\tilde{d}_{(2i-1)}^q, & 1 \leq 2i-1 \leq r(q, n), \\ x_{(2i-1)}^q &= \tilde{x}_{(2i-1)}^q d_{(2i-1)}^q, & 1 \leq 2i-1 < r(q, n), \\ x_{(2i)}^q &= \tilde{x}_{(2i)}^q, & 2 \leq 2i < r(q, n), \\ u_{(2i-1)}^q &= \tilde{u}_{(2i-1)}^q, & 1 \leq 2i-1 < r(q, n), \\ u_{(2i)}^q &= \tilde{u}_{(2i)}^q d_{(2i+1)}^q, & 2 \leq 2i < r(q, n), \\ \tilde{d}_i^{q+1} &= \tilde{d}_{2i}^q - x_{(2i-1)}^q u_{(2i-1)}^q - x_{(2i)}^q u_{(2i)}^q, \\ & & 1 \leq i \leq r(q+1, n), \\ \tilde{x}_i^{(q+1)} &= -x_{2i}^q x_{(2i+1)}^q, & 1 \leq i < r(q+1, n), \\ \tilde{u}_i^{(q+1)} &= -u_{2i}^q u_{(2i+1)}^q, & 1 \leq i < r(q+1, n), \end{aligned} \quad (12)$$

where $r(q,n)$ is the largest integer, r , such that $r \cdot 2^q \leq n$.

For the solve, we let

$$w_i^0 = y_i, \quad i = 1, 2, \dots, n.$$

Then for each $q = 0, 1, 2, \dots, (p-1)$ we take

$$w_i^{(q+1)} = w_{(2i)}^q - \ell_{(2i-1)}^q w_{(2i-1)}^q - u_{(2i)}^q w_{(2i+1)}^q$$

for $i = 1, 2, \dots, r(q+1, n)$. (13)

We set $x_1^p = d_1^p w_1^p$, and then for each $q = (p-1), (p-2), \dots, 4, 3, 2, 1, 0$ we take

$$x_{(2i)}^q = x_i^{(q+1)}, \quad 2 \leq 2i \leq r(q, n) \quad (14)$$

and

$$x_{(2i-1)}^q = d_{(2i-1)}^q [w_{(2i-1)}^q - \ell_{(2i-2)}^q x_{(2i-2)}^q - u_{(2i-1)}^q x_{(2i)}^q], \quad 1 \leq 2i-1 \leq r(q, n). \quad (15)$$

Finally, $x_i = x_i^0$, for $i = 1, \dots, n$.

Clearly all the above operations at the q level of reduction are vectorizable with vector length $r(q+1, n)$. Thus we now have a vectorizable algorithm with vectorization on lengths $n/2, n/4, n/8, \dots, 1$.

B. Implementation on the Cray-1.

The numbers $r(q,n)$ are easily calculated from $r(0,n) = n$ and

$$r(q+1,n) = \text{SHIFTR}[r(q,n),1] , \quad (16)$$

where $\text{SHIFTR}(n_1, n_2)$ is n_1 (in binary notation) shifted right n_2 positions with the rightmost n_2 bits of n_1 lost off the end. $\text{SHIFTR}(r,1) = r/2$ but on the Cray-1 the shift operation is much faster than the integer divide.

It is important to give consideration to storage layout and possible memory bank conflicts. Consider for example the forward sweep of the solve. The "natural" storage layout is to store $w_i^{(q+1)}$ in the $[i \cdot 2^{(q+1)}]$ element of the Y array and just keep overwriting $w_{(2i)}^q$ with $w_i^{(q+1)}$. The problem with this is that vector reads from and writes to memory in increments of 2^q on the Cray-1 cause bank conflicts and loss of speed if $q > 2$. For $q = 0,1,2$ we can read or write one word per clock period. For $q = 3$ this degrades to one word every two clock periods and for $q \geq 4$ the performance degrades to one word every four clock periods. To eliminate this problem we use the following storage scheme.

$w_i^0 = y_i$ is stored in the first $n = r(0,n)$ elements of the Y array. w_i^1 is stored in the next $r(1,n)$ elements of the Y array, and so on. Thus

$$w_i^q = y_{[s(q,n)+i]} , \quad (17)$$

where $s(0,n) = 0$ and $s(q+1, n) = s(q,n) + r(q,n)$. x_i^q is stored in the same locations as w_i^q . The matrix elements x_i^q and $\tilde{x}_i^q$ are stored in $b[s(q,n) + i]$, $\tilde{u}_i^q$ and u_i^q are stored in $c[s(q,n) + i]$, d_i^q and $\tilde{d}_i^q$ are stored in $a[s(q,n)+i]$. With this storage scheme all memory reads and writes are done in increments of 1 and 2, and all bank conflicts are eliminated. Since

$$\sum_{q=0}^p r(q,n) \leq \sum_{q=0}^p (n/2^q) \leq 2n \quad (18)$$

the arrays a, b, c , and y must now be $2n$ elements long instead of n elements long, and so we need twice as much storage as is required for the standard scalar algorithm. If this increase in storage is unacceptable it is possible to shuffle the elements in place and have no increase in storage for y and a but at the expense of the extra time required for the shuffle. For b and c (x and u) the storage must double because both $x_{(2i)}^q$ and $x_{(2i-1)}^q$ and $u_{(2i)}^q$ and $u_{(2i-1)}^q$ are needed for the solve. Thus on the forward sweep of the solve, after calculating w_i^1 , with $i=1,2,\dots,r(1,n)$, we could store w_i^1 in the first $r(1,n)$ words of y_i and move (shuffle) $w_{(2i-1)}^0$ to the last $(n-r(1,n))$ words of y_i . Then w_i^2 is stored in the first $r(2,n)$ words of y_i while $w_{(2i-1)}^1$ is moved to the next $(r(1,n)-r(2,n))$ words of y_i , etc. (The storage for the d_i^q array may be similarly arranged.) With this scheme the storage requirement will be only 50% greater instead of 100%.

C. Performance.

This algorithm was hand-coded for the Cray-1 by R. E. von Holdt for both the symmetric and nonsymmetric matrix cases. In Fig. 1 the hand coded

vector cyclic reduction algorithm is compared with the Cray-1 FORTRAN CFT-compiled standard scalar algorithm. N is the problem dimension and (t_s/t_v) is the relative execution time of the scalar and vector algorithms. The cyclic reduction algorithm was also coded with CFT and it was asymptotically about 4.5 times faster than the CFT scalar algorithm; thus the hand coding made only a 50% improvement over CFT.

PART II

A VECTORIZABLE INCOMPLETE CHOLESKY-CONJUGATE GRADIENT (ICCG) ALGORITHM FOR THE CRAY-1 COMPUTER

In LASNEX, the two-dimensional radiation transport, electron thermal conduction, ion thermal conduction and neutron transport all use ICCG to solve the transport equations. We have been using the ICCG method² to solve the diffusion equation with a nine-point coupling³ scheme on the CDC 7600. In going from the CDC 7600 to the Cray-1, a large part of the algorithm consists of solving tridiagonal linear systems on each L line of the Lagrangian mesh in a manner which is not vectorizable. Therefore a direct translation from the 7600 to the Cray would not give much increase in running speed because the vectorization potential of the Cray cannot be used. We have developed an alternate ICCG algorithm for the Cray-1 which utilizes a block form of the cyclic reduction algorithm described in the previous article. This new algorithm allows full vectorization and runs as much as five times faster than the old algorithm on the Cray-1. It is now being used in Cray LASNEX to solve the two-dimensional diffusion equation in all the physics subroutines mentioned above.

A. Basic Algorithm

We have an equation, $MX = Y$, where M is positive definite and symmetric, and the sparsity pattern is such that $M_{i,i}$, $M_{i,(i+1)}$, $M_{i,(i+KMAX)}$, $M_{i,(i+KMAX+1)}$ are the only nonzero elements.

The ICCG method consists in finding an approximate Cholesky decomposition for $M \approx LDL^T$, and then using the conjugate gradient algorithm. Let $r_0 = Y - MX_0$ and $p_0 = (LDL^T)^{-1}r_0$, then

$$a_i = [r_i, (LDL^T)^{-1}r_i] / (p_i, Mp_i),$$

$$x_{i+1} = x_i + a_i p_i,$$

$$r_{i+1} = r_i - a_i Mp_i,$$

$$b_i = [r_{(i+1)}, (LDL^T)^{-1}r_{(i+1)}] / [r_i, (LDL^T)^{-1}r_i],$$

$$p_{i+1} = (LDL^T)^{-1}r_{(i+1)} + b_i p_i,$$

for $i = 0, 1, 2, \dots$

This algorithm is trivially vectorizable except for the evaluation of the approximate inverse on the residual, $(LDL^T)^{-1}r_i$. With the choice for the approximate Cholesky decomposition given in Ref. 2 this is a recursive operation which can only go at scalar speeds on the Cray-1. To improve it we must use a different approximate decomposition which allows for vectorization.

To see how this is done we first write the exact Cholesky decomposition in block form, in which our matrix M can be written

$$M = \begin{pmatrix} A_1 & B_1^T & 0 & 0 \\ B_1 & A_2 & B_2^T & 0 \\ 0 & B_2 & A_3 & B_3^T \\ 0 & 0 & B_3 & A_4 \end{pmatrix},$$

where the A_i and B_i are tridiagonal matrices of dimension KMAX, and $i = 1, 2, 3, \dots, LMAX$. The standard block Cholesky decomposition may now be written $M = LDL^T$, where

$$L = \begin{pmatrix} L_1 & 0 & 0 & 0 \\ C_1 & L_2 & 0 & 0 \\ 0 & C_2 & L_3 & 0 \\ 0 & 0 & C_3 & L_4 \end{pmatrix},$$

and

$$D = \begin{pmatrix} D_1 & 0 & 0 & 0 \\ 0 & D_2 & 0 & 0 \\ 0 & 0 & D_3 & 0 \\ 0 & 0 & 0 & D_4 \end{pmatrix}.$$

Here the L_i are lower triangular matrices whose diagonal elements are all 1, the D_i are diagonal matrices, and L_i, C_i, D_i are given recursively by

$$C_i = B_i L_i^{-T} D_i^{-1} \quad (19)$$

and

$$L_i D_i L_i^T = A_i - C_{(i-1)} D_{(i-1)} C_{(i-1)}^T = \tilde{A}_i \quad (20)$$

for $i = 1, 2, 3, \dots, LMAX$. We use the notation $L^{-T} = (L^T)^{-1}$.

If we write the unknown and right-hand-side vectors in block form,

$$X = \begin{pmatrix} X_1 \\ X_2 \\ X_3 \\ \vdots \\ \vdots \\ \vdots \end{pmatrix} \quad Y = \begin{pmatrix} Y_1 \\ Y_2 \\ Y_3 \\ \vdots \\ \vdots \\ \vdots \end{pmatrix}$$

where X_i, Y_i are vectors of length $KMAX$, then $LDL^T X = Y$ may be solved by a forward sweep,

$$W_i = L_i^{-1} [Y_i - B_{(i-1)} L_{(i-1)}^{-T} D_{(i-1)}^{-1} W_{(i-1)}] , \quad (21)$$

for $i = 1, 2, 3, \dots, LMAX$, and a backward sweep,

$$X_i = L_i^{-T} D_i^{-1} [W_i - L_i^{-1} B_i^T X_{(i+1)}] , \quad (22)$$

for $i = LMAX, (LMAX-1), \dots, 3, 2, 1$.

To obtain an algorithm which allows vectorization we again apply the cyclic reduction permutation but we apply it only to the KMAX by KMAX blocks and within a block we retain the standard ordering.⁴ Thus the blocks are now ordered $L = 1, 3, 5, 7, \dots, 2, 6, 10, 14, \dots, 2^p$. Here p is the highest power of 2 with $2^p \leq LMAX$. Within a block the elements are still ordered $K = 1, 2, 3, 4, \dots, KMAX$.

As in Part I our equation $MX = Y$, becomes

$$(PMP^{-1})(PX) = (PY), \quad (23)$$

and we now perform block LU decomposition of this block permuted matrix.

Applying the standard formulas for block LU decomposition and retaining only the non zero blocks we obtain the following algorithm: Let

$$B_i^0 = B_i, \quad A_i^0 = A_i, \quad i = 1, 2, \dots, LMAX.$$

Then for each $q = 0, 1, 2, 3, \dots, p$ we take

$$L_{(2i-1)}^q D_{(2i-1)}^q (L_{(2i-1)}^q)^T = A_{(2i-1)}^q, \quad 1 \leq 2i-1 \leq r(q, LMAX), \quad (24)$$

$$C_{(2i-1)}^q = B_{(2i-1)}^q (L_{(2i-1)}^q)^{-T} (D_{(2i-1)}^q)^{-1}, \quad 1 \leq 2i-1 < r(q, LMAX), \quad (25)$$

$$C_{(2i)}^q = (D_{(2i+1)}^q)^{-1} (L_{(2i+1)}^q)^{-1} B_{(2i)}^q, \quad 2 \leq 2i < r(q, LMAX), \quad (26)$$

$$\begin{aligned} A_i^{(q+1)} &= A_{(2i)}^q - C_{(2i-1)}^q D_{(2i-1)}^q (C_{(2i-1)}^q)^T \\ &\quad - (C_{(2i)}^q)^T D_{(2i+1)}^q C_{(2i)}^q, \quad 1 \leq i \leq r(q+1, LMAX), \end{aligned} \quad (27)$$

$$B_i^{(q+1)} = -C_{(2i+1)}^q D_{(2i+1)}^q C_{(2i)}^q \quad 1 \leq i < r(q+1, LMAX), \quad (28)$$

where $r(q, LMAX)$ is the largest integer, r , such that $r \cdot 2^q \leq LMAX$. This completes the decomposition.

For the solve we let

$$\tilde{w}_i^0 = Y_i, \quad i = 1, 2, 3, \dots, LMAX.$$

Then for each $q = 0, 1, 2, 3, \dots, (p-1)$ we take

$$w_{(2i-1)}^q = (L_{(2i-1)}^q)^{-1} \tilde{w}_{(2i-1)}^q, \quad 1 \leq 2i-1 \leq r(q, LMAX), \quad (29)$$

$$\tilde{w}_i^{(q+1)} = \tilde{w}_{(2i)}^q - C_{(2i-1)}^q w_{(2i-1)}^q - (C_{2i}^q)^T w_{(2i+1)}^q, \quad 1 \leq i \leq r(q+1, LMAX), \quad (30)$$

and finally $w_1^p = (L_1^p)^{-1} \tilde{w}_1^p$.

This completes the forward sweep.

For the backward sweep we set

$$x_1^p = (L_1^p)^{-T} (D_1^p)^{-1} w_1^p,$$

and then for $q = (p-1), (p-2), \dots, 3, 2, 1, 0$ we take

$$x_{(2i)}^q = x_i^{(q+1)}, \quad 1 \leq i \leq r(q+1, LMAX), \quad (31)$$

followed by

$$\begin{aligned}
x_{(2i-1)}^q &= (L_{(2i-1)}^q)^{-T} ((D_{(2i-1)}^q)^{-1} w_{(2i-1)}^q - C_{(2i-2)}^q x_{(2i-2)}^q \\
&\quad - (C_{(2i-1)}^q)^T x_{2i}^q), \quad 1 \leq (2i-1) \leq r(q, \text{LMAX}), \quad (32)
\end{aligned}$$

and finally $x_i = x_i^0$ for $i = 1, 2, 3, \dots, \text{LMAX}$. This completes the solve.

Note that the $L_i^q D_i^q (L_i^q)^T$ decomposition in Eq. (24) is uniquely determined by requiring that the diagonal elements of the (KMAX by KMAX) lower triangular matrices L_i^q , all be equal to 1. The D_i^q are diagonal (KMAX by KMAX) matrices and the Cholesky decomposition theorem assures us that all of the diagonal elements of D_i^q are positive.

Now let us examine the sparsity pattern of A_i^q , L_i^q and C_i^q for the case of exact cholesky decomposition. A_i^0 is tridiagonal so $L_{(2i-1)}^0$ will be lower tridiagonal. Since B_i^0 is tridiagonal, Eqs. (25) and (26) imply that $(C_{(2i-1)}^0)_{jk}$ will be nonzero for $k+1 \geq j$, and $(C_{(2i)}^0)_{jk}$ will be nonzero for $j+1 \geq k$. Equations (27) and (28) thus imply that A_i^1 and B_i^1 are dense matrices (all elements nonzero) and so $L_{(2i-1)}^1$ is dense lower triangular and in general A_i^q and B_i^q are dense matrices. Thus we have extensive fill in.

We seek an approximate LDL^T decomposition which has no fill in. To accomplish this, in Eqs. (25) and (26) we throw away all but the tridiagonal part of C_i^q . In evaluating $(L_{(2i+1)}^q)^{-1} B_{(2i)}^q$ and $(B_{(2i-1)}^q)(L_{(2i-1)}^q)^{-T}$, we neither compute nor store any elements outside the tridiagonal bands. Then in equations (27) and (28) we throw away (i.e. neither compute nor store) all but the tridiagonal part of $C_{(2i-1)}^q D_{(2i-1)}^q (C_{(2i-1)}^q)^T$,

$$(C_{2i}^q)^T D_{(2i+1)}^q C_{(2i)}^q \text{ and } C_{(2i+1)}^q D_{(2i+1)}^q C_{(2i)}^q.$$

Thus we assure that A_i^q and B_i^q retain the same sparsity pattern as A_i and B_i and we obtain an incomplete block LDL^T decomposition.

Having obtained the incomplete LDL^T decomposition we perform the forward and backward sweep of the solve exactly as in Eqs. (29) thru (32) except that in Eqs. (30) and (32), $C_{(2i-1)}^q$ is replaced by

$B_{(2i-1)}^q (L_{(2i-1)}^q)^{-T} (D_{(2i-1)}^q)^{-1}$ and C_{2i}^q is replaced by

$(D_{2i+1}^q)^{-1} (L_{(2i+1)}^q)^{-1} B_{(2i)}^q$. These are no longer equivalent because, for

example, $C_{(2i-1)}^q$ is only the tridiagonal part of $B_{(2i-1)}^q (L_{(2i-1)}^q)^{-T} (D_{(2i-1)}^q)^{-1}$

which is dense upper triangular. This has several advantages. First, the C 's are now only temporary quantities used during the course of the incomplete decomposition and need not be saved. Thus, we save memory space. Second,

using $B_{(2i-1)}^q (L_{(2i-1)}^q)^{-T} (D_{(2i-1)}^q)^{-1}$ is closer to the exact LDL^T decomposition

than using the tridiagonal $C_{(2i-1)}^q$ and so we expect (and numerical experiment

confirms this expectation) to converge in fewer conjugate gradient iterations than if we had used the C 's. The only price we pay is that evaluating $(Y-CX)$

requires 3 adds and 3 multiplies per vector element whereas $(Y-(BL^{-T}D^{-1})X)$

requires 4 adds and 5 multiplies per element.

Since we are doing an incomplete decomposition there is no longer any guarantee that in Eq. (24) the pivots $(D_{(2i-1)}^q)_{jj}$ will be > 0 . Therefore, in evaluating $D_{(2i-1)}^q$ we take $(D_{(2i-1)}^q)_{jj} = |Z|$, where

$$Z = (A_{(2i-1)}^q)_{jj} - ((A_{(2i-1)}^q)_{(j-1)j})^2 / (D_{(2i-1)}^q)_{(j-1)(j-1)}, \quad (33)$$

instead of the usual $(D_{(2i-1)}^q)_{jj} = Z$. This assures that our incomplete LDL^T will be positive definite.

Clearly all the operations at the q 'th level of reduction are vectorizable with vector length $r(q+1, LMAX)$. Thus we now have a vectorizable algorithm on vector lengths $LMAX/2, LMAX/4, LMAX/8, \dots, 1$.

B. Array Storage

For our memory usage we simply generalize the storage layout described in Part I (see Eqs. (16), (17), and (18)). On input $X_i, Y_i, M_{(i+KMAX+1)i}, M_{(i+KMAX)i}, M_{(i+KMAX-1)i}, M_{(i+1)i}$, and M_{ij} with $i = 1, 2, 3, \dots, (KMAX * LMAX)$ are stored as 7 vectors, each of length $(2 * KMAX * LMAX)$, except for Y_i which has length $(KMAX * LMAX)$. On input, X_i has the initial guess for the solution vector. In these vectors the input numbers are stored in the first $(KMAX * LMAX)$ words while the second $(KMAX * LMAX)$ words are scratch space. Then as we proceed through the decomposition $(D_i^q)_{jj}$ is stored in M_{kk} , where

$$k = KMAX * s(q, LMAX) + i * KMAX + j, \quad (34)$$

and $s(q, LMAX)$ is defined after Eq. (17),

$(L_i^q)_{(j+1)j}$ is stored in $M_{(k+1)k}$ and $(B_i^q)_{jj}$, $(B_i^q)_{(j+1)j}$ are stored in $M_{(k+KMAX)k}$, $M_{(k+KMAX+1)k}$ respectively. As we proceed through the solve $(W_i^q)_j$ and $(X_i^q)_j$ are stored in X_k . Clearly, since i is the index on which we vectorize, we have avoided all bank conflicts on the Cray so long as $KMAX$ is odd. If $KMAX$ is not odd we simply let $KMAX' = KMAX+1$ and solve a new matrix equation $M'X' = Y'$, where if $i = n*KMAX'$ ($n = 1, 2, \dots, LMAX$), then $M'_{ij} = M'_{ji} = \delta_{ij}$ and if i and j are not multiples of $KMAX'$ then $M'_{ij} = M_{ij}$. Note also that we store $(D_i^q)^{-1}$ instead of (D_i^q) since only $(D_i^q)^{-1}$ appears in the course of the solve, and multiplies are much faster than divides on the Cray 1.

C. Minimization of Bandwidth

It was shown in reference (2) that if $KMAX > LMAX$, faster convergence of the ICCG algorithm can be obtained by first transposing the grid so we have $KMAX$ blocks of dimension $LMAX$ instead of the initial choice of $LMAX$ blocks each of dimension $KMAX$. This is true because by making the dimension of (A_i^q) as small as possible we minimize the error we make when we throw away those elements of (A_i^q) outside the tridiagonal bands. With our new vectorizable scheme transposing the grid if $KMAX > LMAX$ has the added advantage that $(LMAX/2^{(q+1)})$ which is the length of all our vector operations is made as large as possible.

D. Incomplete Cyclic Reduction

A further improvement of the algorithm may be obtained by early termination of the cyclic reduction. It has been shown⁵ that in many cases as we proceed through the decomposition algorithm (Eqs. (24) through (28)) the

elements of $B_i^{(q+1)}$ become negligible relative to the elements of $A_i^{(q+1)}$ as q increases. This suggests that instead of continuing the algorithm until $q = p$, we perform (24) through (28) for $q = 0, 1, 2, 3, \dots, (S-1)$ (where $S < p$) and then simply assume that $B_i^S = 0$. Then we take

$$L_i^S D_i^S (L_i^S)^T = A_i^S, \quad 1 \leq i \leq r(S, LMAX),$$

and we are done with the decomposition.

For the forward sweep of the solve we perform (29) and (30) for $q = 0, 1, 2, \dots, (S-1)$ and then

$$W_i^S = (L_i^S)^{-1} W_i^S, \quad 1 \leq i \leq r(S, LMAX).$$

For the backward sweep of the solve we do

$$X_i^S = (L_i^S)^{-T} (D_i^S)^{-1} W_i^S, \quad 1 \leq i \leq r(S, LMAX),$$

and then perform (31) and (32) for $q = (S-1), (S-2), \dots, 3, 2, 1, 0$.

Thus we can often obtain almost as good an approximate inverse with considerably less computational effort. Furthermore, we have eliminated just those computations which have vector lengths of $(LMAX/2^q)$, where $q > S$, and these are just the very short vector computations which will run at not much better than scalar speeds on the Cray 1. Finally if our assumption that $B_i^S = 0$ is wrong, it will only mean that LDL^T is not as good an approximate inverse as if we had continued the cyclic reduction until $q = p$. This will only make the conjugate gradient algorithm take more iterations

until convergence is reached and the correct answer will still be obtained.

In practice one needs to experiment with one's typical mix of problems until the best compromise between fewer levels of cyclic reduction, and fewer conjugate gradient iterations is found.

E. Performance

The new algorithm has been tried out on a wide variety of test problems including the model problem from Ref. 2, a typical real problem taken from a laser fusion simulation and a set of M matrices whose elements were randomly generated and whose mean degree of diagonal dominance was adjustable to obtain matrices with different degrees of ill conditioning.

Incomplete cyclic reduction worked very well. For all the matrices we tried it was true that the "B" blocks decreased very rapidly (in fact they decreased quadratically as predicted in Ref. 4) relative to the "A" blocks. Even for the stiffest problems it was never necessary to go beyond $S=3$ to get the fastest possible running time (see Sec. D for the definition of S).

Figure 2 shows relative running times for the new vectorizable algorithm and the scalar algorithm given in Ref. 2. $KMAX$ was held fixed at $KMAX=5$ and $LMAX$ was varied from 4 to 8000. The matrices were generated by setting each subdiagonal element $M_{(i+1)i}$, $M_{(i+KMAX)i}$, $M_{(i+KMAX-1)i}$, and $M_{(i+KMAX+1)i}$ to a different random number between -1 and 0.

This fixes the superdiagonals since M is symmetric. Then we set

$$M_{ii} = \alpha * RANF - \sum_j M_{ji} ,$$

where the summation is over all the off diagonal elements in the i 'th column, $RANF$ is a random number between 0 and 1, and α is a stiffness parameter. By construction M is a diagonally dominant M matrix and so is positive definite.⁶ We chose a random X vector, found Y from

$$Y = MX,$$

and then solved $Y = MX$ using ICCG until $\|x^n - x\| / \|x\| \leq 10^{-7}$, where x^n was the ICCG solution for X after n iterations and $\|x\|$ was the

Euclidean norm. Both the old and new algorithms were written in fortran, compiled with CFT and run on the CRAY 1 computer. The new algorithm was completely vectorized. The old algorithm was completely vectorized except for the incomplete factorization,

$$LDL^T \approx M$$

and the incomplete solve

$$(LDL^T)^{-1} r_i .$$

These operations are recursive and cannot be vectorized. The ratio of the CPU times is shown in Fig. 2 for three different values of α . Asymptotically the new algorithm runs 3 to 5 times faster than the old. As the problem got stiffer the new algorithm tended to take a few more iterations than the old. This is why the speed increase is less for small α .

PART III

CYCLIC REDUCTION ON FUTURE MACHINES

On the CRAY 1 computer the algorithms presented in Parts I and II run 3 to 7 times faster than their scalar counterparts. The CRAY 1 is a pipeline machine with no multiprocessing. There is only one vector add unit, one vector multiply unit, etc. As soon as computing machines become available with true parallel processing our algorithms can take immediate advantage of them with no modifications to the algorithms. Thus on a machine with four parallel vector pipelines for each arithmetic operation, each of our vector operations could split into 4 pieces each of which went to one of the 4 pipelines. Assuming the memory was properly designed so it could feed the 4 pipelines as fast as they can compute, the algorithms presented in this article would then run 12 to 28 times faster than their scalar counterparts. In general if we have P parallel processors our vector algorithm for the tridiagonal linear system will run in $\sim(N/Q) + \log_2(Q)$ units of time (where N is the order of the linear system, and $Q = \text{MIN}(N,P)$) as compared to $\sim N$ units of time for the single pipeline ($P = 1$) machine (such as the CRAY 1). Similarly the vector block tridiagonal algorithm will run in $\sim K_{\text{MAX}} * ((L_{\text{MAX}}/Q) + \log_2(Q))$ units of time as compared to $\sim K_{\text{MAX}} * L_{\text{MAX}}$ for the CRAY 1 (here $Q = \text{MIN}(L_{\text{MAX}}, P)$).

Thus even though these algorithms already have a distinct speed advantage on current computing machines their ultimate potential will only be realized on future machines where the ratio of scalar algorithm to vector algorithm execution times will be $\sim N/(\log_2 N)$ for a tridiagonal system of order N and a machine with P parallel processors with $P \geq N$.

ACKNOWLEDGMENTS

I would like to thank Garry Rodrigue and Alex Friedman for a number of helpful discussions and Alex Friedman for his able programming of the algorithm presented in Part II.

DISCLAIMER

This document was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor the University of California nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial products, process, or service by trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or the University of California. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government thereof, and shall not be used for advertising or product endorsement purposes.

REFERENCES

1. For a good discussion of LU decomposition see J. H. Wilkinson, The Algebraic Eigenvalue Problem (Oxford University Press, Oxford, 1965) chap. 4, sec. 36.
2. D. S. Kershaw, "The ICCG Method for the Iterative Solution of Systems of Linear Equations," J. Comp. Phys. 26, 43 (1978).
3. D. S. Kershaw, Differencing of the Diffusion Equation in Lagrangian Hydrodynamics Codes, Lawrence Livermore Laboratory, Livermore, Calif., UCRL-82747 (1980), to appear in J. Comp. Phys.
4. Another possibility is to apply the cyclic reduction permutation only within each block but the blocks retain the standard ordering. Thus within each block we have $K=1,3,5,7,\dots,2,6,10,14,\dots,2^p$ but the blocks are still ordered $L=1,2,3,4,\dots, L_{MAX}$. This alternate vector algorithm is described in detail in the 1979 Laser Program Annual Report, Lawrence Livermore National Laboratory, UCRL-50021-79, sec. 3, p 79. It was found to run at roughly the same speed as the algorithm presented in this article on most of our test problems, but in this author's opinion it is considerably less elegant and much harder to understand and program.
5. D. Heller, "Some Aspects of the Cyclic Reduction Algorithm for Block Tridiagonal Linear Systems," SIAM J. Number. Anal. 13, #4, p. 484, 1976.
- 6 R. J. Plemmons, "M Matrix Characterizations," Linear Algebra and Its Applications, 18, p. 175-188, (1977).

FIGURE CAPTIONS

Fig. 1. Ratio of scalar to vector execution times for the solution of a tridiagonal matrix of length N . For both symmetric (a) and asymmetric (b) matrices, the asymptotic rate of the vector cyclic reduction algorithm is more than six times the scalar rate, but this speedup can be achieved only with relatively long vectors.

Fig. 2. Ratio of scalar to vector execution times for the solution of a 9 point coupling, block tridiagonal matrix. L_{MAX} is the number of blocks. α is a stiffness parameter.

Fig. 1. Ratio of scalar to vector execution times for the solution of a tridiagonal matrix of length N . For both symmetric (a) and asymmetric (b) matrices, the asymptotic rate of the vector cyclic reduction algorithm is more than six times the scalar rate, but this speedup can be achieved only with relatively long vectors.

Fig. 2 Ratio of scalar to vector execution times for solution of the block tridiagonal linear system. LMAX is the number of blocks. α is a stiffness parameter.

Technical Information Department · Lawrence Livermore National Laboratory
University of California · Livermore, California 94550