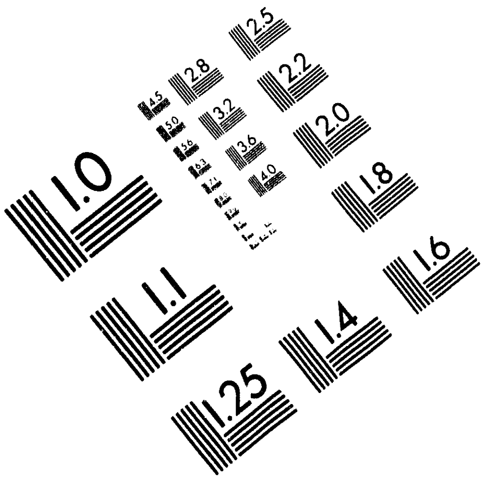
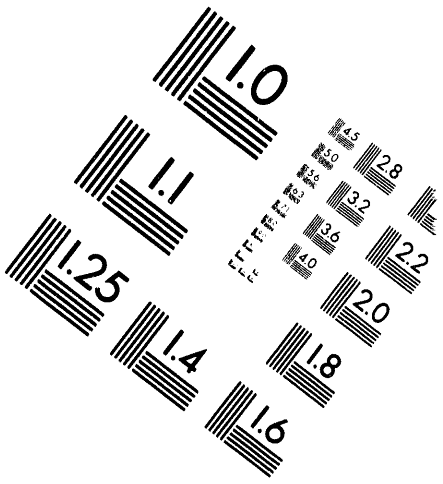
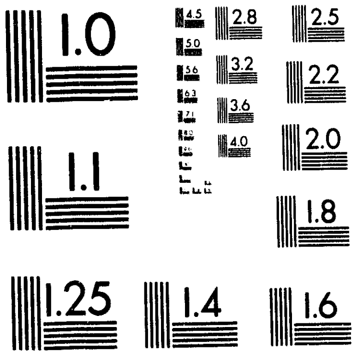
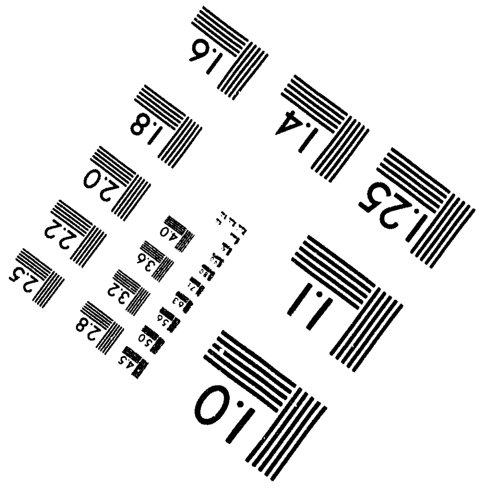
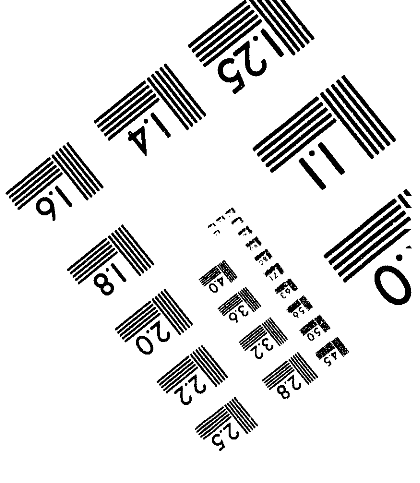




Centimeter
mm
Inches







# Two Portable Parallel Tridiagonal Solvers

Peter G. Eltgroth

## Introduction

Many scientific computer codes involve linear systems of equations which are coupled only between nearest neighbors in a single dimension. The most common situation can be formulated as a tridiagonal matrix relating source terms and unknowns. This system of equations is commonly solved using simple forward and back substitution. The usual algorithm is spectacularly ill suited for parallel processing with distributed data, since information must be sequentially communicated across all domains.

Two new tridiagonal algorithms have been implemented in FORTRAN 77. The two algorithms differ only in the form of the unknown which is to be found. The first and simplest algorithm solves for a scalar quantity evaluated at each point along the single dimension being considered. The second algorithm solves for a vector quantity evaluated at each point. The solution method is related to other recently published approaches, such as that of Bondeli (Bondeli, 1990). An alternative parallel tridiagonal solver, used as part of an Alternating Direction Implicit (ADI) scheme, has recently been developed at LLNL by Lambert (Lambert, 1994). For a discussion of useful parallel tridiagonal solvers, see the work of Mattor, et al (Mattor, 1993). Previous work appears to be concerned only with scalar unknowns. This paper presents a new technique which treats both scalar and vector unknowns. There is no restriction upon the sizes of the subdomains.

Even though the usual tridiagonal formulation may not be theoretically optimal when used iteratively, it is used in so many computer codes that it appears reasonable to write a direct substitute for it. The new tridiagonal code can be used on parallel machines with a minimum of disruption to pre-existing programming. As tested on various parallel computers, the parallel code shows efficiency greater than 50% (that is, more than half of the available computer operations are used to advance the calculation) when each processor is given at least 100 unknowns for which to solve.

The parallel tridiagonal solvers use the Macro Interface for Communication and Allocation (MICA) package (Brown & Mirin, 1994), available from NERSC at LLNL. The macros are used to access message passing calls and for some very simple memory management. The macros provide easy portability to different computers. Most of the communications is managed by high level procedures optimized for groups of processors.

The basic approach assumes that the unknowns are divided into a number of blocks of contiguous variables. Interior unknowns may be eliminated from each domain, leaving only those variables at the edges of the domain. Equations relating the edge variables can be gathered into one system, called the reduced system. This system can be solved by a single processor. Using the edge solutions, the remaining variables for each domain can then be found.

Because the scalar algorithm is a special case of the vector algorithm, we provide the solution details only for the vector algorithm. The scalar case can be recovered by replacing vectors and matrices by appropriate scalar quantities.

**Basic Equations Used to Form a Reduced System**

The equations to be solved can be written:

$$\tilde{\mathbf{A}}_i \cdot \vec{\mathbf{u}}_{i-1} + \tilde{\mathbf{B}}_i \cdot \vec{\mathbf{u}}_i + \tilde{\mathbf{C}}_i \cdot \vec{\mathbf{u}}_{i+1} = \vec{\mathbf{d}}_i \tag{1}$$

or

$$\tilde{\vec{\mathbf{u}}}_{i-1} \cdot \mathbf{A}_i + \tilde{\vec{\mathbf{u}}}_i \cdot \mathbf{B}_i + \tilde{\vec{\mathbf{u}}}_{i+1} \cdot \mathbf{C}_i = \tilde{\vec{\mathbf{d}}}_i \tag{2}$$

where $\mathbf{A}_i$, $\mathbf{B}_i$ , and $\mathbf{C}_i$. are matrices of coefficients and $\vec{\mathbf{u}}_i$ and $\vec{\mathbf{d}}_i$ are column vectors for the unknowns and sources, respectively. All are evaluated at the i-th point along the implicit coordinate. The tilde denotes transpose. The equations are shown in the two different forms because the first form is of use in the usual forward and back substitution approach, while the second is used in the local elimination phase for the distributed quantities. As written, the first matrix index for the coefficient matrices is associated with the unknown vector index, and the second matrix index is associated with the source vector index. In the case of a scalar system, the matrices on the left hand side of Equations (1) and (2) become scalar components of a tridiagonal matrix based on the i index.

Assume that a given processor is to solve for unknowns associated with contiguous indices $i$ ranging from $i=B$ (beginning) to $i = E$ (ending.) The interior indices can be successively eliminated by combining equations starting at either end of the index range using the system of equations in Eqn. (2). Going from $i=B$ to $i = E - 1$, form the dot product of each instance of Eqn.(2) with the auxiliary vector $\vec{\mathbf{g}}_i$ and then combine equations to eliminate interior variables. The following conditions must be met:

$$\mathbf{C}_{i-1} \cdot \vec{\mathbf{g}}_{i-1} + \mathbf{B}_i \cdot \vec{\mathbf{g}}_i + \mathbf{A}_{i+1} \cdot \vec{\mathbf{g}}_{i+1} = \vec{\mathbf{0}} \tag{3}$$

where the i index runs from B+1 to E-1.

Now consider the k-th component of the equation remaining after interior variables are eliminated. This sets $\vec{\mathbf{g}}_B = \hat{\mathbf{e}}_k$, the unit vector along the k-th component. Because we did not actually include the equation with index $i = E$, we have $\vec{\mathbf{g}}_E = \vec{\mathbf{0}}$ . The new equation for the k-th component can be written

$$\begin{aligned} \tilde{\vec{\mathbf{u}}}_{B-1} \cdot \mathbf{A}_B \cdot \vec{\mathbf{g}}_B + \tilde{\vec{\mathbf{u}}}_B \cdot \left(\mathbf{B}_B \cdot \vec{\mathbf{g}}_B + \mathbf{A}_{B+1} \cdot \vec{\mathbf{g}}_{B+1}\right) + \\ \tilde{\vec{\mathbf{u}}}_E \cdot \mathbf{C}_{E-1} \cdot \vec{\mathbf{g}}_{E-1} = \vec{\mathbf{D}}_{Bk} \end{aligned} \tag{4}$$

where the modified source term component in Eqn. (4) is given by

$$\vec{\mathbf{D}}_{Bk} = \tilde{\vec{\mathbf{d}}}_B \cdot \vec{\mathbf{g}}_B + \tilde{\vec{\mathbf{d}}}_{B+1} \cdot \vec{\mathbf{g}}_{B+1} + \cdots + \tilde{\vec{\mathbf{d}}}_{E-1} \cdot \vec{\mathbf{g}}_{E-1} \tag{5}$$

In an exactly analogous procedure, eliminating interior unknowns going from $i = E$ to $i = B + 1$, we can write equations relating unknowns with indices $B$, $E-1$, and $E$. The auxiliary conditions are again given by Eqn. (3), but with different "boundary" specifications, namely, $\vec{\mathbf{g}}_B = \vec{\mathbf{0}}$ and $\vec{\mathbf{g}}_E = \hat{\mathbf{e}}_k$. The k-th component of the new equation centered at index E can be written

$$\vec{\mathbf{u}}_B \cdot \mathbf{A}_{B+1} \cdot \vec{\mathbf{g}}_{B+1} + \vec{\mathbf{u}}_E \cdot \left(\mathbf{B}_E \cdot \vec{\mathbf{g}}_E + \mathbf{C}_{E-1} \cdot \vec{\mathbf{g}}_{E-1}\right) + \vec{\mathbf{u}}_{E+1} \cdot \mathbf{C}_E \cdot \vec{\mathbf{g}}_E = \vec{\mathbf{D}}_{Ek} \qquad (6)$$

where the modified source term component in Eqn. (6) is given by

$$\vec{\mathbf{D}}_{Ek} = \vec{\mathbf{d}}_E \cdot \vec{\mathbf{g}}_E + \vec{\mathbf{d}}_{E-1} \cdot \vec{\mathbf{g}}_{E-1} + \cdots + \vec{\mathbf{d}}_{B+1} \cdot \vec{\mathbf{g}}_{B+1} \qquad (7)$$

Eqns. (4) and (6) represent part of a new system which involves only unknowns at the two borders of the local domain. Considering all domains, we have a new tridiagonal system of equations, which has a number of unknowns equal to twice the number of domains. This "reduced" system can be solved and the resulting border values for each domain used to allow solutions to be found for the remaining interior unknowns. Before considering the reduced system solution, however, we will write out the procedure for determining the auxiliary vectors $\vec{\mathbf{g}}_i$ which are needed to determine the new coefficients and sources for that reduced system.

Consider Eqn. (3) with local boundary conditions $\vec{\mathbf{g}}_E = \vec{\mathbf{0}}$ and $\vec{\mathbf{g}}_B = \hat{\mathbf{e}}_k$. This is a classic tridiagonal system with straightforward solution. Write

$$\vec{\mathbf{g}}_{i+1} = \mathbf{E}_{i+1} \cdot \vec{\mathbf{g}}_i. \qquad (8)$$

By substitution, a recursion relation for the multiplier matrix is

$$\mathbf{E}_i = -\left[\mathbf{B}_i + \mathbf{A}_{i+1} \cdot \mathbf{E}_{i+1}\right]^{-1} \cdot \mathbf{C}_{i-1}. \qquad (9)$$

Starting from $\mathbf{E}_E = \mathbf{0}$, the other multiplier matrices may be found. Then, starting from $\vec{\mathbf{g}}_B = \hat{\mathbf{e}}_k$, the remaining $\vec{\mathbf{g}}_i$ are determined. The matrix inversion in Eqn.(9) can be performed by a variety of techniques. In the implementation reported on here, it is done via LU decomposition with partial pivoting and back substitution. The second set of conditions which lead to Eqn. (6) can be found using the same technique with indices running in the opposite order.

**Solving the Reduced System Equations**

The reduced system of equations given by Eqns. (4) and (6) for each subdomain can be solved using any acceptable technique on a single processor. This could be a forward and back substitution method. However, we have chosen to use a repetition of the interior unknown elimination technique, since it appears to allow a large set of boundary conditions to be treated with a uniform approach. The boundary conditions allowed in the present implementation include fixed values outside the physical interior, periodic, reflecting, and zero derivative across the boundary. Note that the boundary is a physical specification, not related to conditions across the borders of subdomains. We assume that the boundaries are located at the borders of subdomains in order to simplify the accounting procedures.

Take the collection of Eqns. (4) and (6) for each subdomain and eliminate all interior variables using the technique described in the previous section. Let us assume that the index at the left boundary is 1 and the index at the right boundary is N. The index outside the left boundary will be 0 and the index outside the right boundary will be N + 1. The equations involving the boundaries can then be written:

$$\mathbf{A}'_1 \cdot \vec{\mathbf{u}}_0 + \mathbf{B}'_1 \cdot \vec{\mathbf{u}}_1 + \mathbf{C}'_1 \cdot \vec{\mathbf{u}}_N = \vec{\mathbf{d}}'_1 \tag{10}$$

and

$$\mathbf{A}'_N \cdot \vec{\mathbf{u}}_1 + \mathbf{B}'_N \cdot \vec{\mathbf{u}}_N + \mathbf{C}'_N \cdot \vec{\mathbf{u}}_{N+1} = \vec{\mathbf{d}}'_N \tag{11}$$

The primed matrices in Eqns. (10) and (11) result from the second application of the reduction technique to the previously reduced system. Thus they are transpositions of the results from Eqns. (4) and (6) applied to the first reduced system.

Eqns. (10) and (11) involve 2n scalar unknowns where n is the number of components of the vector system. The unknowns are the n components of $\vec{\mathbf{u}}_1$ and $\vec{\mathbf{u}}_N$. Values for $\vec{\mathbf{u}}_0$ and $\vec{\mathbf{u}}_{N+1}$ are either specified or implied by the boundary conditions. For example, zero derivative across the left boundary is enforced by setting $\vec{\mathbf{u}}_0 = \vec{\mathbf{u}}_1$. The boundary equations are solved by writing a simple matrix equation for a vector consisting of all components of $\vec{\mathbf{u}}_1$ and $\vec{\mathbf{u}}_N$. This can be written as

$$\mathbf{M} \cdot \vec{\mathbf{v}} = \vec{\mathbf{d}}'' \tag{12}$$

where **M** is an appropriate block combination of the matrices appearing in Eqns. (10) and (11).

Eqn. (12) is solved using lu decomposition. When $\vec{\mathbf{u}}_1$ and $\vec{\mathbf{u}}_N$ are known, then the interior values for the reduced system are found using the "classic" technique, given below. When all reduced values are known, the appropriate ones are given to each domain and each domain can then find all of its interior values.

**The Interior Solution**

In the interior of a domain, we use Eqn. (1) to describe the system. But now the unknowns range in index from $i = B + 1$ to $i = E - 1$ The values at the ends of the full range are known from the reduced system solution. We solve for the interior unknowns by the usual substitution technique. Assume

$$\tilde{\mathbf{u}}_i = \mathbf{e}_i \cdot \tilde{\mathbf{u}}_{i+1} + \vec{\mathbf{f}}_i \tag{13}$$

Set the initial matrix $\mathbf{e}_B = \mathbf{0}$ and the initial vector $\vec{\mathbf{f}}_B = \tilde{\mathbf{u}}_B$. The remaining auxiliary values are found from the recursions

$$\mathbf{e}_i = -\left[\tilde{\mathbf{B}}_i + \tilde{\mathbf{A}}_i \cdot \mathbf{e}_{i-1}\right]^{-1} \cdot \tilde{\mathbf{C}}_i \tag{14}$$

and

$$\vec{\mathbf{f}}_i = \left[\tilde{\mathbf{B}}_i + \tilde{\mathbf{A}}_i \cdot \mathbf{e}_{i-1}\right]^{-1} \cdot \left(\vec{\mathbf{d}}_i - \tilde{\mathbf{A}}_i \cdot \vec{\mathbf{f}}_{i-1}\right) \tag{15}$$

The final values are given by Eqn.(13) starting with $\tilde{\mathbf{u}}_E$ at the right hand side.

**Strategy**

The multiprocessing technique discussed above admits several different strategies for obtaining the solution. Three will be discussed in this section. Assume that the data decomposition involves $N_d$ subdomains. The first two strategies (A and B) assume that there are $N_d$ processors to deal with an equal number of subdomains. The third strategy (C) assumes $N_d + 1$ processors.

In case A ("No overlap"), each processor prepares border information, and passes its border data to one (arbitrarily chosen, but fixed) processor to solve the reduced system. That processor, after solving the reduced system and applying boundary conditions, passes the needed border values back to the other block processors. Each block processor then determines the remaining unknowns.

In case B ("All solve"), each processor passes its border data to all other processors, and then each processor solves the reduced system. Because the reduced solve is done locally, there is no further need for communication (at the expense of a small amount of duplicated computation.) Each processor then determines the remaining unknowns.

In case C ("Overlap"), we have $N_d$ block processors and one "reduced system" processor. Each block processor prepares its border information, and passes border data to the reduced system processor. That processor, after solving the reduced system and applying boundary conditions, passes the needed border values back to the block processors. Each block processor then determines the remaining unknowns. There is a computational overlap in this case, since, while Eqn.(12) is being solved on the reduced system processor, the block processors can proceed with Eqn.(14), at the same time obtaining useful partial results for Eqn.(15).

## Performance

The solutions for tridiagonal systems of equations have been implemented into two sets of procedures, referred to as trid_solvers (scalar unknowns) and trid_solverv (vector unknowns.) The two solver modules have been tested on Cray C90, BBN TC2000, and Meiko CS-2 computers. The Cray tests have been single cpu only, while the tests on the two remaining machines have involved multiple cpus. The solvers were written so that vectorization could be exploited on machines which support that feature.

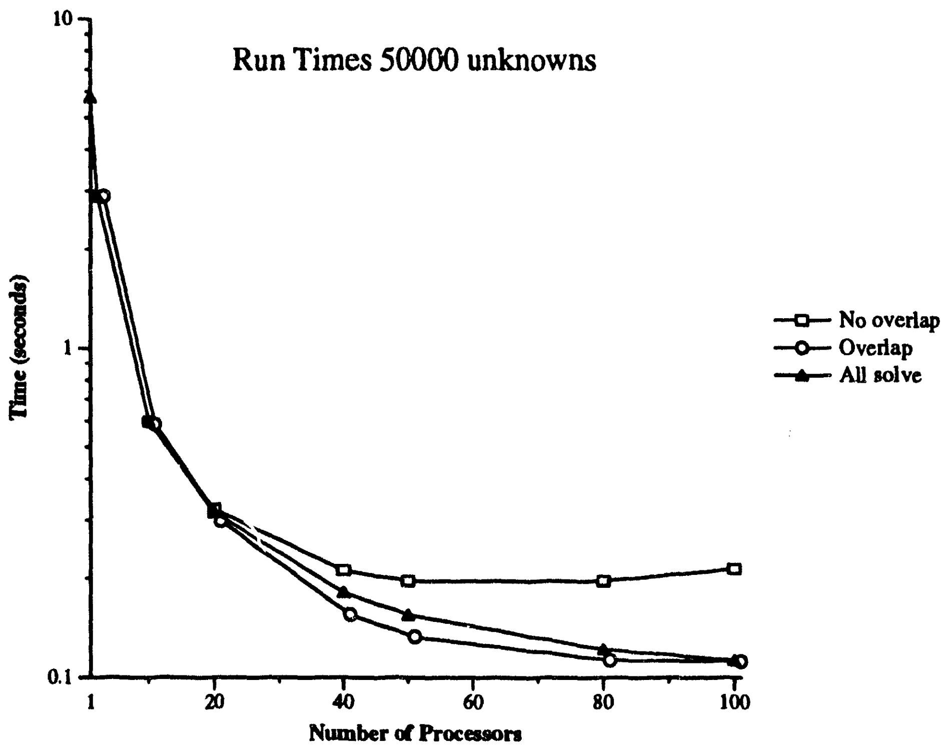


Figure 1. Execution times on the BBN TC 2000 for a simple scalar test case with equal size subdomains with a total of 50000 unknowns.

The first test problem reported on here involves a simple code for which the user specifies the number of scalar unknowns and the number of processors to be used in the solver. Subdomains are each the same size. The tridiagonal coefficients are specified using a deterministic, but problem size dependent, procedure. Boundary conditions were held to fixed values. Solutions were checked by direct substitution into the original equations. Figure 1 shows results for a series of tests on the BBN TC 2000 for the scalar unknowns solver. The accuracy of results was typically better than one part in $10^{15}$.

The two best strategies on the BBN appear to be "Overlap" and "All solve." In the former, no extra computation is performed, but one processor is dedicated to finding the reduced solution. In the latter, extra computation is performed, but a round of communication is avoided. It seems clear that optimal performance may be architecture and system dependent. The procedures described here allow the user to make a choice at run time. As a point of comparison, the single cpu optimal back substitution procedure for solving the tridiagonal system with fixed boundary values took one quarter of the time of the parallel scalar solver with general boundary conditions on a single cpu. This is a measure of the overhead involved in the parallel method. The overhead can be reduced by a significant amount, but this has not been accomplished at present.

A similar sequence of timings was carried out on the Meiko CS2 computer at LLNL. The number of processors was fewer, and the vector capabilities of the processors were not available. The timing results were similar to those in Figure 1, except that all times were smaller by approximately a factor of four.

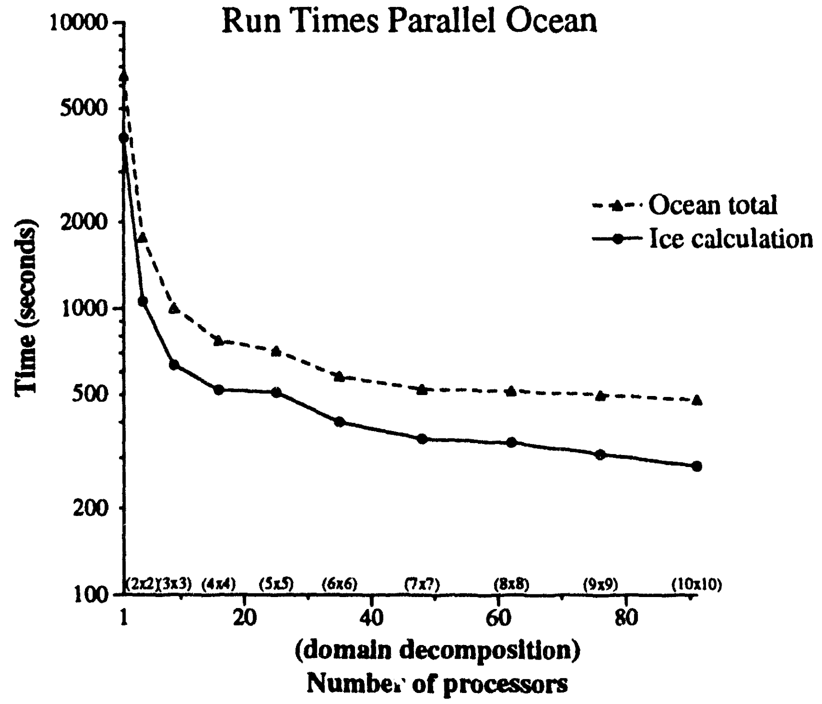


Figure 2. Execution times on the BBN TC 2000 for a production ocean simulation code. Both scalar and vector tridiagonal systems are used in the ice calculation. The ocean calculation times include the ice calculation.

The second test situation involves both solvers in a calculation of sea ice dynamics in an Earth System Model developed at LLNL. The scalar solver was used to determine three quantities (e.g., ice thickness), while the vector solver was used to find one two-component vector (horizontal ice mass flux.) Both solvers were used within a semi-implicit iteration scheme, which had its own communication and synchronization requirements arising from convergence criteria. Test runs were for a 20 day period involving the entire oceanic surface with thousands of zones. The decomposition was approximately regular, but not uniform. Land zones were omitted when possible.

Boundary conditions were either fixed or periodic. Again, solutions were checked by direct substitution into the basic equations. The timings were taken on the entire ice dynamics calculation, and thus include the overhead of iteration control and data preparation.

Figure 2 shows the times observed for an ocean simulation, including the complete ice calculation of both dynamics and thermodynamics. The ice calculation times are dominated by the dynamics, which is, in turn, dominated by one vector and three scalar tridiagonal system solutions. The decomposition refers to the number of subdomains as longitude x latitude. The number of processors is not equal to the product of the number of longitudinal subdivisions by the number of latitudinal subdivisions, since processors lying entirely over land are omitted from the work force. The longitude resolution is coarse (4°), so that the number of unknowns (at a given latitude) assigned to a single processor ranges from 90 to 9. The timing results are quite satisfactory, showing continued performance improvement for the tridiagonal section, even after the standard hydrodynamics (involving a global elliptic system solver) has begun to get worse with an increase in the number of processors. No attempt was made to obtain timings in dedicated mode, which may explain the apparent perturbations in the timing curves.

Some timing results were also obtained for single processor runs on Cray C90 computers. These were compared with the very highly optimized (non parallel) code originally used for the ice dynamics. Because the parallel coding could not assume too much about memory layout, the vector lengths were shorter in the parallel code. As a result, the C90 times for the parallel code were approximately 15% longer than for optimal single cpu Fortran. However, the optimized single cpu code used a solver unable to handle periodic boundary conditions. Instead, the original code used data copying and shifts to approximate and iterate to a satisfactory periodic condition. The new parallel tridiagonal solvers directly solve for the periodic conditions. As a consequence, a significant amount of data movement can be avoided. Using the direct periodic boundary conditions in the parallel tridiagonal solver (running on a single C90 cpu), solutions were obtained with run times approximately 10% faster than the times of the original code.

## Source Code

There are four different collections of source code which are needed to use the tridiagonal solvers discussed in this report. The first two are trid_solvers and trid_solverv, which contain the high level operations needed to carry out solutions for scalar and vector unknowns, respectively. The third is a collection of procedures which carry out the collective communications needed by the solvers. The collective communications routines were provided by B. K. Grant of LLNL. The fourth code source is the Macro Interface for Communication and Allocation (MICA) package, which permits portable source code for different platforms. These procedures were provided by A. Mirin and J. Brown of NERSC.

The first three collections of subroutines can be obtained directly from the author (eltgroth@llnl.gov.) The procedures have been released for dissemination by LLNL under the name TRID_SOLVER, code release number 940032. The MICA package has been made available by anonymous ftp using the procedures outlined in the report by Mirin and Brown.

The user interface is a single call by all involved procedures. The input parameters specify the data and the role which the calling processor is to take in the solution process. The answers are placed into a user supplied array. All memory allocation is performed by the user before the solver is invoked. The source code for the solvers is extensively commented. Sample makefiles are provided.

## Conclusions

Two parallel tridiagonal system solvers have been written in FORTRAN. The solvers calculate results for scalar or vector unknowns which are coupled to previous and next values along one dimension. The solvers have been tested in both simple cases and a realistic global simulation code. The solvers show favorable scaling properties as the numbers of subdomains and processors increases. The solvers have been written in modular and portable form. They are available for use by others.

## Acknowledgments

This work was partially supported by the U.S. Department of Energy Computer Hardware, Advanced Mathematics and Model Physics (CHAMMP) Program. Work performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract W-7405-Eng-48.

The author wishes to thank B. K. Grant for providing the collective communication procedures. He also thanks A. Mirin and J. Brown for their portable macro package MICA. Finally, he thanks A. Bourgeois for assistance in the use of these solvers in an oceanic general circulation model.

## References

Bondeli, S. (1990). Divide and Conquer: A New Parallel Algorithm for the Solution of a Tridiagonal Linear System of Equations. In H. Burkhardt (Ed.), Lecture Notes in Computer Science, No. 457 Springer-Verlag.

Brown, J. C., & Mirin, A. A. (1994). MICA, a Facility to Achieve Portability for Message-Passing and Dynamic Memory Management in Fortran. NERSC Buffer, 18(2), 12.

Lambert, M. (1994). Private Communication.

Mattor, N., T. J. Williams, and D. W. Hewett (1993). Algorithm for Solving Banded Diagonal Matrix Problems in Parallel (Preprint No. UCRL-JC-114756). Lawrence Livermore National Laboratory.

DATE
FILMED
10/20/94
END