A MEMORY EFFICIENT PARALLEL TRIDIAGONAL SOLVER

TRAVIS M. AUSTIN[†], MARKUS BERNDT[†], AND J. DAVID MOULTON[†]

Abstract. We present a memory efficient parallel algorithm for the solution of tridiagonal linear systems of equations that are diagonally dominant on a very large number of processors. Our algorithm can be viewed as a parallel partitioning algorithm. We illustrate its performance using some examples. Based on this partitioning algorithm, we introduce a recursive version that has logarithmic communication complexity.

Key words. parallel partition method, tridiagonal linear systems, parallel linear algebra

AMS subject classifications. 15A06, 65F05, 65F50, 65Y05

1. Introduction.

Large tridiagonal systems of linear equations appear in many numerical analysis applications. In our work, they arise in line relaxations needed by robust multigrid methods for structured grid problems [6, 7, 12]. Using this as our motivation, we present a new memory efficient partitioning algorithm for the solution of diagonally dominant tridiagonal linear systems of equations. This partitioning algorithm is well suited for distributed memory parallel computers. For simplicity, we assume in this paper that each processor has roughly the same number of subsequent rows of the tridiagonal system, and the number of processors N_P is strictly less than the number of unknowns N. Note however that our algorithm can be applied to the case $N_P = N$.

On a serial computer, Gaussian elimination without pivoting can be used to solve a diagonally dominant tridiagonal system of linear equations in $O(N)$ steps. This algorithm, first described in [15], is commonly referred to as the Thomas algorithm. Unfortunately, this algorithm is not well suited for parallel computers. The first parallel algorithm for the solution of tridiagonal systems was developed by Hockney and Golub and described in 1965 in [9]. It is now usually referred to as cyclic reduction. Stone introduced his recursive doubling algorithm in 1973 [13]. Both cyclic reduction and recursive doubling are designed for fine grained parallelism, where each processor owns exactly one row of the tridiagonal matrix. In 1981, Wang proposed a partitioning algorithm that is aimed at more coarse-grained parallel computation, where $N_P << N$ [17] . Diagonal dominance of the resulting reduced system in Wang's method was established in [16] and numerical stability of Wang's algorithm was analyzed in [18]. A unified approach for the derivation and analysis of partitioning methods is given in [1, 2]. There has also been attention directed towards a parallel partitioning of the standard LU algorithm. In 1986, Sun et al. [14] introduced the parallel partitioning LU algorithm that is very similar to Bondeli's divide and conquer algorithm [4].

For both partitioning algorithms and divide and conquer algorithms, a reduced tridiagonal system of interface equations must be solved. Each processor owns only a small number of rows in this reduced system. As an example, in Wang's partitioning algorithm each processor owns one row of the reduced system. In [10], this reduced system is solved by recursive doubling. However, numerical experiments were performed only on very small numbers of processors. We target parallel computers with thousands to tens of thousands of processors, such that for a 2D structured grid, line

[†]Mathematical Modeling and Analysis Group, Theoretical Division, Mail Stop B284, Los Alamos National Laboratory, Los Alamos, NM 87545, Email: {taustin,berndt,moulton}@lanl.gov. Supported by the U.S. Department of Energy, under contract W-7405-ENG-36. LA-UR-04-4149.

solves spanning hundreds of processors are realistic.

In the remainder, we proceed as follows. In the next section, we describe in more detail our motivation for introducing a new memory efficient parallel tridiagonal solver. Section 3 contains an overview of the existing parallel tridiagonal solvers with their storage requirements in the context of line relaxation. The main section is Section 4, where we describe our new memory efficient parallel tridiagonal solver. We present detailed timing and efficiency results for our line relaxation in Section 5. This section is followed by a description of our recursive partitioning algorithm and an analysis of its complexity. We end with concluding remarks.

2. Motivation. Multigrid methods gained recognition in the late 1970's as an efficient algorithm for the solution of the discrete linear systems that arise from models of diffusive phenomena (e.g., heat conduction, neutron diffusion, single-phase saturated flow). These methods achieve their efficiency through the recursive use of successively coarser discrete problems in conjunction with smoothing on each level to damp the highly oscillatory errors associated with each grid. Unfortunately, early multigrid algorithms were fragile, with their efficiency strongly dependent on the variability of the model's coefficients.

Considerable research in the early 1980's [6, 8] led to the first multigrid algorithms that could be used reliably for a large class of practical problems. The key to the success of these robust *Black Box* methods, was the use of the fine-scale discrete operator to construct, through a variational principle, the successively coarser coarse-grid operators. To achieve robustness in this variational coarsening approach, line relaxation in 2D and plane relaxation in 3D are a necessary ingredient. In practice, plane relaxation is performed by using one cycle of a 2D variational coarsening multigrid code. As a result, in both 2D and 3D, the computational workhorse of these multigrid methods is line relaxation, or in other words, the solution of tridiagonal linear systems.

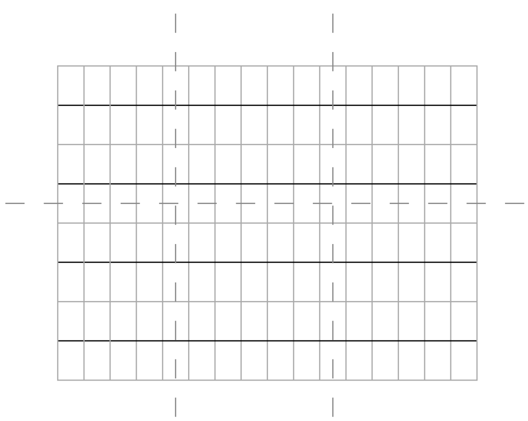

FIG. 2.1. *Example of a parallel structured grid, dashed lines indicate processor boundaries. Line relaxations in x-direction along lines of the same color are independent of each other and can be performed simultaneously on the bottom and the top group of three processors.*

The original black box multigrid code developed by Dendy in the 1980's was for serial computers [6]. Recently, the authors have implemented an MPI-based version for both 2D and 3D structured grid problems [3]. Just as with its serial counterpart, the workhorse for both versions of the parallel code is line relaxation. Therefore, an efficient parallel version of black box multigrid demands an efficient parallel version of line relaxation. We also note that Brown et al. in [5] describe a similar parallel multigrid algorithm whose efficiency also depends on the existence of a fast line relaxation.

In the black box multigrid code, we consider discretizations of elliptic systems that yield nine-point stencils (in 2D) or 27-point stencils (in 3D) on structured grids. In 2D, a relaxation of a single line in either the x- or y-direction is set up by keeping all unknowns, except the ones along the line, at their current value, and solving for the unknowns along the line in a block fashion. Since we limit ourselves to nine point stencils in 2D, this block solve yields a tridiagonal system of linear equations. By introducing the well known two-color zebra coloring of all lines of a particular direction (see Figure 2.1), we ensure that lines residing on different groups of processors can be relaxed simultaneously. After all lines of one color are relaxed, we switch to the other color in the same direction, and then perform the same procedure for all lines in the other coordinate direction.

In 3D, we use a two-color zebra coloring of all xy-, yz-, and xz-planes, and perform a single 2D black box multigrid cycle for each plane of the same color in one direction, and then one cycle for each plane of the other color in the same direction. We do the same for the planes in the remaining two directions. Since in each plane relaxation step many line relaxation steps are performed, it is paramount for the parallel performance of the black box multigrid code that tridiagonal solves are performed efficiently in parallel.

3. Overview of parallel tridiagonal solvers. The Thomas algorithm is a very efficient algorithm for solving tridiagonal systems of equations in serial [15]. It is equivalent to Gaussian elimination without pivoting; therefore, it is inherently serial in the sense that that its communication has a complexity of $O(N_P)$. In the context of alternating line relaxation, one can alleviate this problem by relaxing lines in the alternate direction once a processor is idle [11]. This procedure is only efficient, though, when a large number of successive alternating line relaxations are performed. In the context of a multigrid method, where only one or two sweeps of alternating line relaxation are needed on each level, the Thomas algorithm is simply impractical in a parallel setting.

Cyclic reduction was first introduced in 1965 by Hockney and Golub [9]. If we number equations succesively, then in cyclic reduction the odd-neighbor equations of an even equation are used to eliminate the off-diagonal entries in the even equation. In this step, a tridiagonal system of linear equations for the even equation is generated. This reduced system has only about one-half as many equations as the original system. Now the same procedure is applied recursively to the reduced system until there remains only one linear equation with one unknown. The solution of successive reduced systems can be computed to finally yield the solution of the original system.

Cyclic reduction requires $2 * \log_2 N_P$ steps of nearest neighbor communication. Additionally, storage requirements can be held to a minimum by overwriting the original tridiagonal system with all reduced systems of equations. Since the line relaxations needed by the robust black box multigrid method require the same tridiagonal system of equations with different right-hand sides, one cannot overwrite the original system of equations. As a result, additional storage of one-half times the storage of the original tridiagonal system is needed in the context of line relaxation. An example of an implementation of parallel cyclic reduction is the parallel semicoarsening multigrid code described in [5].

Recursive doubling was introduced in 1973 by Stone [13].

Wang introduced a new partitioning algorithm in 1981 [17]. The basic idea is that a tridiagonal interface of N_P linear equations is generated without communication. Each processor owns one equation of this interface system. After solving the interface system of equations a back substitution step generates the solution. In the context of line relaxation, to generate an interface equation, storage for one local tridiagonal system and the right hand side is needed. The parallel solution of the small interface system can, for example, be done using parallel cyclic reduction. In this paper, we present a new partitioning algorithm that is more memory efficient, and based on it, a recursive version that has logarithmic communication complexity.

The divide and conquer algorithm introduced by Bondeli in 1991 [4] consists of inverting the tridiagonal systems that reside on each processor, by disregarding the connections to neighbor processors. Similar to partitioning algorithms, this also yields a interface system of equations that must be solved in parallel, for example, by cyclic reduction. The storage requirement for this divide an conquer algorithm is about twice the storage of a tridiagonal system.

In the context of line relaxation in a multigird method, it is desirable to minimize the number of messages that are sent between processors. Two color zebra relaxation is well suited for this purpose, since lines of the same color can be relaxed simultaneously, and therefore the number of messages is greatly reduced.

4. The memory efficient partitioning algorithm. We now describe our memory efficient algorithm for the parallel solution of tridiagonal systems of linear equations. We are interested in solving the linear system

$$\mathbf{A}\mathbf{x} = \mathbf{b}, \tag{4.1}$$

where $\mathbf{A} \in \Re^{N\times N}$ is a tridiagonal matrix. To simplify the presentation of our algorithm, we denote by $\mathbf{v}_i \in \Re^N$ the ith row vector of $\mathbf{A}$ such that

$$\mathbf{A} = \begin{pmatrix} \mathbf{v}_1^t \\ \vdots \\ \mathbf{v}_N^t \end{pmatrix}. \tag{4.2}$$

Note that for each $\mathbf{v}_i = (v_{i,1}, v_{i,2}, \ldots, v_{i,N})^t$ we have $v_{i,j} = 0$, for $j \neq i-1, i, i+1$. Our algorithm is designed to work for diagonally dominant tridiagonal matrices. Thus, we assume that

$$v_{i,i} > |v_{i,i-1}| + |v_{i,i+1}|, \quad i = 1, \ldots, N, \tag{4.3}$$

where we set $v_{1,0} = v_{N,N+1} = 0$. We assume that there are $N_P < N$ processors and that each processor has a unique index or rank. We denote by j_k the smallest row number such that row j_k is owned by processor k. We assume that $j_1 = 1$, $j_{N_P+1} = N+1$, and $j_k < j_{k+1}$ for $k = 1, \ldots, N_P$. Hence, processor k owns $N_k = j_{k+1} - j_k$ rows. Matrix $\mathbf{A}$ is assumed to be distributed over all processors such that processor k owns rows $j_k, \ldots, j_{k+1}-1$, for $k = 1, \ldots, N_P$.

The first phase. First, each processor computes a linear combination of its rows to generate the lower interface equation. We store this row in $\mathbf{v}_{l_k}$ and the corresponding right hand side in b_{l_k}. We perform the following steps simultaneously on all processors.

1. Set $\mathbf{v}_{l_k}^{(j_k+1)} \leftarrow \mathbf{v}_{j_k+1}$ and $b_{l_k}^{(j_k+1)} \leftarrow b_{j_k+1}$.

2. For $i = j_k + 2, \ldots, j_k + N_k - 1$,

$$\begin{aligned} \alpha_i &\leftarrow v_{i,i-1}/v_{l_k,i-1} \\ \mathbf{v}_{l_k}^{(i)} &\leftarrow \mathbf{v}_i - \alpha_i \mathbf{v}_{l_k}^{(i-1)} \\ b_{l_k}^{(i)} &\leftarrow b_i - \alpha_i b_{l_k}^{(i-1)}. \end{aligned}$$

Here the superscript denotes the iterate. Then each processor computes a linear combination of its rows to generate the upper interface equation. Analogous to the previous steps, the resulting row is stored in $\mathbf{v}_{u_k}$ with the corresponding right hand side being stored in b_{u_k}. The following steps are performed simultaneously on all processors.

1. Set $\mathbf{v}_{u_k}^{(j_k+N_k-2)} \leftarrow \mathbf{v}_{j_k+N_k-2}$ and $b_{u_k}^{(j_k+N_k-2)} \leftarrow b_{j_k+N_k-2}$.
2. For $i = j_k + N_k - 3, \ldots, 1$,

$$\begin{aligned} \beta_i &\leftarrow v_{i,i+1}/v_{u_k,i} \\ \mathbf{v}_{u_k}^{(i)} &\leftarrow v_i - \beta_i \mathbf{v}_{u_k}^{(i+1)} \\ b_{u_k}^{(i)} &\leftarrow b_i - \beta_i b_{u_k}^{(i+1)}. \end{aligned}$$

During the entire computation, both $\mathbf{v}_{l_k}^{(i)}$ and $\mathbf{v}_{u_k}^{(i)}$ have at most three non-zero entries. In iteration i, for the lower interface row $\mathbf{v}_{l_k}^{(i)}$, these are v_{l_k,j_k}, $v_{l_k,i}$, and $v_{l_k,i+1}$, while for the upper interface row $\mathbf{v}_{u_k}^{(i)}$, these are v_{u_k,j_k+N_k-1}, $v_{u_k,i}$, and $v_{u_k,i-1}$. In an implementation, we only need storage for these three values for both upper and lower interface row.

The second phase. The resulting interface system can be written as

$$\begin{pmatrix} \mathbf{v}_{u_1}^t \\ \mathbf{v}_{l_1}^t \\ \vdots \\ \mathbf{v}_{u_{N_P}}^t \\ \mathbf{v}_{l_{N_P}}^t \end{pmatrix} \begin{pmatrix} x_{j_1} \\ x_{j_2-1} \\ \vdots \\ x_{j_{N_P}} \\ x_{j_{N_P+1}-1} \end{pmatrix} = \begin{pmatrix} b_{u_1} \\ b_{l_1}^t \\ \vdots \\ b_{u_{N_P}}^t \\ b_{l_{N_P}}^t \end{pmatrix}. \tag{4.4}$$

Note that interface system (4.4) is tridiagonal, and that for a given k the rows with index u_k and l_k reside on processor k.

To solve the interface system, we can simply gather the entire system to one of the processors, where we solve it using the sequential Thomas algorithm. Then we send the solution back to the individual processors: Processor k receives x_{j_k-1}, x_{j_k}, $x_{j_{k+1}-1}$, and $x_{j_{k+1}}$, for $k = 2, \ldots, N_P - 1$. Processor 1 receives x_{j_1}, x_{j_2-1}, and x_{j_2}, and processor N_P receives $x_{j_{N_P}-1}$, $x_{j_{N_P}}$, and $x_{j_{N_P+1}-1}$.

The third phase. After substituting the received solution values simultaneously on each processor, we can immediately solve for unknowns x_{j_k+1} and $x_{j_{k+1}-1}$. What remains on each processor is a tridiagonal system of equations for unknowns $x_{j_k+2}, \ldots, x_{j_{k+1}-2}$ that can be solved using the sequential Thomas algorithm.

Our algorithm can be interpreted as a partitioning algorithm. After completion of phase one, row vector $\mathbf{v}_{l_k}$ is a linear combination of row vectors $\mathbf{v}_{j_k+1}, \ldots, \mathbf{v}_{j_{k+1}-1}$, and $\mathbf{v}_{u_k}$ is a linear combination of row vectors $\mathbf{v}_{j_k}, \ldots, \mathbf{v}_{j_{k+1}-2}$. If we replace row vector $\mathbf{v}_{j_k}$ and $\mathbf{v}_{j_{k+1}-1}$ with row vectors $\mathbf{v}_{u_k}$ and $\mathbf{v}_{l_k}$, respectively for $k = 1, \ldots, N_P$, we obtain the partitioned matrix $\tilde{\mathbf{A}}$. In Figure 4.1, we show the sparsity pattern of a partitioned matrix, with parameters $N = 15$, $N_P = 3$, and $N_k = 5, k = 1, 2, 3$.

$$\tilde{\mathbf{A}} = \left(\begin{array}{ccccccccccccccc}
\blacksquare & & & & \blacksquare & & & & & & & & & & \\
\times & \times & \times & & & & & & & & & & & & \\
& \times & \times & \times & & & & & & & & & & & \\
& & \times & \times & \times & & & & & & & & & & \\
\blacksquare & & & & \blacksquare & \blacksquare & & & & & & & & & \\
\hline
& & & & \blacksquare & \blacksquare & & & & \blacksquare & & & & & \\
& & & & & \times & \times & \times & & & & & & & \\
& & & & & & \times & \times & \times & & & & & & \\
& & & & & & & \times & \times & \times & & & & & \\
& & & & & \blacksquare & & & & \blacksquare & \blacksquare & & & & \\
\hline
& & & & & & & & & \blacksquare & \blacksquare & & & & \blacksquare \\
& & & & & & & & & & \times & \times & \times & & \\
& & & & & & & & & & & \times & \times & \times & \\
& & & & & & & & & & & & \times & \times & \times \\
& & & & & & & & & & \blacksquare & & & & \blacksquare
\end{array}\right)$$

FIG. 4.1. *Sparsity pattern of a partitioned matrix: non-zero entries marked with* $\times$ *are equal to the corresponding entries in the original tridiagonal matrix, non-zero entries marked with* $\blacksquare$ *are computed in phase one of the partitioning algorithm. Horizontal lines indicate processor boundaries.*

THEOREM 4.1. *If the matrix in (4.1) is tridiagonal and diagonally dominant with positive diagonal entries, then so is the interface system (4.4).*

Proof. The proof is by induction. We first prove that $\mathbf{v}_{l_k}^{(i)}$ has at most three non-zero entries in positions j_k, i, and $i+1$, for all $i = j_k+1, \ldots, j_k+N_k-1$. The statement is certainly true for $i = j_k+1$, since $\mathbf{v}_{l_k}^{(j_k+1)}$ equals row j_k+1 of matrix $\mathbf{A}$. We assume that $\mathbf{v}_{l_k}^{(i-1)}$ has at most three non-zero entries in positions j_k, $i-1$, and i. We know that $\mathbf{v}_i$ has at most three non-zero entries in positions $i-1$, i, and $i+1$. Since $\mathbf{v}_{l_k}^{(i)}$ is a linear combination of $\mathbf{v}_{l_k}^{(i-1)}$ and $\mathbf{v}_i$, and α_i is chosen such that

$$v_{l_k,i-1}^{(i)} = v_{i,i-1} - \alpha_i v_{l_k,i-1} = 0, \tag{4.5}$$

we conclude that $\mathbf{v}_{l_k}^{(i)}$ can have non-zero entries only in positions j_k, i, and $i+1$. As a result, the lower interface equation can have non-zero coefficients only in positions j_k, $j_{k+1}-1$, and j_{k+1}, for $k = 1, \ldots, N_P-1$. The lower interface equation on processor N_P, has at most two non-zero entries, since entry j_{N_P+1} does not exist.

Now we show that the diagonal element in the lower interface equation on processor k is positive and that the lower interface equation on processor k is diagonally dominant. Both statements are certainly true for $i = j_k+1$, since $\mathbf{v}_{l_k}^{(j_k+1)}$ equals row j_k+1 of matrix $\mathbf{A}$. Assuming that $v_{l_k,i-1}^{(i-1)} > |v_{l_k,j_k}^{(i-1)}| + |v_{l_k,i}^{(i-1)}|$, we must show that

$$v_{l_k,i}^{(i)} = v_{i,i} - \alpha_i v_{l_k,i}^{(i-1)} > 0. \tag{4.6}$$

Using diagonal dominance of $\mathbf{v}_{l_k}^{(i-1)}$ and $\mathbf{v}_i$, and $v_{l_k,i-1}^{(i-1)} > 0$, we show

$$\begin{aligned} v_{l_k,i-1}^{(i-1)} v_{i,i} - v_{i,i-1} v_{l_k,i}^{(i-1)} &> \left(|v_{l_k,j_k}^{(i-1)}| + |v_{l_k,i}^{(i-1)}|\right)(|v_{i,i-1}| + |v_{i,i+1}|) - v_{i,i-1} v_{l_k,i}^{(i-1)} \\ &\geq |v_{l_k,j_k}^{(i-1)}||v_{i,i-1}| + |v_{l_k,j_k}^{(i-1)}||v_{i,i+1}| + |v_{l_k,i}^{(i-1)}||v_{i,i+1}| \\ &\geq 0, \end{aligned}$$

which establishes (4.6). We now show diagonal dominance of $\mathbf{v}_{l_k}^{(i)}$, which is equivalent to

$$v_{l_k,i-1}^{(i-1)} v_{i,i} - v_{i,i-1} v_{l_k,i}^{(i-1)} > |v_{l_k,j_k}^{(i-1)} v_{i,i-1}| + |v_{i,i+1} v_{l_k,i-1}^{(i-1)}|. \tag{4.7}$$

Using diagonal dominance of $\mathbf{v}_i$ and $\mathbf{v}_{l_k}^{(i-1)}$, we show

$$\begin{aligned} & v_{i,i-1} v_{l_k,i}^{(i-1)} + |v_{l_k,j_k}^{(i-1)} v_{i,i-1}| + |v_{i,i+1} v_{l_k,i-1}^{(i-1)}| \\ \leq\; & |v_{i,i-1} v_{l_k,i}^{(i-1)}| + |v_{l_k,j_k}^{(i-1)} v_{i,i-1}| + |v_{i,i+1} v_{l_k,i-1}^{(i-1)}| \\ =\; & |v_{l_k,i}^{(i-1)}| \left(|v_{i,i-1}| + |v_{l_k,j_k}^{(i-1)}|\right) + |v_{i,i+1} v_{l_k,i-1}^{(i-1)}| \\ <\; & |v_{l_k,i}^{(i-1)} v_{l_k,i-1}^{(i-1)}| + |v_{i,i+1} v_{l_k,i-1}^{(i-1)}| \\ =\; & |v_{l_k,i-1}^{(i-1)}| \left(|v_{l_k,i}^{(i-1)}| + |v_{i,i+1}|\right) \\ <\; & v_{l_k,i-1}^{(i-1)} v_{i,i}, \end{aligned}$$

which establishes (4.7).

The proof for $\mathbf{v}_{u_k}$ is analogous. ☐

We remark that Theorem 4.1 implies that phase one of our algorithm is numerically stable, since at any time in the iterations $\mathbf{v}_{l_k}^{(i)}$ and $\mathbf{v}_{u_k}^{(i)}$ are diagonally dominant. Clearly, both phase two and three are stable, since the stable Thomas algorithm is used.

In phase one of the algorithm, only six variables are needed to compute the interface equations on each processor. Note that the original tridiagonal system is not modified in this step. In the context of two-color zebra line relaxation in a multigrid method, the tridiagonal system does not need to be strored at all since the 2D matrix can be used directly to generate the interface equations. In contrast to Wangs's partitioning algorithm, our algorithm does not require additional storage for a modified right hand side in phase one. In phase two, $4N_P$ variables are needed on one of the processors to store one interface system, in addition to storing the coefficients of the complete interface tridiagonal system. In phase three of our algorithm, we need storage for one tridiagonal system, and we use the original right hand side. In the context of two-color zebra line relaxation, storage for one tridiagonal system of equations is enough. Since in phase three of our algorithm the original tridiagonal system is used, its coefficients can be taken directly from the 2D matrix.

5. Examples. In Figure 5.1, we show timings for 20 $V(1,1)$ cycles of symmetric BoxMG with alternating line relaxation. Here the problem size on the finest grid is fixed on all processors at 1000×1000. All timings are obtained on square processor grids, from 2×2 through 22×22 processors. The discrete problem was Poisson's equation with the standard five-point stencil discretization. We observe a slight growth in the time per V-cycle in both the aggregate time as well as the time just for line

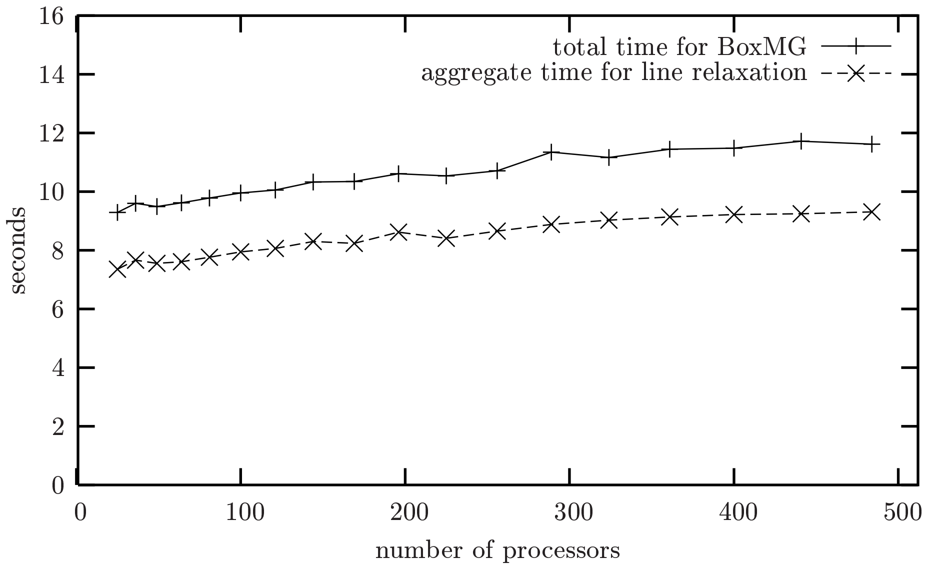

FIG. 5.1. *Timings for 20 $V(1,1)$ BoxMG cycles with red-black line relaxation on square processor grids ranging from 2×2 to 22×22.*

relaxation. Note that for the largest problem, lines span at most 22 processors. In the following, we consider lines that span significantly more processors.

Denote by $T(N, M, N_P)$ the time it takes to perform our line relaxation algorithm for a system of equations with N unknowns on N_P processors for M lines. The scaled efficiency is then defined as

$$E(N, M, P) \equiv T(N, M, 1)/T(N_P N, M, N_P) \tag{5.1}$$

Figure 5.2 shows the scaled parallel efficiency of our line relaxation implementation for $N = 1000 * N_P$ and $M = 500$. We see a slow degradation of scaled parallel efficiency to about 30% for $N_P = 250$. Note that the interface problem is a tridiagonal system with $2 * N_P$ unknowns and that M of such interface problems must be solved. Since we gather all these equations on one of the processors and solve this interface system using the Thomas algorithm, our algorithm has a linear dependence on N_P. However, we propose a recursive algorithm that is based on our partitioning algorithm: this algorithm reduces the dependency to a logarithmic dependence on P.

Figure 5.3 shows detailed timings for the different phases of our algorithm. As expected, the calculation of the right hand side, phase 1, and phase 3 scale perfectly because no communication occurs. The complexity of phase 2 is linear in N_P.

In Figure 5.4, we show detailed timings for all parts of phase two of our algorithm. As expected, the complexity of each part of phase two appears to depend linearly on N_P. The scatter operation has the strongest dependence on N_P.

In the next section, we propose a recursive version of our partitioning algorithm. This recursive partitioning algorithm will exhibit only logarithmic dependence on N_P.

6. The recursive partitioning algorithm. For very large number of processors, the solution time for our parallel tridiagonal solver is dominated by the two communication steps in phase 2: scatter and gather. Our numerical experiments

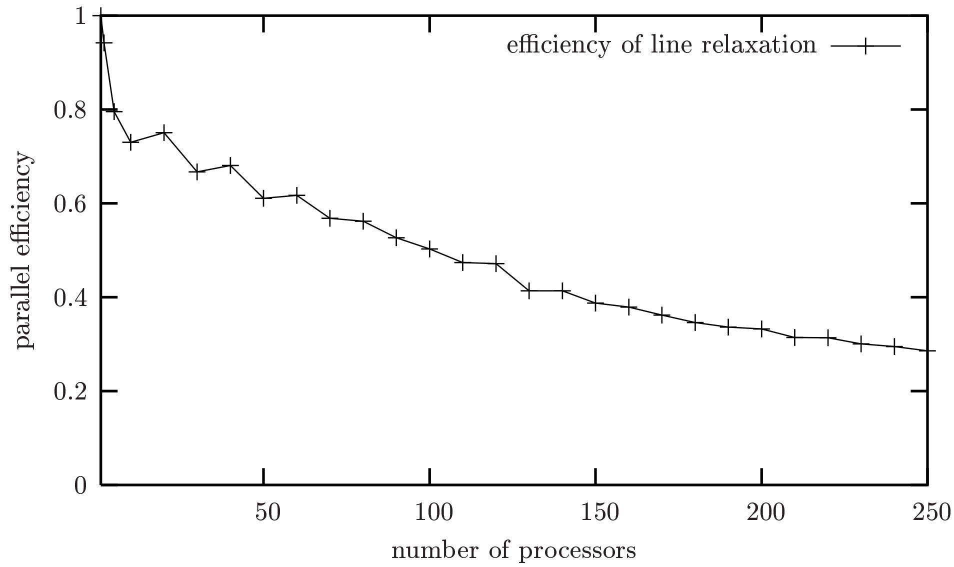

FIG. 5.2. *Scaled linear efficiency on of the line relaxation for $N = 1000N_P$ and $M = 500$*

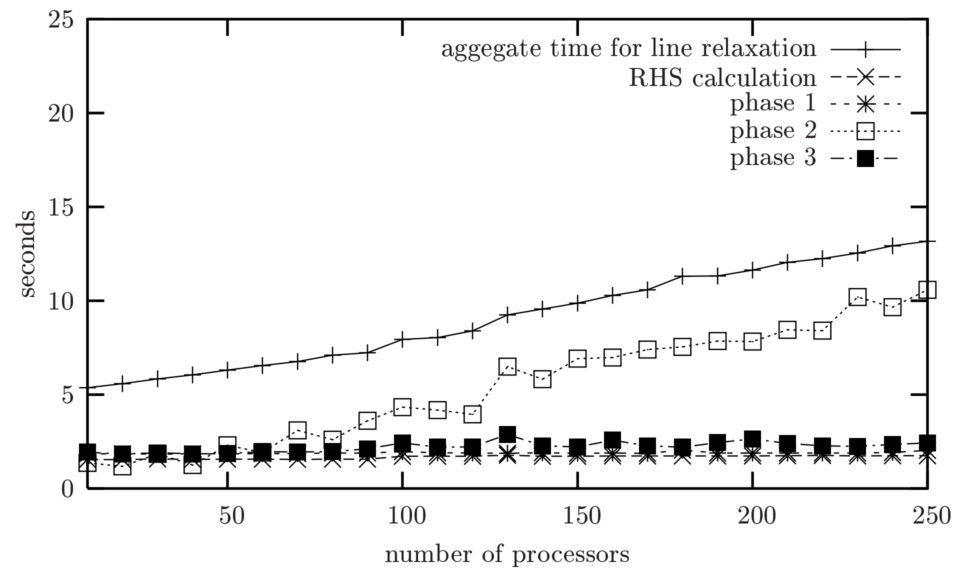

FIG. 5.3. *Detailed timings for all parts of our partitioning algorithm for $N = 1000N_P$ and $M = 500$.*

show that the scatter operation has a complexity of $O(N_P)$, while the gather operation has a slightly better complexity.

To alleviate this problem, we propose a recursive version of our partitioning algorithm. Since the interface system of equations is tridiagonal, diagonally dominant with positive diagonal entries, we solve using our partitioning algorithm. We do this by combining groups of interface equations on one processor. To be specific, we se-

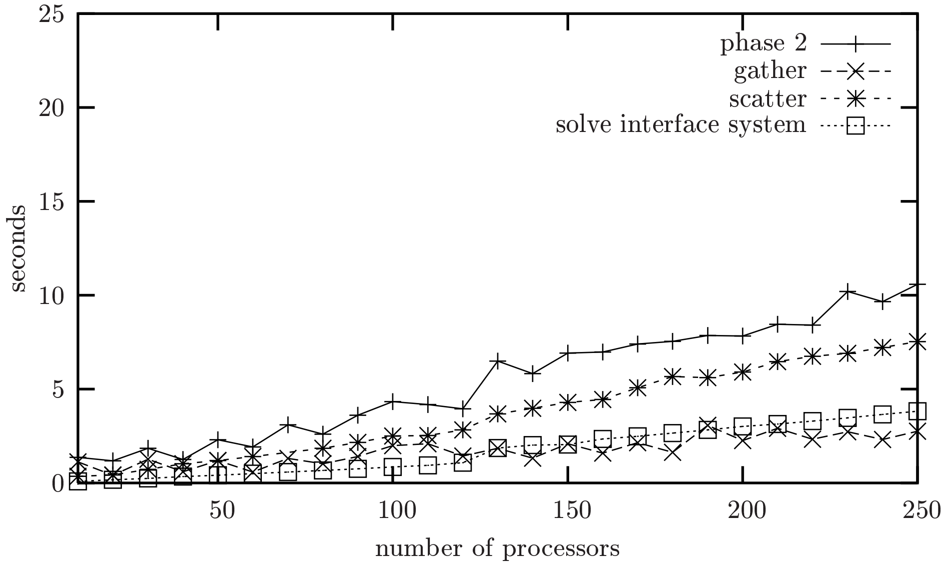

FIG. 5.4. *Detailed timings for all parts of phase two for $N = 1000N_P$ and $M = 500$.*

lect M_P groups of subsequent processors in such a way that there are roughly equal numbers of processors in each group. This can, for example, be accomplished by defining

$$\begin{aligned} M_k &\equiv \lfloor N_P/M_P \rfloor, \quad k = 1, \ldots, N_P - 1 \\ M_{M_P} &\equiv N_P - (M_P - 1)\lfloor N_P/M_P \rfloor, \end{aligned}$$

and assigning processors $\sum_{i=1}^{k-1} M_i+1$ through $\sum_{i=1}^{k} M_k$ to group k, for $k = 1, \ldots, M_P$.

After each processor has generated its two interface equations (phase one of our algorithm), these interface equations are gathered to the processor of lowest rank in the group. We call this processor the base processor in its group. This base processor now owns the part of the interface tridiagonal linear system that was generated in its group.

If the number of groups is larger than M_k, we can proceed recursively by dividing the set of all base processors in several groups of roughly equal size M_k. Then each of the base processors generates two interface equations from its part of the interface tridiagonal linear system, and so forth

After solving the interface system, the solution must be sent from the processor with lowest rank to all other processors in each group. This can be accomplished by using a scatter operation that involves all processors in each group. Since each processor is a member of only one group, all groups can perform these scatter operations simultaneously. Now we can proceed with phase three of our partitioning algorithm.

Table 6.1 illustrates this coarsening procedure using 18 processors and a group size of at most three. Suppose that we are given a tridiagonal linear system of equations that is distributed across 18 processors (row A). Each processor runs through phase one of the non-recursive algorithm to generate two interface equations. Groups of three successive processors (row B) gather these interface equations to the lowest rank processor in their group (row C). In this example, now each of the lowest

A	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
B	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
C	1			4			7			10			13			16		
D	1			4			7			10			13			16		
E	1									10								
F	1									10								
G	1																	

TABLE 6.1
Illustration of the coarsening procedure for 18 processors and a group size of three.

rank processors owns six equations of the first level interface tridiagonal system of equations. Processors 1, 4, 7, 10, 13, and 16 then each generate two new interface equations using their part of the first level interface tridiagonal system. Now the lowest rank processors are grouped in successive groups of three (row D) and gather the two interface equations that were just generated to the lowest rank processor in each group (row E) to form the second level interface tridiagonal system. Processors 1 and 10 make up just one group (row F) and each own a part of this second level interface tridiagonal system. Each of these two processors proceed to generate two interface equations from it, which are then gathered to processor 1, the lowest rank processor in this group, to form the third level interface tridiagonal system. This small linear system is solved directly (row F).

The solution to the third level interface problem is now scattered to all processors in the group (row F) and used to compute the solution to the second level interface problem. The solution of the second level interface problem can now be computed (row E) and scattered to all processors in the level two groups (row D). This solution to the second level interface system is now used to compute the solution to the first level interface problem (row C), which is then scattered to all processors in the level one groups (row B). Now each processor knows the solution to the first level interface problem and can solve its part of the original problem.

Since communication is orders of magnitude slower than computation, it is desirable to create groups with enough processors to balance communication and computation. The group size, for which the best performance is achieved, depends on the target computing architecture and must be determined experimentally for a given parallel computer. In the next section, we present a complexity analysis that can aid in determining the optimal group size.

7. Complexity analysis. Our numerical experiments indicate that the time for a scatter operation grows linearly with the number of participating processors. The gather operation also appears to scale linearly, albeit with a smaller slope. All three phases of our algorithm, without taking into account the communication, also scale linearly. Denote by L the number of lines that must be solved on a given set of processors to complete a sweep of line relaxation (e.g., the black lines) Denote by $N_{k_{max}} = \max_{k=1,\dots,N_p} N_k$ the maximum number of equations that a single processor owns. Also, denote by γ, σ, and ρ_i, $i = 1, 2, 3$ the scaling factors of the gather operation, the scatter operation, and phase i, $i = 1, 2, 3$ of our algorithm, respectively. Then a simple model for the time it takes to complete our partitioning algorithm without recursion is

$$T_1 = (\rho_1 + \rho_3)LN_{k_{max}} + (\gamma + \rho_2 + \sigma)LN_p, \tag{7.1}$$

If we use one level of recursion, where we have M_P groups and the maximum number of processors in a single group is $M_{k_{max}}$, then the complexity of the algorithm is

$$T_2 = (\rho_1 + \rho_3)LN_{k_{max}} + (\gamma + \rho_1 + \rho_3 + \sigma)LM_{k_{max}} + (\gamma + \rho_2 + \sigma)LM_P \tag{7.2}$$

If we use multiple levels of recursion, where we have $M_P^{(\ell)}$ groups of processors on level ℓ and the maximum number of processors in a single group on level ℓ is $M_{k_{max}}^{\ell}$, then the complexity of the algorithm is

$$T_\ell = (\rho_1 + \rho_3)LN_{k_{max}} + (\gamma + \rho_1 + \rho_3 + \sigma)\sum_{i=2}^{\ell} LM_{k_{max}}^{(\ell)} + (\gamma + \rho_2 + \sigma)LM_P^{(\ell)} \tag{7.3}$$

Now assume a constant coarsening ratio, such that $M_{k_{max}}^{(\ell)}$ is the same for all levels and the number of processors in each group is the same for all groups on all levels. We are interested in the case where the algorithm uses as many levels of recursion as possible, that number is $\ell_{max} = \log_{M_{k_{max}}} N_P$. Then (7.3) becomes

$$T_{\ell_{max}} = (\rho_1+\rho_3)LN_{k_{max}} + (\gamma+\rho_1+\rho_3+\sigma)(\ell_{max}-1)LM_{k_{max}} + (\gamma+\rho_2+\sigma)LM_P^{(\ell_{max})}. \tag{7.4}$$

Equation (7.4) suggests that the recursive algorithm should scale logarithmically with the number of processors. Note that $M_P^{(\ell_{max})} \leq M_{k_{max}}$ is small and not dependent on N_P. Note that the constants in (7.4) must be measured experimentally to determine the coarsening strategy for a given parallel computer.

In the context of line relaxation in a multilevel scheme, the length of lines is halved on each coarser level. This will only affect the first term in (7.4), since $N_{k_{max}}$ will be smaller on coarser multigrid levels. The other two terms in (7.4) do not depend on the length of a line.

8. Conclusions. We have introduced a new memory efficient partitioning algorithm for the solution of tridiagonal linear systems of equations. Based on this algorithm we proposed a recursive version of this algorithm that we expect to exhibit only logarithmic complexity with respect to the number of processors. This algorithm is different from other tridiagonal solvers with logarithmic complexity with respect to the number of processors, in that it can be tuned for a given parallel computer. We will explore this recursive algorithm in a forthcoming paper.

REFERENCES

[1] P. Amodio and L. Brugano, *Parallel factorizations and parallel solvers for tridiagonal linear-systems*, Linear Algebra and its Applications, 172 (1992), pp. 347 – 364.
[2] P. Amodio, L. Brugano, and T. Politi, *Parallel factorizations for tridiagonal matrices*, SIAM J. Numer. Anal., 30 (1993), pp. 813 – 823.
[3] T. M. Austin, M. Berndt, B. K. Bergen, J. E. Dendy, and J. D. Moulton, *Parallel, scalable, and robust multigrid on structured grids*, tech. report, Los Alamos National Laboratory, LA-UR-03-9167, 2003.
[4] S. Bondeli, *Divide and conquer: a parallel algorithm for the solution of a tridiagonal linear system of equations*, Parallel Computing, 17 (1991), pp. 419–434.
[5] P. N. Brown, R. D. Falgout, and J. E. Jones, *Semicoarsening multigrid on distributed memory machines*, SIAM J. Sci. Stat. Comput., 21 (2000), pp. 1823–1834.
[6] J. E. Dendy, *Black-box multigrid*, Journal of Computational Physics, 48 (1982), pp. 366 – 386.
[7] ———, *Black-box multigrid for nonsymmetric problems*, Applied Mathematics and Computation, 13 (1983), pp. 261 – 283.

[8] J. E. Dendy, *Two multigrid methods for three-dimensional problems with discontinuous and anisotropic coefficients.*, SIAM J. Sci. Stat. Comput., 8 (1987), pp. 673–685.

[9] R. W. Hockney, *A fast direct solution of poissons equation using fourier analysis*, J. ACM, 12 (1965), pp. 95–113.

[10] J. Hofhaus and E. F. V. de Velde, *Alternating-direction line-relaxation methods on multicomputers*, SIAM J. Sci. Comput., 17 (1996), pp. 454–478.

[11] A. Povitzky, *Parallelization of pipelined algorithms for sets of linear banded systems*, J. Par. Distr. Comput., 59 (1999), pp. 68–97.

[12] S. Schaffer, *A semi-coarsening multigrid method for elliptic partial differential equations with highly discontinuous and anisotropic coefficients*, SIAM J. Sci. Stat. Comput., 20 (1998), pp. 228–242.

[13] H. S. Stone, *An efficient parallel algorithm for the solution of a tridiagonal linear system of equations*, J. ACM, 20 (1973), pp. 27–38.

[14] X.-H. Sun, H. Z. Sun, and L. M. Ni, *Parallel algorithms for solution of tridiagonal systems on multicomputers*, in Proceedings of the 3rd international conference on Supercomputing, ACM Press New York, NY, USA, 1986, pp. 303–312.

[15] L. H. Thomas, *Elliptic problems in linear difference equations over a network*, Watson Sci. Comput. Lab. Rept., Columbia University, New York, (1949).

[16] C. H. Walshaw, *Diagonal dominance in the parallel partition method for tridiagonal systems*, SIAM J. Matrix Anal. Appl., 16 (1995), pp. 1086–1099.

[17] H. H. Wang, *A parallel method for tridiagonal equations*, ACM Trans. Math. Software, 7 (1981), pp. 170–183.

[18] P. Yalamov and V. Pavlov, *On the stability of a partitioning algorithm for tridiagonal systems.*, SIAM Journal on Matrix Analysis and Applications, 20 (1998), pp. 159 – 81.