ARGONNE NATIONAL LABORATORY
9700 South Cass Avenue
Argonne, Illinois 60439

THE EFFICIENT PARALLEL ITERATIVE SOLUTION OF LARGE SPARSE LINEAR SYSTEMS*

Mark T. Jones and Paul E. Plassmann

Mathematics and Computer Science Division
Preprint MCS-P314-0692
June 1992

ABSTRACT

The development of efficient, general-purpose software for the iterative solution of sparse linear systems on a parallel MIMD computer requires an interesting combination of expertise. Parallel graph heuristics, convergence analysis, and basic linear algebra implementation issues must all be considered.

In this paper, we discuss how we have incorporated recent results in these areas into a general-purpose iterative solver. First, we consider two recently developed parallel graph coloring heuristics. We show how the method proposed by Luby, based on determining maximal independent sets, can be modified to run in an asynchronous manner and give an expected running time bound for this modified heuristic. In addition, a number of graph reduction heuristics are described that are used in our implementation to improve the individual processor performance. The effect of these various graph reductions on the solution of sparse triangular systems is categorized. Finally, we discuss the performance of this solver from the perspective of two large-scale applications: a piezoelectric crystal finite-element modeling problem, and a nonlinear optimization problem to determine the minimum energy configuration of a three-dimensional, layered superconductor model.

Key words: graph coloring heuristics, iterative methods, parallel algorithms, preconditioned conjugate gradients, sparse matrices

AMS(MOS) subject classifications: 65F10, 65F50, 65Y05, 68R10

*This paper is based on a talk presented by the second author at the IMA Workshop on Sparse Matrix Computations: Graph Theory Issues and Algorithms, October 14–18, 1991. This work was supported by the Applied Mathematical Sciences subprogram of the Office of Energy Research, U.S. Department of Energy, under Contract W-31-109-Eng-38.

1. Introduction. The computational kernel of many large-scale applications is the solution of sparse linear systems. Given the increasing performance of individual processors and the dramatic recent improvements in engineering parallel machines composed of these processors, a scalable parallel computer is an attractive vehicle for solving these problems. In this paper we endorse a particular perspective: (1) we note that in many applications one is interested in solving as large a problem as can feasibly fit into the available memory of the machine, and (2) that the underlying geometric structure of these applications is often three-dimensional or greater. These observations, and a simple "back-of-the-envelope" calculation, lead one to conclude that a parallel direct factorization method is in general not feasible for such problems, in terms of the amount of space and time required. This perspective motivates one to consider an approach to the iterative solution of sparse linear systems in a manner that ensures scalable performance.

In this paper we present an approach to solving such systems that satisfies the above requirements. Central to our method is a reordering of the matrix based on a coloring of the symmetric graph corresponding to the nonzero structure of the matrix, or a related graph. To determine this ordering, we use a recently developed parallel heuristic. However, if many colors are used, a straightforward parallel implementation, as is described in [10], suffers poor processor performance on a high-performance processor such as the Intel i860. In this paper we present several possible graph reductions that can be employed to greatly improve the performance of an implementation on high-performance RISC processors.

Consider an implementation of any of the standard general-purpose iterative methods [7, 15]: consistently ordered SOR, SSOR accelerated by conjugate gradients (CG), or CG preconditioned with an incomplete matrix factorization. It is evident that the major obstacle to a scalable implementation [6] is the inversion of sparse triangular systems with a structure based on the structure of the linear system. For example, the parallelism inherent in computing and applying an incomplete Cholesky preconditioner is limited by the solution of the triangular systems generated by the incomplete Cholesky factors [20]. It was noted by Schreiber and Tang [19] that if the nonzero structure of the triangular factors is identical to that of the original matrix, the minimum number of major parallel steps possible in the solution of the triangular system is given by the chromatic number of the symmetric adjacency graph representing those nonzeros. Thus, given the nonzero structure of a matrix A, one can generate greater parallelism by computing a permutation matrix, P, based on a coloring of the symmetric graph $G(A)$. The incomplete Cholesky factor $\tilde{L}$ of the permuted matrix PAP^T is computed, instead of the factor based on the original matrix A.

In this permutation, vertices of the same color are grouped and ordered sequentially. As a consequence, during the triangular system solves, the unknowns corresponding to these vertices can be solved for in parallel, after the updates from previous color groups have been performed. The result of Schreiber and Tang states that the minimum number of inherently sequential computational steps required

to solve either of the triangular systems, $\tilde{L}y = b$ or $\tilde{L}^T x = y$, is given by the minimum possible number of colors, or chromatic number, of the graph.

For general graphs, the computation of the chromatic number is an NP-hard problem. However, recent theoretical and experimental work has demonstrated scalable heuristics for determining coloring that are close to optimal for practical problems [14, 13]. In §2 we review these heuristics and demonstrate that they have equivalent expected running times for bounded degree graphs.

We note that this bound on the number of communication steps assumes that only vector operations are performed during the triangular systems solves. This assumption is equivalent to restricting oneself to a fine-grained parallel computational model, where we assign each unknown to a different processor. When many unknowns are assigned to a single processor, it is possible to reduce the number of communication steps by solving non-diagonal submatrices of L on individual processors at each step. In this case, the minimum number of communication steps is given by a coloring of a quotient graph obtained from a partitioning of unknowns to processors.

The remainder of the paper is organized as follows. In §3 we present several possible graph reductions, including the clique partitions that allow for the use of higher-level Basic Linear Algebra Subprograms (BLAS) in the software. We consider a general framework that can incorporate these ideas into efficient triangular system solvers in §4. Finally, in §5 we present experimental results obtained for our software implementation on the Intel DELTA for problems arising in two different applications and in §6 we discuss our conclusions.

2. Asynchronous Parallel Graph Coloring Heuristics. In this section we consider two recently developed graph coloring heuristics suitable for asynchronous parallel computers. Our perspective is that if a scalable iterative solver is to be based on a matrix ordering derived from a graph coloring, then a scalable heuristic is necessary to determine this coloring. We review two parallel heuristics based on Monte Carlo steps for which expected running times are known: a synchronous PRAM heuristic developed by Luby [14], and a recent asynchronous heuristic presented by Jones and Plassmann [13]. The interesting aspect of the asynchronous method is that it combines aspects of sequential greedy graph coloring heuristics with a Monte Carlo step to determine independent sets. Finally, we show how a modification can be made to Luby's maximal independent set heuristic both to make it asynchronous and to satisfy the same running time bound obtained for the second heuristic.

It is important to note that we do not address the problem of determining a good partitioning of the graph onto the processors. For the applications problems we consider in §5, a physical partition can be used to generate a good vertex assignment to processors. When the determination of a partition is not straightforward, a partitioning heuristic would have to be used. Some possibilities exist; for example, recent advances in the automatic partitioning of three-dimensional domains [21] or in spectral dissection methods [17] could be employed. However,

the parallel graph partitioning problem deserves much additional research.

First, we briefly review the graph coloring problem. Let $G = (V, E)$ be a symmetric graph with vertex set V, with $|V| = n$, and edge set E. We say that the function $\sigma : V \rightarrow \{1, \ldots, s\}$ is an s-coloring of G, if $\sigma(v) \neq \sigma(w)$ for all edges $(v, w) \in E$. We denote the minimum possible value for s, the chromatic number of G, by $\chi(G)$.

The question as to whether a general graph G is s-colorable is NP-complete [5]. It is known that unless $P = NP$, there does not exist a polynomial approximation scheme for solving the graph coloring problem [5]. In fact, the best polynomial time heuristic known [8] can theoretically guarantee a coloring of only size $c\,(n/\log n)\,\chi(G)$, where c is some constant.

Given these pessimistic theoretical results, it is quite surprising that, for certain classes of graphs, there exist a number of sequential graph coloring heuristics that are very effective in practice. For graphs arising from a number of applications, it has been demonstrated that these heuristics are often able to find colorings that are within one or two of an optimal coloring [4, 10].

These sequential heuristics are based a greedy heuristic that colors vertices in an order determined by a cost function. Choices for the cost function that are particularly effective are the saturation degree order (choose the most constrained vertex [3]) or the incidence degree order (choose the vertex adjacent to the maximum number of previously colored vertices [4]). Unfortunately, these heuristics do not parallelize well, because they essentially represent a breadth-first search of the graph.

A different approach was suggested by Luby [14]. His observation was that if one can determine a maximal independent set efficiently in parallel, then a partition of the vertices of the graph into maximal independent sets yields a coloring. Luby's algorithm for determining an independent set, I, is based on the following Monte Carlo rule. Here we denote the set of vertices adjacent to vertex v by $adj(v)$.

1. For each vertex $v \in V^{'}$ determine a distinct, random number $\rho(v)$.
2. $v \in I \Leftrightarrow \rho(v) > \rho(w)$, $\forall\, w \in adj(v)$.

In the Monte Carlo algorithm described by Luby [14], this initial independent set is augmented to obtain a maximal independent set. The approach is the following. After the initial independent set is found, the set of vertices adjacent to a vertex in I, the neighbor set $N(I)$, is determined. The union of these two sets is deleted from $V^{'}$, the subgraph induced by this smaller set is constructed, and the Monte Carlo step is used to choose an augmenting independent set. This process is repeated until the candidate vertex set is empty and a maximal independent set (MIS) is obtained. The complete Monte Carlo algorithm suggested by Luby for generating an MIS is shown in Fig. 1. In this figure we denote by $G(V^{'})$ the subgraph of G induced by the vertex set $V^{'}$. Luby shows that an upper bound for the expected time to compute an MIS by this algorithm on a CRCW P-RAM is $EO(\log(n))$. The algorithm can be adapted to a graph coloring heuristic by using it to determine a sequence of distinct maximal independent sets and by coloring each MIS a different

color. Thus, this approach will solve the $(\Delta + 1)$ vertex coloring problem, where Δ is the maximum degree of G, in expected time $EO((\Delta + 1)\log(n))$.

$I \leftarrow \emptyset$;
$V' \leftarrow V$;
$G' \leftarrow G$;
While $G' \neq \emptyset$ **do**
    Choose an independent set I' in G';
    $I \leftarrow I \cup I'$;
    $X \leftarrow I' \cup N(I')$;
    $V' \leftarrow V' \setminus X$;
    $G' \leftarrow G(V')$;
enddo

FIG. 1. *Luby's Monte Carlo algorithm for determining a maximal independent set*

A major deficiency of this approach on currently available parallel computers is that each new choice of random numbers in the MIS algorithm requires a global synchronization of the processors. A second problem is that each new choice of random numbers incurs a great deal of computational overhead, because the data structures associated with the random numbers must be recomputed. The asynchronous heuristic proposed by Jones and Plassmann [13] avoids both of these drawbacks. This heuristic is presented in Fig. 2. The heuristic is written assuming that each vertex v is assigned to a different processor and the processors communicate by passing messages.

With the asynchronous heuristic the first drawback (global synchronization) is eliminated by choosing the independent random numbers only at the start of the heuristic. With this modification, the interprocessor communication can proceed asynchronously once these numbers are determined. The second drawback (computational overhead) is alleviated because with this heuristic, once a processor knows the values of the random numbers of the vertices to which it is adjacent, the number of messages it needs to wait for can be computed and stored. Likewise, each processor computes only once the processors to which it needs to send a message once its vertex is colored. Finally, note that this heuristic has more of the "flavor" of the sequential heuristic, since we choose the smallest color consistent with the adjacent vertices previously colored.

An upper bound for the expected running time of a synchronous version of this algorithm of $EO(\log(n)/\log\log(n))$ can be obtained for graphs of bounded degree [13]. The central idea for the proof of this bound is the observation that the running time of the heuristic is proportional to the maximum length *monotonic path* in G. A *monotonic path* of length t is defined to be a path of t vertices $\{v_1, v_2, \ldots, v_t\}$ in G such that $\rho(v_1) > \rho(v_2) > \ldots > \rho(v_t)$.

Choose $\rho(v)$;
$n\text{-}wait = 0$;
$send\text{-}queue = \emptyset$;
For each $w \in adj(v)$ **do**
    Send $\rho(v)$ to processor responsible for w;
    Receive $\rho(w)$;
    if $(\rho(w) > \rho(v))$ **then** $n\text{-}wait = n\text{-}wait + 1$;
    else $send\text{-}queue \leftarrow send\text{-}queue \cup \{w\}$;
enddo
$n\text{-}recv = 0$;
While $(n\text{-}recv < n\text{-}wait)$ **do**
    Receive $\sigma(w)$;
    $n\text{-}recv = n\text{-}recv + 1$;
enddo
$\sigma(v)$ = smallest available color consistent with the
    previously colored neighbors of v;
For each $w \in send\text{-}queue$ **do**
    Send $\sigma(v)$ to processor responsible for w;
enddo

FIG. 2. *An asynchronous parallel coloring heuristic*

We now show that the Luby's MIS algorithm can be modified to obtain the same bound. Consider the following modification to the asynchronous coloring heuristic given in Fig. 2. Let the function $\gamma(v)$ equal one if v is in the independent set I, two if v is in $N(I)$, and let it be undefined otherwise. We have the following asynchronous algorithm to determine the set I.

The following lemma proves the correctness of the asynchronous algorithm.

LEMMA 2.1. *At the termination of the algorithm given in Fig. 3, the function $\gamma(v), v \in V$ defines a maximal independent set.*

Proof: At the completion of the algorithm in Fig. 3, $\gamma(v)$ is defined for each $v \in V$. Thus, each vertex $v \in V$ satisfied one of the following based on the definition of γ:

1. $v \in I$, or
2. $v \in N(I)$.

It is clear that the set I is independent, and each member of $N(I)$ must be adjacent to a member of I. Thus, the above two conditions imply that the independent set I is maximal. □

Based on Theorem 3.3 and Corollary 3.5 given in [13], we have the following corollary.

COROLLARY 2.2. *For graphs of bounded degree Δ, the expected running time is*

```
Choose ρ(v);
n-wait = 0;
send-queue = ∅;
For each w ∈ adj(v) do
    Send ρ(v) to processor responsible for w;
    Receive ρ(w);
    if (ρ(w) > ρ(v)) then n-wait = n-wait + 1;
    else send-queue ← send-queue ∪ {w};
enddo
n-recv = 0;
While (n-recv < n-wait) do
    Receive γ(w);
    n-recv = n-recv + 1;
enddo
if (all the previously assigned neighbors w of v
    have γ(w) = 2), then γ(v) = 1;
else γ(v) = 2;
endif
For each w ∈ send-queue do
    Send γ(v) to processor responsible for w;
enddo
```

FIG. 3. *An asynchronous algorithm to determine a maximal independent set*

$EO(\log(n)/\log\log(n))$ *for the maximal independent set algorithm given in Fig. 3.*

Proof: As for the bound for the asynchronous parallel coloring heuristic, the expected running time for the asynchronous maximal independent set algorithm is proportional to the expected length of the longest monotonic path. By Theorem 3.3 and Corollary 3.5 in [13] this length is bounded by $EO(\log(n)/\log\log(n))$. □

Finally, we note that this maximal independent set algorithm can be used in place of Luby's MSI algorithm to generate a sequence of maximal independent sets, each of which can be colored a different color. The running time of this coloring heuristic would again be bounded by $EO(\log(n)/\log\log(n))$ because the maximum number of colors used is bounded by $\Delta + 1$, and we have assumed the maximum degree Δ of the graph is bounded.

3. Graph Reductions. In this section we present several graph reductions that are used in our iterative solver implementation. These reductions are employed in §4 to describe several possible alternatives for the solution of the triangular systems involving the preconditioned systems.

It is often observed that the sparse systems arising in many applications have a great deal of special local structure, even if the systems are described as "un-

structured." We have attempted to illustrate some of this local structure, and how it can be identified, in the following sequence of figures.

In Fig. 4 we depict a subsection of a graph that would arise from a two-dimensional, linear, multicomponent finite-element model with three degrees of freedom per node point. We illustrate the three degrees of freedom by the three dots at each node point; the linear elements imply that the twelve degrees of freedom sharing the four node points of each face are completely connected. In the figure we show edges only between the nodes, these edges represent the complete interconnection of all the vertices on each element or face.

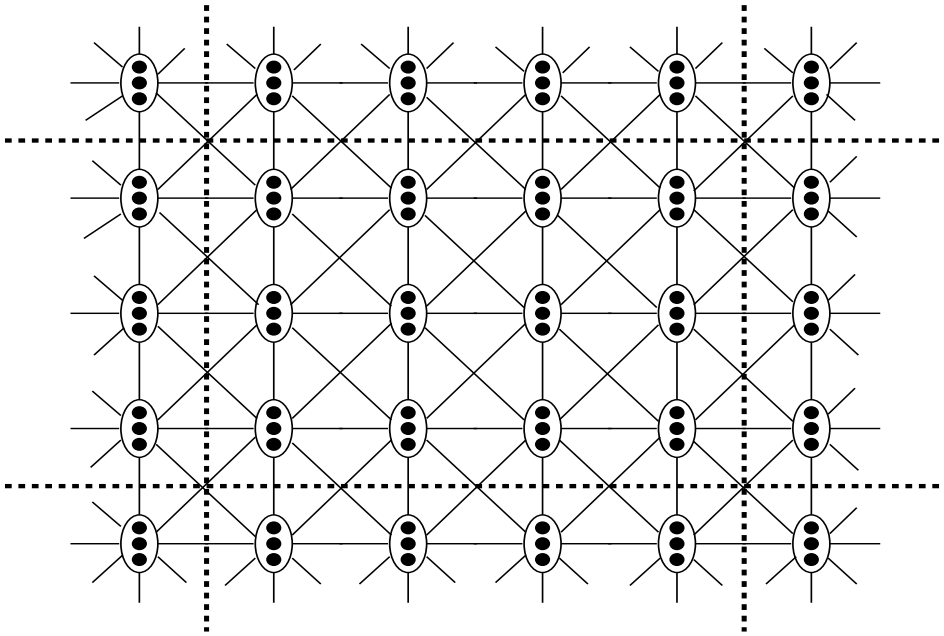

FIG. 4. *A subgraph generated by a two-dimensional, linear finite element model with three degrees of freedom per node point. The geometric partition shown by the dotted lines yields an assignment of the vertices in the enclosed subregion to one processor.*

The dashed lines in the figure represent a geometric partitioning of the grid; we assume that the vertices in the central region are all assigned to one processor. We make several observations about the local structure of this subgraph. First, we note that the adjacency structure of the vertices at the same geometric node (i.e., the nonzero structure of the associated variables) are identical, and we call such vertices *identical vertices*. It was noted by Schreiber and Tang [19] that a coloring of the graph corresponding to the geometric nodes results in a system with small dense blocks, of order the number of degrees of freedom per node, along the diagonal. We note that this observation can also be used to decrease the storage required for indirect indexing of the matrix rows since the structures are identical.

We also consider another graph reduction based on the local clique structure of the graph. In Fig. 5 the dotted lines show one possible way the vertices assigned to the shown partition and its neighbors can be partitioned into cliques. Denote such a partition by Q. If we associate a super vertex with each clique, the quotient

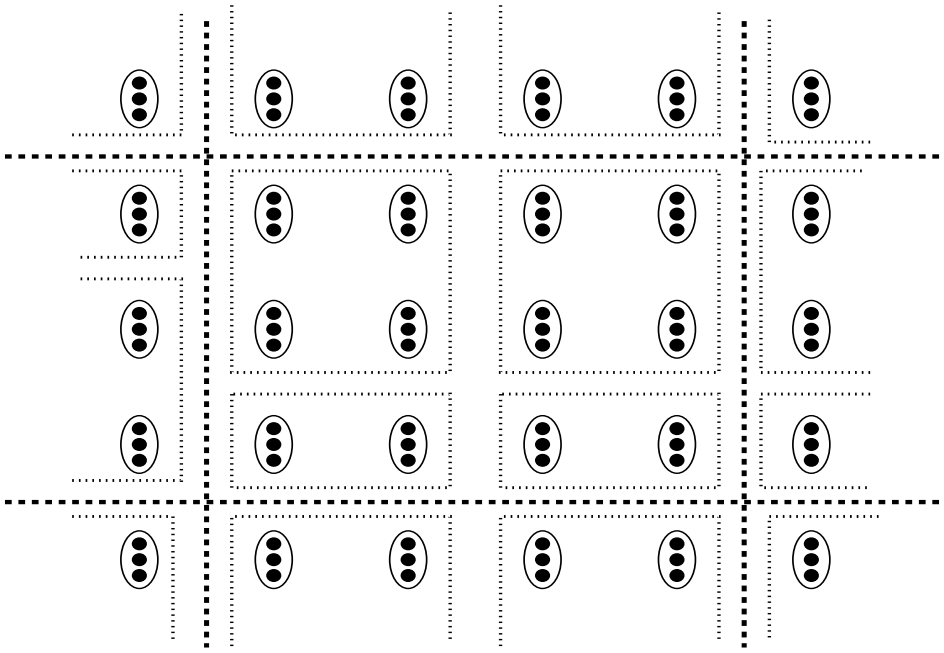

FIG. 5. *A partition of the vertices into cliques*

graph G/Q can be constructed based on the rule that there exists an edge between two super vertices v and w if and only if there exists an edge between two vertices of their respective partitions in G. The quotient graph constructed by the clique partition shown in Fig. 5 is shown in Fig. 6.

Of course the quotient graph reduction is not limited to the choice of a maximal clique partition; any local partition of the subgraph assigned to a processor can be used to generate the reduced graph. We use a clique decomposition because the submatrix associated with the clique is dense, thus allowing for the use of higher level dense linear algebra operations (BLAS) in an implementation. The aspect of the graph reduction is discussed in more detail in §4. Finally, we note that the efficient determination of identical nodes, and a local maximal clique decomposition, is straightforward. Since the adjacency structure of the vertices assigned to a processor is known locally, no interprocessor communication is required, and a greedy heuristic can be used to determine a clique partition.

4. The Inversion of Triangular Systems. In this section we review the problem of the parallel inversion of a sparse triangular system. The triangular system solution is the central problem in the parallelization of the standard iterative methods. For example, it is involved in the application of a preconditioner derived from an incomplete factorization, or in an SOR or SSOR iteration.

Consider the lower triangular matrix L decomposed into the following block

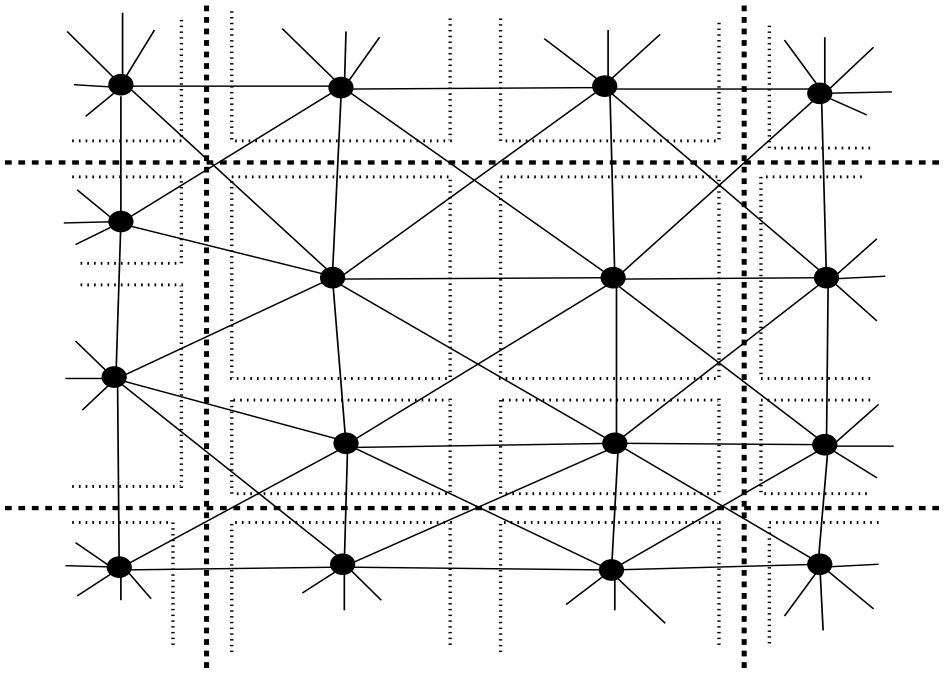

FIG. 6. *The quotient graph given the clique partition shown in Fig. 5*

For $i = 1, \ldots, \chi$ **do**

1. Local Solve (requires no interprocessor communication):
   $L_{i,i} y_i = b_i$
2. Update (communication without interdependencies):
   $b_{J_i} \leftarrow b_{J_i} - L_{J_i,K_i} y_{K_i}$

enddo

FIG. 7. *A general framework for the parallel forward elimination of the lower triangular system* $Ly = b$

structure.

$$\begin{bmatrix} L_{1,1} & 0 & 0 & 0 \\ L_{2,1} & L_{2,2} & 0 & 0 \\ \vdots & \vdots & \ddots & 0 \\ L_{\chi,1} & L_{\chi,2} & \cdots & L_{\chi,\chi} \end{bmatrix}. \tag{4.1}$$

In Fig. 7 we present a general framework for the forward elimination required to solve the system $Ly = b$. By y_i and b_i we mean the partition of components implied by the block partition of L given above. The index sets J_i and K_i can be anything equivalent to the standard forward elimination algorithm. With this framework we divide the solution in two phases. In phase 1, the diagonal block solution phase, we assume that no interprocessor communication is required. In the second phase, when the partial updates to the right hand side are performed, we include all

the interprocessor communication, but we assume that this communication can be performed in any order. Thus, the number of major communication step required in this framework is χ.

We classify a number of possible approaches to solving these triangular systems based on the choice of the diagonal blocks $L_{i,i}$ as follows:

Pointwise colorings – Given a coloring of the graph $G(A)$ for the incomplete factorization matrix A, we order unknowns corresponding to same colored vertices consecutively. An implementation based on this approach and computational results are given in [10].

Partitioned inverse – One can determine a product decomposition of L; for example,

$$L = \prod_{i=1}^{\kappa} L_i \ , \tag{4.2}$$

where the nonzero structure, S, of the product elements satisfy $S(L_i) = S(L_i^{-1})$ [1, 2]. The inversion of L can be performed with κ matrix products once the partitioned inverse is formed. We note that this can always done with a pointwise coloring, where κ is the number of colors used. It has been observed by Robert Schreiber [18] that the partitioned inverse approach can reduce the steps in pointwise coloring approach by a factor of two. Suppose two colors are used. We write the pointwise system as

$$L = \begin{bmatrix} D_{1,1} & 0 \\ L_{2,1} & D_{2,2} \end{bmatrix} \ , \tag{4.3}$$

where $D_{1,1}$ and $D_{2,2}$ are diagonal. Schreiber makes the following observation:

$$L^{-1} = \begin{bmatrix} D_{1,1}^{-1} & 0 \\ -D_{2,2}^{-1} L_{2,1} D_{1,1}^{-1} & D_{2,2}^{-1} \end{bmatrix} \ , \tag{4.4}$$

where the structures of L and L^{-1} are identical. Thus, one can group pairs of colors together and form the inverse of the combined diagonal block by a simple rescaling the off-diagonal part.

Nodewise colorings – Identify adjacent vertices with identical structure. As described in §3, such vertices often arise in finite element models for independent degrees of freedom defined at the same geometric node. Let the set I identify identical nodes. A matrix ordering based on a coloring G/I, where identically colored nodes are ordered consecutively, yields a system where $L_{i,i}$ is block diagonal, with dense blocks the size of the number of identical nodes at each node point. Given a geometric partition of the nodes, these dense blocks are local to a processor. In addition, the observation made by Schreiber and illustrated in Equation 4.4 can be used

to decrease the number of major communication step by a factor of two for a nodewise coloring. The inverse formula given in Equation 4.4 with $D_{1,1}$ and $D_{2,2}$ block diagonal will still preserve the nonzero structure of L, because the nonzero structure of the columns in each dense block are identical.

Quotient graph colorings derived from a local clique partition – This approach is used in our implementation. The local cliques correspond to local dense diagonal blocks in $L_{i,i}$. The inverses of these blocks are computed. Thus the local solve, step 1 in Fig. 7, can be implemented using Level-2 BLAS. Usually the number of colors required to color the quotient graph will be smaller than the number of colors required for the original graph. However, if fewer colors are used, recent theoretical results [11] indicate that the convergence of the iterative algorithm could suffer. This aspect is discussed more fully in §5.

Quotient graph colorings derived from general local systems – Any local structure can chosen for the diagonal systems $L_{i,i}$. However, if general sparse systems are used, the processor performance is not necessarily improved over a pointwise coloring. In addition, load balancing becomes more difficult as larger partitions are chosen.

Given the above possibilities, we have chosen to implement a method based on quotient graph colorings derived from a local clique partition. This approach enables our software to take advantage of both any identical node structure and local clique partitions. The former allows for a reduction in the indirect indexing required; the latter allows for the use of larger dense blocks and consequentially better performance with the Level-2 BLAS. The software is designed so that the maximum size of the identical node sets, the maximum clique size, and maximum number of cliques per color can all be set by the user in case of load balancing or convergence problems. However, for the results presented in §5, no such limits were imposed.

5. Computational Results. In this section we present computational results obtained on the Intel DELTA with the software we have developed. We consider two applications: a piezoelectric crystal modeling problem, and a three-dimensional superconductivity modeling problem. These problems are described in more depth in [12]; we give only a brief description of them here.

5.1. The piezoelectric crystal modeling problem. The first set of sparse systems that we consider arise from a second-order finite element of a piezoelectric crystal strip oscillator. These crystals are thin strips of quartz that vibrate at a fixed frequency when an electric forcing field is applied to the crystal. A diagram of a strip oscillator affixed to an aluminum substrate with epoxy is shown in Fig. 8.

Second-order, 27-node finite elements are used to model the crystal. Higher-order elements are required to accurately model high frequency vibrational modes

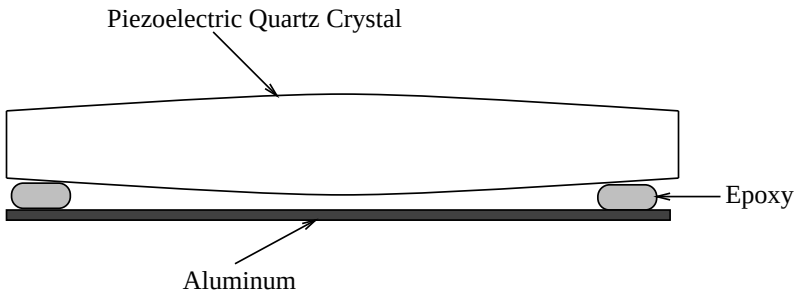

FIG. 8. *Piezoelectric crystal strip oscillator*

of the crystal. There are four degrees of freedom at each geometric node point: three mechanical displacements and an electric field potential. The solution phase has two steps. First, the deformation of the crystal caused by thermal displacement is found. For example, if the crystal was mounted on aluminum at 25^oC, it will deform when the temperature is raised to 35^oC. This requires solving a nonlinear static thermal stress problem. Second, to find the vibrational modes of interest for the deformed crystal, we solve a linear vibration problem – a generalized eigenproblem.

To solve the nonlinear static thermal stress problem, a series of linear systems of the form $Ku = f$ must be solved, where K represents the stiffness matrix, u represents the displacements, and f represents the forces due to thermal loads and displacement constraints. The major task here, of course, is the solution of very large, sparse systems of equations.

To solve the linear vibration problem, we must solve a generalized eigenproblem of the form $Kx = \omega^2 Mx$, where K represents the stiffness matrix, M represents the mass matrix, x is a vibrational modeshape, and ω is a vibrational mode. We use a shifted, inverted variant of the Lanczos algorithm to solve this eigenproblem [16]. This method has been shown to be very efficient for the parallel solution of the vibration problem [9]. Again, the major computational task is the solution of large sparse systems of linear equations.

The three-dimensional finite element grid needed to model the crystals is much more refined in the length and width directions than it is in the thickness direction. We can take advantage of this fact and partition the grid among the processors in only the length and width directions. This approach reduces communication and maps nicely onto the DELTA architecture. Each processor is assigned a rectangular solid corresponding to a portion of the three-dimensional grid. Each processor is responsible for evaluating the finite elements in its partition and for maintaining all relevant geometric and solution data for its partition.

We have solved problems consisting of over 480,000 equations with 161,150,990 nonzeros on 512 processors of the Intel DELTA. Over 99 percent of the time is spent

in solving the linear systems and evaluating the finite elements. The solution of the linear systems has achieved speeds of approximately 2 gigaflops on 512 processors. This speed is scalable; the individual processor performance degrades only from 4.16 Mflops per processor to 3.83 Mflops per processor when one goes from 128 processors to 512 processors and keeps the sub-grid size fixed.

5.2. The layered superconductor modeling problem. The sparse linear systems for the superconductivity problem arise in the determination of the damped Newton step in the inner loop of an optimization algorithm. The optimization algorithm attempts to determine the minimizer of a free energy functional that is defined on a three-dimensional rectangular mesh with the geometric layout depicted in Fig. 9. The structure of the sparse linear system is determined by the Hessian of free energy given a linear finite difference discretization of the model.

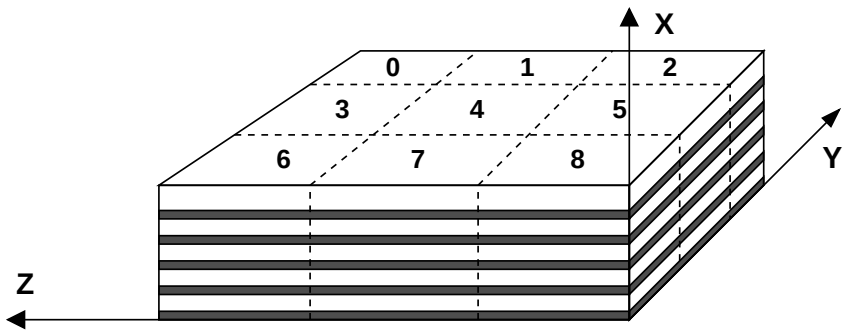

FIG. 9. *The 3-dimensional layered superconductor model partitioned in 2-dimensions*

Shown in the figure are alternating layers of superconducting and insulating material. The independent variables are two vector fields, one defined in the superconducting sheets, and the other in the insulating layer. The two fields are coupled in the free energy formulation. When the model is discretized a finer degree of resolution is generally given to the insulating layers. For the problems of interest, the number of grid points necessary to represent the model in the direction perpendicular to the layers (the X-axis in Fig. 9) is smaller than the number of points required in the two directions parallel to the layers (the Y-axis and Z-axis in Fig. 9). We make use of this property and partition the grid in the Y and Z directions. For example, in Fig. 9 the Y-Z domain is shown partitioned among 9 processors.

We denote the discretization in the X, Y, and Z directions by NX, NY, and NZ, respectively. As the discretization within an insulating layer, NK, varies, the size of the local cliques changes, and therefore so does the individual processor

performance. In Table 1 we note the effect of varying the layer discretization on the i860 processor performance during the solution of the linear systems. For these numbers we have used 128 processors and fixed the local problem size to be roughly equivalent. The second column shows the number of identical nodes found in the graph by the solver; the third column shows the average clique size found. The final column shows the average computational rate per processor during the solution of the linear systems.

TABLE 1
The effect of varying the layer discretization on the processor performance in solving the linear systems

NK	I-Node Size	Avg. Clique Size	Avg. Mflops/Processor
2	8	32.0	2.97
4	14	44.8	5.42
6	20	60.0	6.71
8	26	78.0	8.96

In Table 2 we present results for the linear solver on three problems with differing geometric configurations on 512 processors.

TABLE 2
Computational results obtained for three different problem configurations on 512 processors

	PROBLEM-1	PROBLEM-2	PROBLEM-3
NX	24	64	20
NK	8	4	2
NY	80	64	150
NZ	96	96	150
N	6.0×10^5	1.6×10^6	1.8×10^6
NNZ	2.0×10^8	1.7×10^8	1.9×10^8
GFlops	3.25	2.55	1.38

In the solution of both of these systems, the diagonal of the matrix was scaled to be one. If the incomplete factorization fails (a negative diagonal element created during the factorization), a small multiple of the identity is added to diagonal, and the factorization is restarted. This process is repeated until a successful factorization is obtained [15]. The average number of conjugate gradient iterations required to solve one nonlinear iteration of the thermal equilibrium problem for the crystal model to a relative accuracy of 10^{-7} is approximately 700. The average number of conjugate gradient iterations required per nonlinear iteration for the superconductivity problem is approximately 250. The linear systems arising in the superconductivity problem are solved to a relative accuracy of 5.0×10^{-4}. However, it should be noted that these are special linear systems: they are highly singular (more than one-fifth of the eigenvalues are zero, because of physical symmetries).

However, they are consistent near a local minimizer because a projection of the right hand side (the gradient of the free energy function) onto the null space of the matrix is zero near the minimizer.

6. Conclusions. In this paper we have presented an implementation of a general-purpose iterative solver for MIMD machines. The scalable performance of the solver is based on a reordering of the sparse system according to a graph coloring of a reduced graph obtained from the nonzero structure of the sparse linear system. This approach is effective for any of the standard iterative methods; however, the experimental results we present are for the conjugate gradient algorithm with an incomplete matrix factorization preconditioner.

We have emphasized an approach where all the manipulations required by the solver are all done in parallel. In this spirit, we have presented two recently developed parallel heuristics for determining a graph coloring. We have shown that the synchronous heuristic proposed by Luby, based on determining a sequence of maximal independent sets, can be modified to run in an asynchronous manner. Furthermore, we show that the expected running time of the modified heuristic is $EO(\log(n)/\log\log(n))$ for bounded degree graphs using the bounds developed for the other coloring heuristic.

A number of possible approaches toward the solution of the sparse triangular system solutions are classified. We have chosen to use a graph reduction based on a clique partition in our implementation for two reasons: (1) to allow for the use of higher-level BLAS in a triangular system solver, and (2) to reduce the number of required colors and the size of the quotient graph. The implementation allows the user to specify the maximum clique size and the maximum number of cliques per color, in case load-balancing or convergence problems arise. In the experimental results section we demonstrate the improvement in processor performance for larger clique sizes for the superconductivity problem. In addition, the concentration of the basic computation in the BLAS allows for an efficient, portable implementation.

Finally, we note that recent theoretical results have shown that for a model problem, the convergence rate improves as the number of colors is increased [11]. This possibility was investigated for the piezoelectric crystal problem, and a definite, but moderate, decrease in the convergence rate was found in going from a pointwise coloring (≈ 108 colors) to a clique coloring (≈ 10 colors). However, the increase in efficiency of the implementation for the clique coloring more than offset the convergence differences.

Overall, we feel that this approach represents an effective approach for efficiently solving large, sparse linear systems on massively parallel machines. We have demonstrated that our implementation is able to solve general sparse systems from two different applications, achieving both good processor performance and convergence properties.

Acknowledgment. The second author acknowledges a number of helpful discussions with Fernando Alvarado, Stanley Eisenstat, and Robert Schreiber while attending the IMA workshop.

REFERENCES

[1] F. L. Alvarado, A. Pothen, and R. Schreiber, *Highly parallel sparse triangular solution*, Tech. Rep. CS-92-09, The Pennsylvania State University, May 1992.

[2] F. L. Alvarado and R. Schreiber, *Optimal parallel solution of sparse triangular systems*, SIAM Journal on Scientific and Statistical Computing, (to appear).

[3] D. Brélaz, *New methods to color the vertices of a graph*, Comm. ACM, 22 (1979), pp. 251–256.

[4] T. F. Coleman and J. J. Moré, *Estimation of sparse Jacobian matrices and graph coloring problems*, SIAM Journal on Numerical Analysis, 20 (1983), pp. 187–209.

[5] M. R. Garey and D. S. Johnson, *Computers and Intractability*, W. H. Freeman, New York, 1979.

[6] J. L. Gustafson, G. R. Montry, and R. E. Benner, *Development of parallel methods for a 1024-processor hypercube*, SIAM Journal on Scientific and Statistical Computing, 9 (1988), pp. 609–638.

[7] L. A. Hageman and D. M. Young, *Applied Iterative Methods*, Academic Press, New York, 1981.

[8] D. S. Johnson, *Worst case behavior of graph coloring algorithms*, in Proceedings 5th Southeastern Conference on Combinatorics, Graph Theory, and Computing, Utilitas Mathematica Publishing, Winnipeg, 1974, pp. 513–527.

[9] M. T. Jones and M. L. Patrick, *The Lanczos algorithm for the generalized symmetric eigenproblem on shared-memory architectures*, Preprint MCS-P182-1090, Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, Ill., 1990.

[10] M. T. Jones and P. E. Plassmann, *Scalable iterative solution of sparse linear systems*, Preprint MCS-P277-1191, Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, Ill., 1991.

[11] ——, *The effect of many-color orderings on the convergence of iterative methods*, Preprint MCS-P292-0292, Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, Ill., 1992.

[12] ——, *Solution of large, sparse systems of linear equations in massively parallel applications*, Preprint MCS-P313-0692, Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, Ill., 1992.

[13] ——, *A parallel graph coloring heuristic*, SIAM Journal on Scientific and Statistical Computing, (to appear).

[14] M. Luby, *A simple parallel algorithm for the maximal independent set problem*, SIAM Journal on Computing, 4 (1986), pp. 1036–1053.

[15] T. A. Manteuffel, *An incomplete factorization technique for positive definite linear systems*, Mathematics of Computation, 34 (1980), pp. 473–497.

[16] B. Nour-Omid, B. N. Parlett, T. Ericsson, and P. S. Jensen, *How to implement the spectral transformation*, Mathematics of Computation, 48 (1987), pp. 663–673.

[17] A. Pothen, H. Simon, and K.-P. Liou, *Partitioning sparse matrices with eigenvectors of graphs*, SIAM Journal on Matrix Analysis, 11 (1990), pp. 430–452.

[18] R. Schreiber. Private communication, 1991.

[19] R. Schreiber and W.-P. Tang, *Vectorizing the conjugate gradient method*. Unpublished manuscript, Department of Computer Science, Stanford University, 1982.

[20] H. A. van der Vorst, *High performance preconditioning*, SIAM Journal on Scientific and Statistical Computing, 10 (1989), pp. 1174–1185.

[21] S. Vavasis, *Automatic domain partitioning in three dimensions*, SIAM Journal on Scientific and Statistical Computing, 12 (1991), pp. 950–970.