ARGONNE NATIONAL LABORATORY
9700 South Cass Avenue
Argonne, Illinois 60439

SCALABLE ITERATIVE SOLUTION OF SPARSE LINEAR SYSTEMS*

Mark T. Jones and Paul E. Plassmann

Mathematics and Computer Science Division
Preprint MCS-P277-1191
November 1991

ABSTRACT

The efficiency of a parallel implementation of the conjugate gradient method preconditioned by an incomplete Cholesky factorization can vary dramatically depending on the column ordering chosen. One method to minimize the number of major parallel steps is to choose an ordering based on a coloring of the symmetric graph representing the nonzero adjacency structure of the matrix. In this paper, we compare the performance of the preconditioned conjugate gradient method using these coloring orderings with a number of standard orderings on matrices arising from finite element models. Because optimal colorings for these systems may not be known *a priori*, we employ a graph coloring heuristic to obtain consistent colorings. Based on lower bounds obtained from the local structure of these systems, we find that the colorings determined by the heuristic are nearly optimal. For these problems, we find that the increase in parallelism afforded by the coloring-based orderings more than offsets any increase in the number of iterations required for the convergence of the conjugate gradient algorithm. We give results from the Intel iPSC/860 to support our claims.

Key words: conjugate gradient methods, distributed-memory computers, graph coloring heuristics, incomplete Cholesky, parallel algorithms, sparse matrices

AMS(MOS) subject classifications: 65F10, 65F50, 65Y05, 68Q22

*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 preconditioned conjugate gradient method [13] is one of the most successful iterative methods for solving large, sparse, symmetric, positive-definite linear systems. A preconditioner that has been shown to be very effective for a wide variety of problems is the incomplete Cholesky factorization [18]. Recently, several authors [5, 7, 20, 22] have examined the effect of multicoloring based matrix orderings on the convergence properties of iterative methods. However, this work has considered only problems generated from regular grids, for which an optimal coloring is known *a priori*. These problems generate M-matrices [19] that are not representative of general systems of equations for which the straightforward incomplete Cholesky factorization may not exist.

In this paper, we consider sparse linear systems that arise both from finite element models and standard grid problems. For many of the former problems, optimal multicolorings are not known. In general, the determination of an optimal coloring is an NP-hard problem [8]. Thus, we have explored the use of graph coloring heuristics to obtain the desired orderings. Our experimental results show that the combination of incomplete factorization and coloring heuristics results in a parallel preconditioner that is applicable to the symmetric, positive-definite matrices arising in many applications. We also compare the effectiveness of the coloring heuristics to some standard orderings: minimum degree, reverse Cuthill-McKee, and nested dissection.

The parallelism inherent in computing and applying the preconditioner is limited by the solution of the triangular systems generated by the incomplete Cholesky factors [22]. It was noted by Schreiber and Tang [21] 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.

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 the quotient graph obtained from the partitioning of unknowns to processors. This approach can lead

to load-balancing problems, however, and our main objective is the demonstration of scalable performance. Thus, in our current implementation, we allow only the solution of diagonal systems between communication steps.

The combination of graph coloring and incomplete factorization gives a parallel algorithm that is scalable as defined in [11]. In general, for graphs arising from physical models, the maximum degree of any node is bounded independently from the number of nodes. Because the number of colors is bounded by the maximum degree of a node, the number of parallel steps is also bounded independently from the problem size. The computation rate per processor should therefore be constant as the number of processors increases, given a fixed problem size per processor.

This matrix permutation is reminiscent of the reorderings done to minimize the fill in a direct factorization. However, in an incomplete factorization, fill that corresponds to initial zeros of the matrix is ignored. Instead, the permutation is chosen to minimize the number of communication steps inherent in the solution of the triangular systems generated by the incomplete factorization.

It is important to note that the scalability results presented in this paper apply to most classical iterative methods. The successive overrelaxation (SOR) method with consistent ordering[1] could be implemented in a scalable multi-level algorithm using graph coloring, as briefly outlined in [17]. One could also use the symmetric successive overrelaxation method (SSOR) as a preconditioner for the conjugate gradient algorithm, or as a stand-alone iterative method, and obtain the same communication complexity. We have chosen to implement incomplete Cholesky as a preconditioner because we have observed that it provides a more effective preconditioner for the applications with which we have been concerned.

The organization of this paper is as follows. In Section 2 we briefly present some important issues in computing graph colorings, and we review the coloring heuristic that we have employed in our experiments. In Section 3 we present and discuss an execution time model for the triangular matrix solutions used in the incomplete factorization algorithm. In Section 4 we introduce a suite of test problems and present experimental results. Finally, in Section 5 we summarize our research and suggest areas for future investigation.

2. Coloring Heuristics. Given the nonzero structure of an $n \times n$ symmetric matrix A, one can associate the symmetric graph $G(A) = (V, E)$ with the matrix, where the vertex set is given by $V = \{1, \ldots, n\}$ and the edge set is given by $E = \{\, (i,j) \mid A_{ij} \neq 0, \text{ and } i \neq j \,\}$. We say that the function $\sigma : V \rightarrow \{1, \ldots, s\}$ is an s-coloring of $G(A)$, if $\sigma(i) \neq \sigma(j)$ for all edges (i,j) in E. The minimum possible value for s is known as the *chromatic number* of $G(A)$, which we denote as $\chi(G)$.

The question of whether a general graph $G(A)$ is s-colorable is NP-complete [8]. It is known that unless $P = NP$, there does not exist a polynomial approximation scheme for solving the graph coloring problem [8]. In fact, the best polynomial time heuristic known [14] can theoretically guarantee a coloring of only size $c\,(n/\log n)\,\chi(G)$, where

[1] With a consistently ordered matrix, it is straightforward to determine the optimal relaxation parameter [12].

c is some constant.

It is therefore rather surprising that a coloring heuristic could perform well in practice. However, for the problems considered in this paper, we find that the heuristics obtain colorings only slightly worse than a lower bound determined from the local structure of the graphs considered. To obtain this lower bound, we employ the following well-known result which bounds the chromatic number by the size of any complete subgraph in G.

Given a subset V' of the vertices V, the *induced subgraph* $G' = (V', E')$ of G contains the edges in the set $E' = \{ (i,j) \mid (i,j) \in E, \text{ and } i,j \in V' \}$. A complete subgraph of size r, which we call an *r-clique*, is a subset V' of V, with $|V'| = r$, for which every possible edge exists in the induced subgraph. Since the r vertices in an r-clique must be assigned different colors, we simply state the following well-known lemma.

LEMMA 2.1. *If G contains an r-clique, then $\chi(G) \geq r$.*

The matrices we have considered for our experiments arise from finite element models. We say that the finite element model contains an *r-element* if there is an element in the model that contains r directly interacting independent variables. Thus, in the graph representing the corresponding linear system, we obtain an r-clique associated with this r-element. By the above lemma, this clique reveals to a lower bound for the chromatic number of the graph $G(A)$. The advantage of this observation is that it is usually straightforward to determine the maximum sized r-element in the finite element model.

COROLLARY 2.2. *If the finite element model contains an r-element, $\chi(G) \geq r$.*

Frequently, each node in the finite element model may have q unknowns associated with it. In the matrix A, unknowns associated with the same node in the finite element model are usually structurally identical. These q identical unknowns can be colored the same color if one is willing to solve block diagonal submatrices of L, rather than diagonal submatrices [21]. If this approach is used and each node in the model has q structurally identical unknowns, the minimum number of major parallel steps could be reduced to $\chi(G)/q$. In our demonstration of scalability, we have not chosen this approach.

It is known that an optimal coloring can be obtained via a greedy heuristic if the vertices are visited in the correct order [1]. The basic structure of the greedy heuristic is the following.

GREEDY HEURISTIC. *Compute a vertex ordering $\{v_1, \ldots, v_n\}$ for V. For $i = 1, \ldots, n$, set $\sigma(v_i)$ equal to the smallest available consistent color.*

The maximum degree of the graph determines an upper bound for the chromatic number [1]. Let $\Delta(G) = \max_{v \in V} deg(v)$, where $deg(v)$ is the degree of vertex v in G. This upper bound is given by $\chi(G) \leq \Delta(G) + 1$. Note that a greedy heuristic will always satisfy this bound. Also, independent of the size of system, the graphs arising from the applications we consider have bounded degree. Thus, the colorings determined by a greedy heuristic will also be bounded.

The only aspect of the greedy heuristic that must be specified is the method for obtaining the initial vertex ordering. Several strategies for obtaining this vertex

ordering have been proposed in work by other authors. Two of the most effective and efficient strategies proposed are orderings of the vertices by saturation degree and by incidence degree.

The *saturation degree ordering* (SDO) heuristic was first proposed by Brélaz [2] and is defined as follows. Suppose that vertices $v_1, \ldots, v_{i-1}$ have been chosen. Vertex v_i, the next vertex chosen, is a vertex adjacent to the maximum number of different colors in the vertex set $\{v_1, \ldots, v_{i-1}\}$. The SDO heuristic can be implemented to run in time proportional to $\sum_{v \in V} deg^2(v)$.

A modification of the SDO heuristic, the *incidence degree ordering* (IDO) heuristic, was suggested by Coleman and Moré in their work [4] on using coloring heuristics to obtain consistent partitions for use in Jacobian estimation. Again suppose that vertices $v_1, \ldots, v_{i-1}$ have been chosen. Vertex v_i is chosen to be a vertex whose degree is a maximum in the induced subgraph of G with vertices $\{v_1, \ldots, v_{i-1}\}$. The IDO heuristic has the desirable property that it can be implemented to run in a time proportional to $\sum_{v \in V} deg(v)$, or the number of nonzeros in the matrix. This heuristic was found by Coleman and Moré [4] to obtain the best colorings, for a linear time heuristic, over a wide variety of problems.

The computational cost of these graph coloring heuristics is modest compared to the time required to compute the incomplete factors and repeatedly solve the resulting triangular systems. For the results presented in this paper, the IDO heuristic was used to obtain the graph colorings used in the matrix reorderings. However, we note that recently a scalable, parallel coloring heuristic has been developed [16] that is able to determine colorings comparable to these sequential heuristics. In a complete parallel implementation, the use of a parallel heuristic is preferable; we have used the sequential coloring heuristic because our comparison is with other sequential ordering heuristics.

3. Analytic Execution Time Model. In this section we present a model of the computational complexity of a parallel implementation of the solution of the triangular systems involving the incomplete Cholesky factors. First we review the connection between a coloring of the graph G, and the solution of a triangular system with the nonzero structure corresponding to this graph.

Given an s-coloring σ of G, we define a directed graph $D_\sigma = (V, F)$ as follows. For each edge, $(u, v) \in E$, construct the directed edge, $< x, y > \in F$, directed from x to y, where $\sigma(x) < \sigma(y)$ and $\{x, y\} = \{u, v\}$. This construction is well defined because there cannot exist an edge $(u, v) \in E$ with $\sigma(u) = \sigma(v)$, since σ is a coloring of G.

By this construction, the directed graph D_σ is acyclic. In the discussion that follows, we assume, for notational simplicity, that the component indices v, w of vectors and matrices correspond to a topological ordering of this directed acyclic graph (DAG). The complexity of traversal of the DAG, D_σ, can be seen to be equivalent to the complexity of solution of the lower triangular system $Ly = b$, where L is the incomplete Cholesky factor. In the standard forward elimination algorithm the traversal of the edge $< w, v >$ in D corresponds to the computational step

$$b_v \leftarrow b_v - L_{vw} y_w \ . \tag{3.1}$$

Likewise, each vertex v in the DAG corresponds to the computational step

$$y_v \leftarrow b_v / L_{vv} \ . \tag{3.2}$$

For vertices w and v with $\sigma(w) = \sigma(v)$, the computation of y_w and y_v, as given in equation (3.2), can be done in parallel, given that the updates to b_w and b_v have been completed. Thus, the triangular system may be solved in parallel with χ_σ major communication steps. An outline of the parallel algorithm executed by each processor for forward elimination is given in Figure 1. In this algorithm the function $proc(v)$ returns the processor assigned the v-th component of y; a processor's own processor number is me.

```
For i = 1, ..., χ_σ do
    For each v ∈ V with σ(v) = i do
        b̃_v ← 0;
        For each <w, v> ∈ F with proc(w) = me do
            b̃_v ← b̃_v − L_vw y_w;
        enddo
        If proc(v) ≠ me then
            Send b̃_v to proc(v);
        else
            b_v ← b̃_v;
            For each update b̃_v do
                Receive update b̃_v;
                b_v ← b_v + b̃_v;
            enddo
            y_v ← b̃_v / L_vv;
        endif
    enddo
enddo
```

FIG. 1. *Parallel algorithm for the forward elimination of the lower triangular system* $Ly = b$

In our implementation of this algorithm, all updates $\tilde{b}_v$ and $\tilde{b}_w$ with $\sigma(v) = \sigma(w)$ to be sent from processor a to processor b are combined into one message. This combination overcomes the high cost of starting a message. We also note that the communication pattern for back substitution required to solve the system $L^T x = y$ is the exact reverse of what is given in Figure 1. Rather than receiving the updates $\tilde{b}_v$, $proc(v)$ sends x_v to the same processors it received updates from in the forward elimination phase. Thus, the communication complexities of the forward elimination and back substitution algorithms are equivalent.

3.1. Bounds on the Communication Complexity. Given certain assumptions, one can construct an analytic execution time model for the triangular matrix

solutions required by the incomplete factorization algorithm. In this section we construct such a model for a message-passing parallel computer. We make the following assumptions about the partitioning of the problem across the processors.

1. *Bounded connectivity* – We assume that the problem is partitioned such that processor i sends/receives at most an average of c_i messages per level of the DAG.
2. *Load balancing* – We assume that every processor has a roughly equal share of the nodes of each level of the directed acyclic graph associated with the triangular matrix solution.

We also assume that the time required to send k bytes between two processors, $t(k)$, obeys the linear relation

$$t(k) = \alpha + \beta k \ , \tag{3.3}$$

where α is the time for starting a message and β is the time per byte for sending a message.

Given these assumptions, we obtain a lower bound for the execution time, T, of the algorithm on p processors:

$$T \geq \max_{i=1}^{p} \left(\tau x_i + \alpha c_i \chi_\sigma + \beta v_i \right) \ , \tag{3.4}$$

where τ is the time for a floating-point operation, x_i is the number of floating-point operations on processor i, v_i is the number of bytes sent by processor i, c_i is the average number of messages sent per level of the DAG, and χ_σ is the height of the DAG. Any contention for communication paths in the interconnection network is ignored in this model. The significance of this omission depends on the topology of the interconnection network and the number and size of messages being sent.

The first assumption can be satisfied for most problems arising from physical models if they are properly partitioned; the second assumption is crucial and depends on the ordering of the nodes. In this paper we compare the orderings produced by the IDO heuristic with three other standard orderings: minimum degree (MDO), nested dissection (NDO), and the reverse Cuthill-McKee ordering (RCM) [9]. If the problem partition on each processor is similar in structure, then the second assumption holds for the minimum degree ordering and the incidence degree ordering, because they are locally generated orderings that impose no global structure on the matrix. For the nested dissection heuristic, the second assumption holds if the heuristic is able find good separators.

In contrast, the RCM ordering must violate one of these assumptions. If the matrix is partitioned to maximize locality, (i.e., connected nodes are either on the same processor or on a nearby processor), then it is the nature of the RCM ordering that the nodes on each level of the DAG be on the same processor, giving an essentially sequential algorithm. If the matrix is partitioned such that the nodes on each level of the DAG are evenly distributed among the processors, the bounded connectivity assumption must be violated; processors must send messages to most other processors

at every level of the DAG. Either condition is unacceptable, of course, when attempting to construct a scalable algorithm. In Section 4, we compare the model given in equation (3.4) with experimental results obtained on the Intel iPSC/860.

An examination of this model reveals that the height of the DAG will be the major factor in determining the efficiency of any ordering. The ordering of the matrix has a significant effect on the height of the DAG; depending on the ordering, the height could be as large as n or as small as the chromatic number. On all currently available message-passing computers, α is significantly larger than τ and β [6]. Thus, the message volume contribution to the model is of secondary importance because of the relative sizes of α and β, and because, as we show below in Theorem 3.1, the message volume cannot vary greatly between orderings.

We consider the message volume when using the inner-product algorithm for the forward solve; messages are combined when possible. It is sufficient to consider the volume of messages between two processors. Let V_1 and V_2 be the set of vertices on processors 1 and 2 that are connected to vertices on the other processor, and let N_1 and N_2 be the respective size of these sets. Let E_{12} be the set of edges that connect V_1 and V_2. Note that the graph, $G_{12} = (V_1, V_2, E_{12})$, is undirected and bipartite. Let $\mathcal{M}(G_{12})$ be the size of a maximum matching for G_{12}. Recall that a maximum matching is a largest pairing of distinct vertices $\{(u_1, v_1), \ldots, (u_k, v_k)\}$ with $u_i \in V_1$ and $v_i \in V_2$ and $(u_i, v_i) \in E_{12}$. The following theorem gives a lower and upper bound to the message volume between any two processors.

THEOREM 3.1. *The minimum message volume that could be sent during the forward elimination algorithm is $\mathcal{M}(G_{12})$. The maximum message volume is $N_1 + N_2 - 1$.*

Proof: As discussed above, assume that vertices of G_{12} are ordered and labeled according to a coloring of the graph. Consider the directed acyclic graph $D_{12} = (V_1, V_2, F_{12})$ constructed from this ordering, where F_{12} are the directed edges corresponding to E_{12}. For each edge, $< v, w > \in F_{12}$, v contributes to the inner product associated with w in the forward elimination algorithm. An example of such a directed graph is given in Figure 2.

If there exist directed edges $< u, w >$ and $< v, w >$ in F_{12}, then we have two contributions to the inner product associated with w. There is no need to send both partial sums; instead, the two partial sums can be added together on one processor, and only one partial sum sent to its neighbor. We can reflect this combination in a graph construction where, for each edge $< v, w >$ in F_{12}, we remove all edges $< u, w >$ with $u < v$. After all such edges are removed, we denote this modified graph by $D_{12}^* = (V_1, V_2, F_{12}^*)$. Note that the number of edges in F_{12}^* is equal to the number of vertices in D_{12} that are terminals of directed edges (the vertex v is a terminal if there exists an edge $< u, v >$ in F_{12}). The remaining edges in the graph D_{12}^* represent all the partial sums that must be communicated in the forward elimination algorithm, and therefore the total message volume between the two processors. An example of a graph D_{12} and its reduced graph D_{12}^* is given in Figure 2.

Consider a maximum matching of the bipartite graph G_{12}. In the directed graph D_{12} there are at least $\mathcal{M}(G_{12})$ terminals of directed edges, because each matched edge will be directed to a distinct vertex in D_{12}. Since the number of edges in F_{12}^* is equal

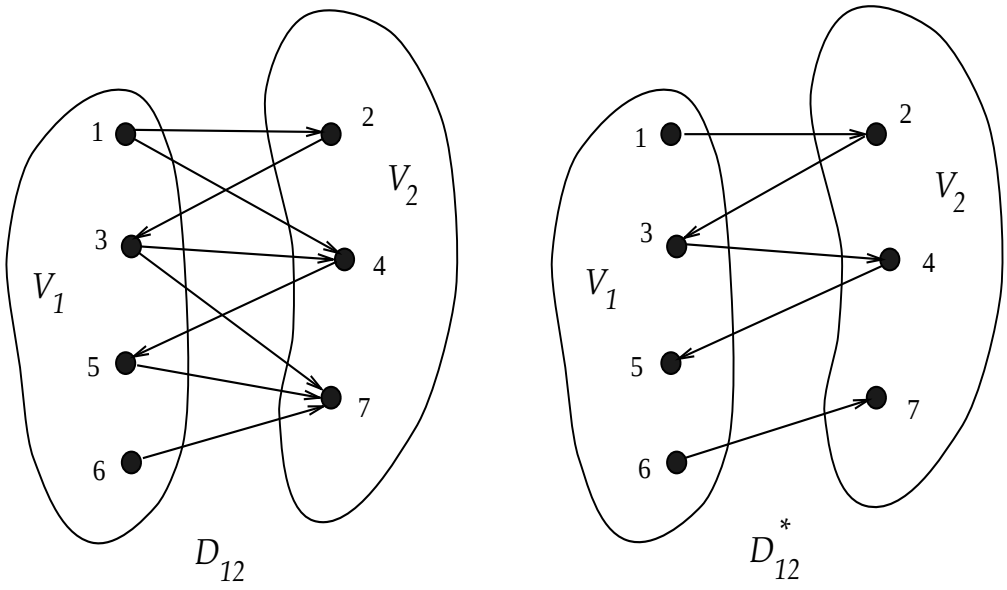

FIG. 2. *An example of the graph D_{12} and its reduced graph D_{12}^* representing the communicated partial sums*

to the number of terminals in F_{12}, we have that $|F_{12}^*| \geq \mathcal{M}(G_{12})$. To get an upper bound, we note that at most $N_1 + N_2 - 1$ vertices can be terminals of directed edges in D_{12}; thus, $|F_{12}^*| \leq N_1 + N_2 - 1$. □

We note that if one partitions the nodes of a standard finite element into two equal sets, the resulting graph partition will contain a perfect matching, since the corresponding graph is a clique. Likewise, for problems arising from physical applications, one expects $\mathcal{M}(G_{12})$ to be close to $\min(N_1, N_2)$ for the partitioned graph. Thus, the following corollary explains why one would expect less than a factor of 2 difference in message volume between different orderings.

COROLLARY 3.2. *If G_{12} has a perfect matching, then the message volume differs by at most 2 for any two orderings.*

Proof: If G_{12} has a perfect matching, we have that $N_1 = N_2 = \mathcal{M}(G_{12})$. By Theorem 3.1 we have that the upper and lower bounds for the message volume differ by less than 2. Thus, the message volume for any two orderings cannot differ by more than this amount. □

THEOREM 3.3. *The maximum number of messages that could be sent during the forward solve is $N_1 + N_2 - 1$. The minimum number of messages is one.*

Proof: Clearly, the number of messages sent cannot exceed $N_1 + N_2 - 1$. To construct a worst case example, suppose G_{12} contained a simple path that included all the vertices in G_{12}. We could then obtain a coloring σ by numbering the vertices consecutively,

in the order in which they were visited along this path. The height of the resulting DAG D_{12} is $N_1 + N_2$ and the number of terminals would be $N_1 + N_2 - 1$. Therefore, $N_1 + N_2 - 1$ messages must be sent.

To show that the minimum number of messages is one, we could order the nodes in V_1 before the nodes in V_2. All the inner products associated with V_1 can be computed without partial sums from V_2. Then all partial sums contributed by nodes in V_1 can be sent in a single message to processor 2. □

The forward elimination algorithm presented in Figure 1 could also be written to use sparse "daxpy" operations in the inner loop, as opposed to sparse inner products. We note that the communication pattern required by the daxpy-based algorithm is the same as that required by the inner product algorithm with the vertex ordering reversed. Thus, the results presented above would apply to the forward elimination algorithm if the algorithm were rewritten in this manner.

4. Experimental Results. We divide the experimental results section into two parts: (1) an evaluation of the performance of the coloring heuristic, and (2) an evaluation of the scalability of the combination of the IDO heuristic and incomplete factorization. The majority of the matrices in the test set arise from finite element models. Although it is possible to take advantage of the underlying structure of many finite elements problems, we wish to be more general and, therefore, only use information present in the assembled matrices.

A finite element model is constructed by piecing together many elements to approximate a structure. Each element typically contains $k \geq 2$ nodes, each node typically having $1 \leq d \leq 6$ degrees of freedom (dof), resulting in kd unknowns per element. Adjacent elements share nodes, reducing the total number of unknowns. Also, the number of unknowns may be reduced by other factors, including the application of constraints on the structure.

The subgraph containing these unknowns is usually completely connected and thus comprises an r-element, with $r = kd$. Many different element types, of course, can be included in a model. The coloring heuristics can be expected to perform well on these matrices because of the local nature of the models and the bound of kd on the maximum clique size.

For the sake of comparison, matrices arising from the five-point and nine-point finite-difference discretizations on a 30×30 grid were included in the test suite. It is well known that matrices arising from these stencils can be colored by using two and four colors, respectively.

In Table 1, the complete suite of test problems is described.[2] The diagonal of each matrix was scaled to be the identity matrix. For every problem except the BUILDING and PLATE problems, the right-hand side was the vector of ones, scaled to have a 2-norm of one. The initial guess for the conjugate gradient algorithm was the zero vector. Solutions were sought to a relative accuracy of 0.001, where the relative accuracy at

[2] Because of the application of constraints and the elimination of superfluous degrees of freedom, the maximum clique size for several of the problems is less than the maximum number of degrees of freedom per node times the number of nodes per element.

TABLE 1
The suite of test problems

Name	Size	Description
LAP5	900	5-pt finite difference discretization on a 30x30 grid
LAP9	900	9-pt finite difference discretization on a 30x30 grid
CUBE3	180	finite element model of a cube with 3x3x3 8-node elements with 3 degrees of freedom per node
CUBE5	636	finite element model of a cube with 5x5x5 8-node elements with 3 degrees of freedom per node
CUBE7	1524	finite element model of a cube with 7x7x7 8-node elements with 3 degrees of freedom per node
CYL11	510	finite element model of a circular cylindrical shell with 100 4-node elements with *up to* 6 dof per node
PLT4	327	finite element model of a plate with 64 4-node elements with *up to* 6 dof per node
PLT9	1295	finite element model of a plate with 64 9-node elements with *up to* 6 dof per node
PLANE	2141	finite element model of a airplane with a mixture of 2-D element types
BUILDING	6000/story	finite element model of a building using 3-D "brick" elements, the number of stories can vary
PLATE	1500/section	finite element model of a plate using 3-D "brick" elements, the number of sections in the X and Y directions, but not the Z, can vary

step k is defined as $\|r_k\|_2/\|r_0\|_2$, where r_k is the residual at step k. When the incomplete Cholesky factorization failed, we use the shifted incomplete factorization method [18] and add 0.01 to the diagonal until the factorization succeeded.

4.1. Coloring Heuristic Results. We tested the performance of the IDO heuristic on the first nine, smaller problems in Table 1. An examination of the results in Figure 3 shows that, for every problem, IDO found optimal or near-optimal colorings. A lower bound for the chromatic number was obtained by finding a large clique. The determination of the largest clique in a general graph is an NP-hard problem. We have used a heuristic to find a *large* clique, which is reported as $clique(G)$. Therefore, if the lower bound is not tight, there may exist a larger clique than this number.

4.2. Scalability Results. We compared the matrix ordering derived from the IDO graph coloring heuristic to three orderings that are applicable to *general* undirected graphs. The reverse Cuthill-McKee (RCM) heuristic has been shown in numerous tests to be the best, or nearly the best, ordering for the convergence of the ICCG(0) algorithm [5]. The nested dissection heuristic has been shown to be effective when reordering a matrix to both reduce fill-in and increase the parallelism of a direct, sparse factorization. While not as good as RCM in terms of convergence of ICCG(0), the

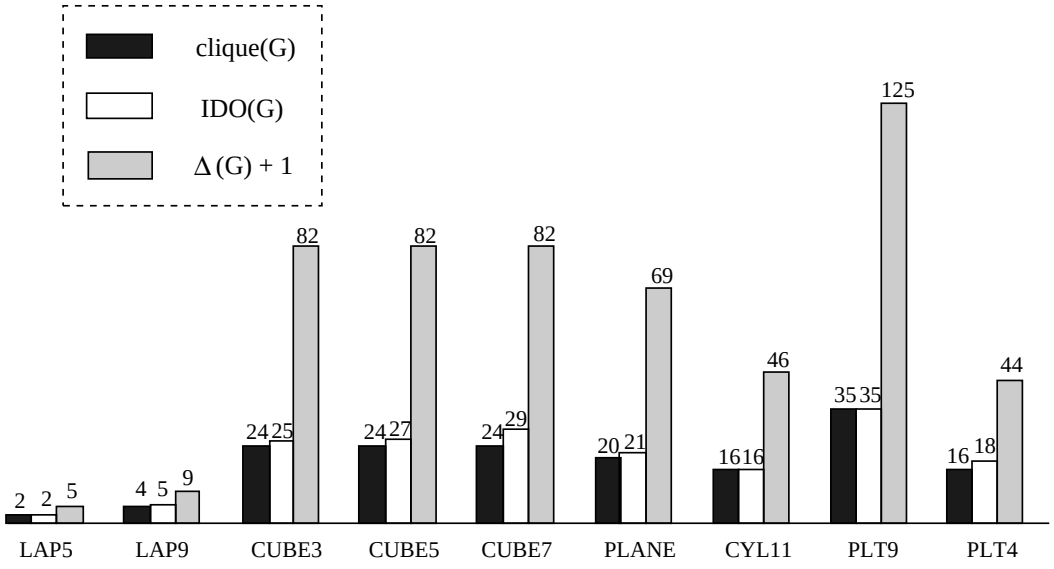

FIG. 3. *A comparison of the colorings obtained by the IDO heuristic on problems from the test suite compared with the upper and lower bounds on the chromatic number for these problems*

nested dissection heuristic was often near optimal [5]. The minimum degree ordering is the most commonly used fill reducing algorithm for direct sparse factorization; it is generally far from optimal in terms of convergence of ICCG(0) [5]. A good description of these heuristics can be found in [9].

The objective in this section is to show that a good coloring heuristic, in combination with incomplete factorization, is a scalable algorithm whose parallel performance is superior to that of standard orderings. Specifically, we show two results: (1) for the problems considered, the parallel efficiency and the total solution time for the multicoloring incomplete factorization approach is superior to other orderings, even though this ordering can have deleterious effects on convergence, and (2) the multicoloring algorithm is scalable as defined in [11].

To carry out our experiments, we selected the last two problems from Table 1, which could be scaled in size. Because of space considerations, we do not in every case present results for both problems, but generally the results are similar. The initial numbering of the equations was such that nearby nodes in the finite element models were generally close in number. Each processor was assigned a contiguous block of n/p columns, where n is the order of the matrix and p is the number of processors. This partitioning reduces the number and volume of messages that must be sent.

The effect of orderings on the convergence ICCG must be taken into account [5]. For model grid problems, Elman and Agròn [7] used a parameter study of a theoretical computational model to show that the cost of increased iterations needed for multicoloring was usually worth the benefit of increased parallelism. In our experiments we examine this benefit on the Intel iPSC/860. In Figures 4 and 5 we give the number of iterations for both problems using all four orderings. For the PLATE problem, which is geometrically similar to a model grid problem, the iteration counts for each of the orderings are close to what we expect from [5], with the exception of the iteration count for nested dissection which is higher than might be expected. This increase could result from an inability of the nested dissection heuristic to determine a good ordering because of the three-dimensional nature of this problem.

An examination of Figure 5 reveals that for the BUILDING problem, which is far from a model grid problem, the iteration counts are not similar to those from [5]. This discrepancy can be attributed to two factors: (1) different-sized shifts were needed for the orderings to obtain a positive definite preconditioner, and (2) unlike the model grid problem, where the orderings can be explicitly and regularly applied, the orderings for general graphs are heuristics whose effect is not as structured as for the model grid problems. This discrepancy has been observed in other "non-model" problems; however, in general we find that the RCM and nested dissection orderings are superior to the coloring and minimum degree orderings in terms of convergence [15]. It is interesting to note that the number of iterations in each problem grows as predicted in [3]: linearly with the relative refinement. In the PLATE problem, when the problem size quadruples, the relative refinement doubles in the X and Y directions, and the number of iterations doubles. In the BUILDING problem, when the problem size doubles, the relative refinement doubles in the Z direction, and the number of iterations doubles.

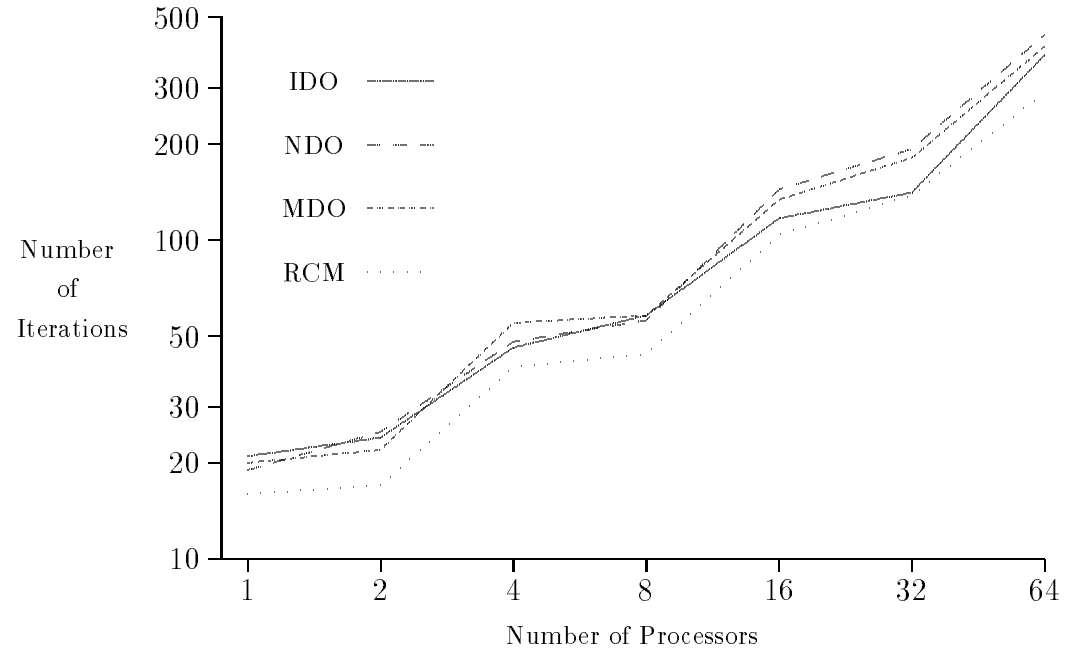

FIG. 4. *The number of iterations for the PLATE problem*

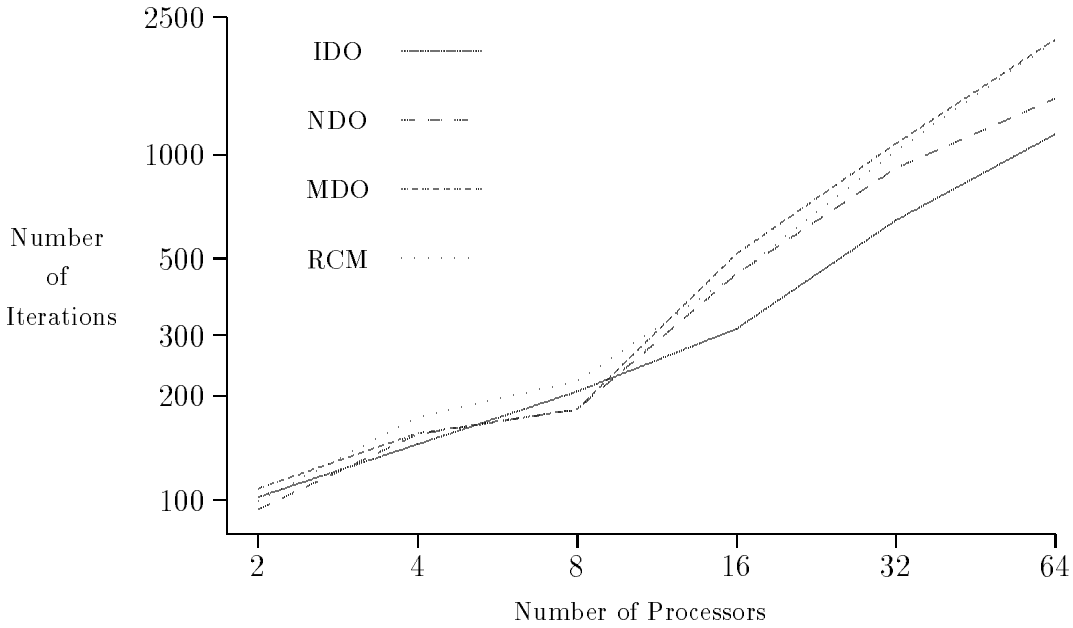

FIG. 5. *The number of iterations for the BUILDING problem*

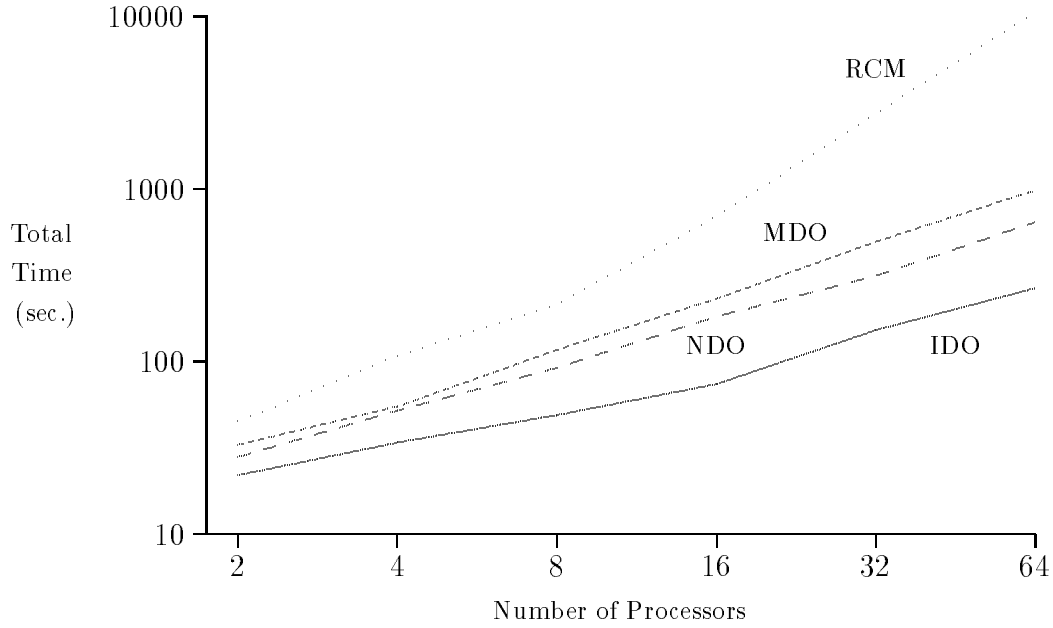

FIG. 6. *Total execution times for the BUILDING problem*

Given that colorings generally have a negative effect on convergence, we must show that their parallel efficiency is sufficiently superior to other orderings to justify their use as an ordering. To show this, we give total execution time, the time for both the incomplete factorization and the triangular matrix solutions, for the BUILDING problem in Figure 6. Note that each axis of the graph has a logarithmic scale. From this figure, it is clear that the multicolor ordering gives rise to better parallel execution times than the other orderings.

We now examine the parallel efficiency to show that it is asymptotically constant for the multicoloring algorithm. We give the computation rates for the forward triangular matrix solution for the BUILDING problem in Figure 7. The trends are similar for the incomplete factorization and for the backward triangular matrix solution and are not shown. For more than 16 processors it can be concluded that the computational rates for the multicoloring algorithm are essentially constant and are much higher than those for the other orderings. The minimum degree and nested dissection orderings appear to be almost scalable. As discussed above, the nested dissection must be able to find good global separators to be successful, unlike the minimum degree and multicolor orderings which are local ordering hueristics. However, the RCM algorithm is clearly not scalable.

It is instructive to observe both the number of messages and the message volume associated with each ordering. Given that the amount of computation on each processor is close to constant, the message traffic is a good indicator of performance. We give these numbers for the BUILDING problem in Figures 8 and 9. Not surprisingly, the number of messages is lowest for IDO and highest for RCM. Also, as expected, IDO has the highest message volume. As predicted by Corollary 3.2, the message volume varies by less than a factor of 2 between the different orderings.

We now validate the model from Section 3 using results from the PLATE prob-

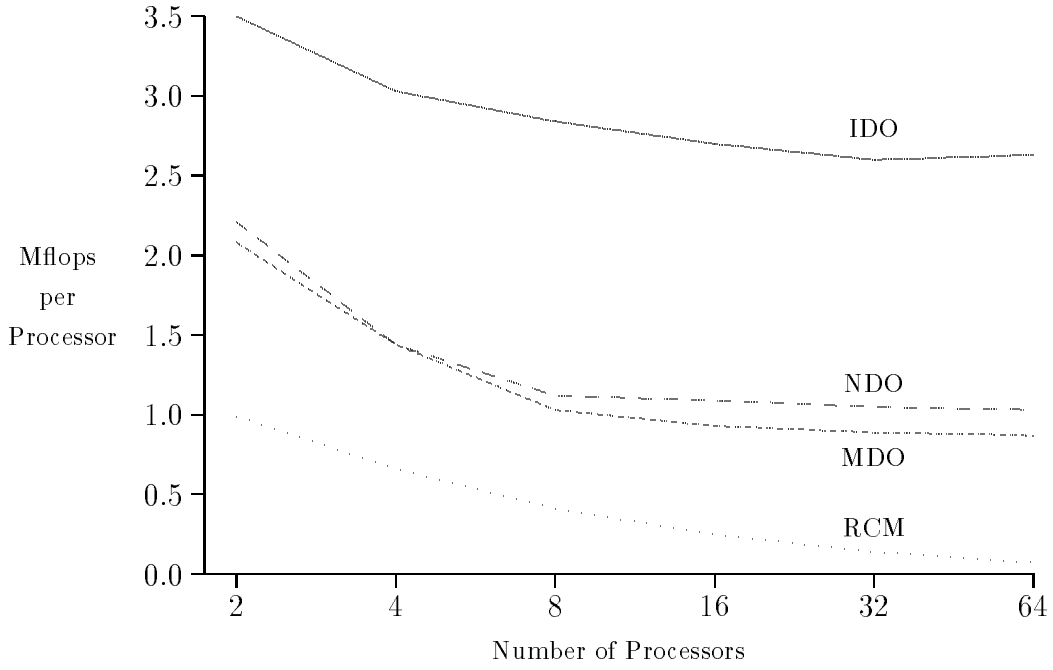

FIG. 7. *Megaflop rate per processor for forward solves on BUILDING problem*

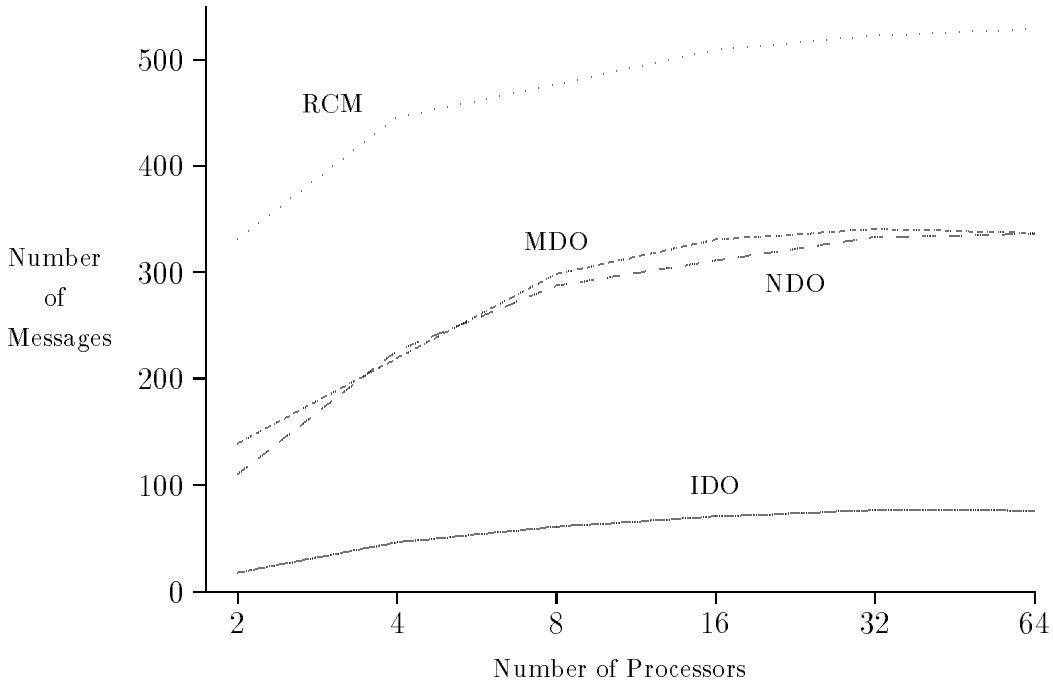

FIG. 8. *The average number of messages sent per processor*

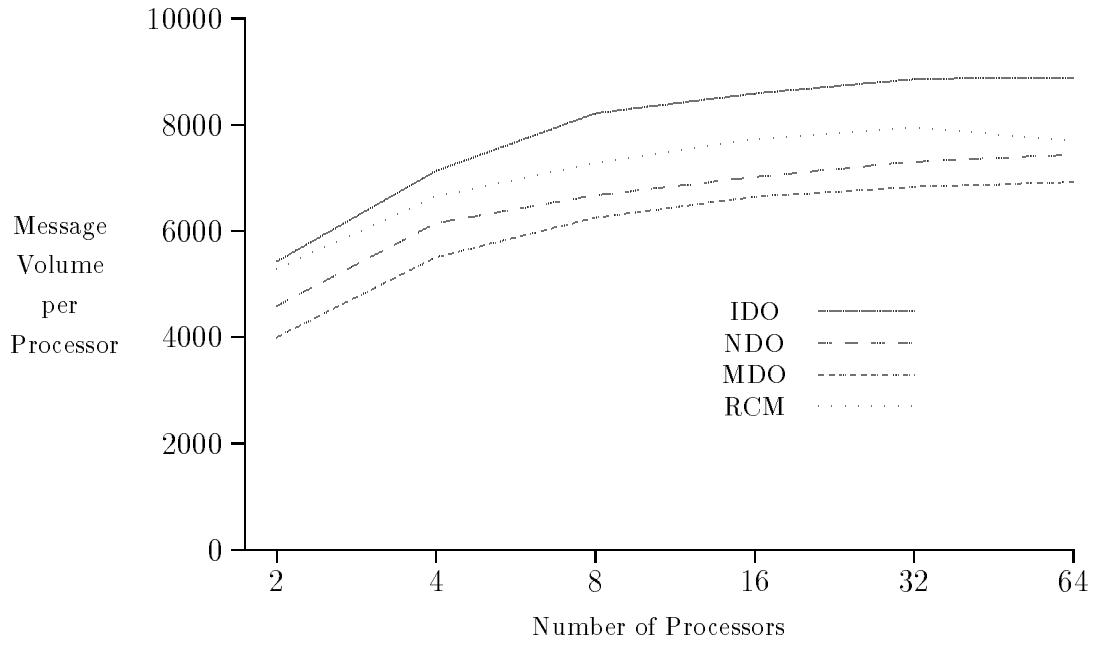

FIG. 9. *The average number of bytes sent per processor*

lem. Estimated and actual execution times are given in Figure 10. The model does a reasonable job of predicting performance for the IDO ordering. The gap is likely caused by two factors: (1) time for the data structure manipulation associated with packing messages, and (2) contention among messages for edges in the interconnection network.

5. Conclusions. Our results have shown that for the matrices considered the coloring heuristics find close to optimal colorings. The combination of the graph coloring and incomplete factorization was shown to result in a scalable, parallel algorithm. The increase in parallelism afforded by this reordering more than offsets any of the increases seen in the number of iterations required for convergence over other commonly used ordering heuristics. Lastly, we note several topics possibly requiring additional study.

The coloring produced by the heuristic may often leave a single color with very few nodes. One solution to this problem is to remove from the graph the constraining edges associated with those nodes [15] and use this smaller graph as the structure for the triangular system. However, a better approach would be an algorithm that could try to recolor a subset of the nodes to eliminate this problem.

It is well known that the convergence rate of ICCG can be improved by allowing limited fill-in during the incomplete factorization. If edges in the graph between nodes of the same color are not created during factorization, then such fill-in will not reduce the level of parallelism. In fact, such fill-in should increase the computation to communication ratio. However, by disallowing fill-in from occurring in particular edges, we may reduce the effect of fill-in on the convergence rate.

To demonstrate scalability we used problems in which the same element type was used throughout the structure. If the element mix on each processor was significantly different, then the number of nodes in each color might vary greatly on each processor.

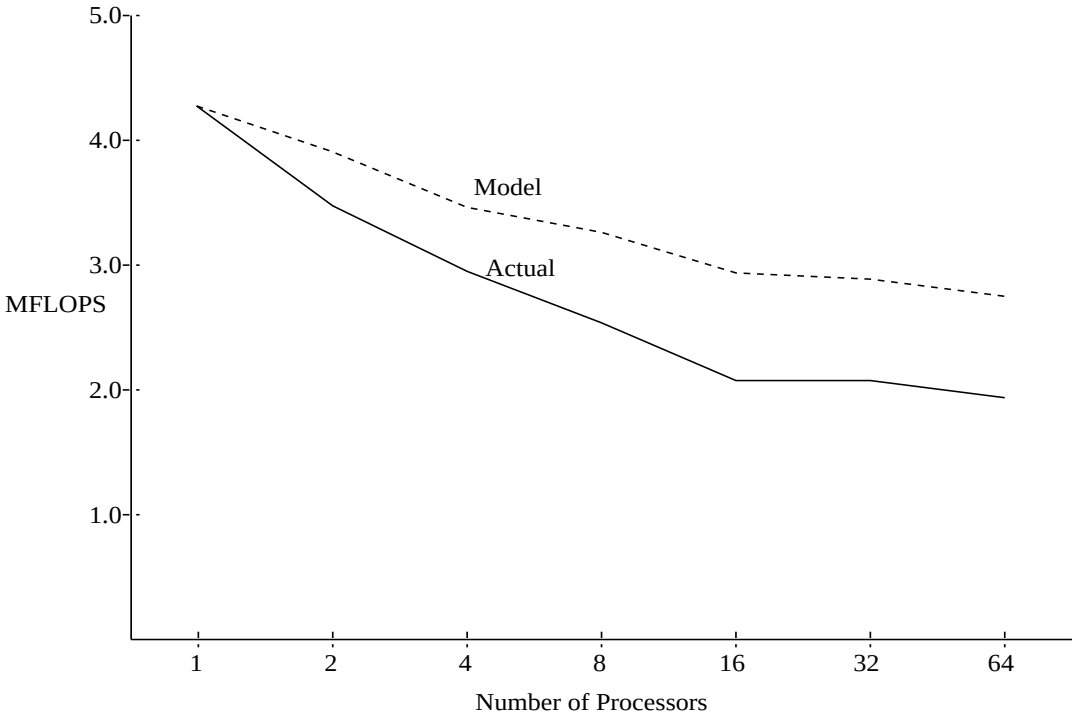

FIG. 10. *Comparison of execution time model against actual execution times*

This situation would result in a workload imbalance. However, this problem can be addressed by restricting the number of nodes on each processor that can be colored the same color, or by "shuffling" the colors to balance them.

Finally, a significant pitfall for the straightforward incomplete factorization algorithm is that it may fail to produce a positive-definite factorization, even though the matrix is positive definite. Because positive definiteness is required for the conjugate gradient method [10], some mechanism must be included to deal with the detection of indefiniteness during the incomplete factorization process. It is not sufficient to simply change a diagonal that becomes non-positive during the factorization to become negative; this approach will often result in a poor preconditioner because the non-positive diagonal was caused by something that occurred earlier during the factorization and remains in the incomplete factor. For the results presented in this paper, we have used the technique of adding an increasing multiple of the diagonal until the matrix can be successfully factored. However, we note that an improvement of these methods for forcing a positive definite factorization while still maintaining a good preconditioner is an important area for future research.

REFERENCES

[1] B. BOLLOBÁS, *Graph Theory*, Springer-Verlag, New York, 1979.
[2] D. BRÉLAZ, *New methods to color the vertices of a graph*, Comm. ACM, 22 (1979), pp. 251–256.

[3] T. F. Chan and H. C. Elman, *Fourier analysis of iterative methods for elliptic boundary value problems*, SIAM Review, 31 (1989), pp. 20–49.

[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] I. S. Duff and G. A. Meurant, *The effect of ordering on preconditioned conjugate gradients*, BIT, 29 (1989), pp. 635–657.

[6] T. H. Dunigan, *Performance of the Intel iPSC/860 Hypercube*, ORNL/TM 11491, Oak Ridge National Laboratory, 1990.

[7] H. C. Elman and E. Agrón, *Ordering techniques for the preconditioned conjugate gradient method on parallel computers*, Computer Physics Communications, 53 (1989), pp. 253–269.

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

[9] A. George and J. W.-H. Liu, *Computer Solution of Large Sparse Positive Definite Systems*, Prentice-Hall, Englewood Cliffs, N.J., 1981.

[10] G. H. Golub and C. F. V. Loan, *Matrix Computations*, The Johns Hopkins University Press, Baltimore, 1983.

[11] 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.

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

[13] M. R. Hestenes and E. Stiefel, *Methods of conjugate gradients for solving linear systems*, Journal of Research of the National Bureau of Standards, 49 (1952), pp. 409–436.

[14] 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.

[15] M. T. Jones and P. E. Plassmann, *Parallel iterative solution of sparse systems using orderings from graph coloring heuristics*, Preprint MCS-P198-1290, Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, Ill., 1990.

[16] ——, *A parallel graph coloring heuristic*, Preprint MCS-P246-0691, Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, Ill., 1991.

[17] C.-C. J. Kuo and B. C. Levy, *A two-level four-color SOR method*, SIAM Journal on Numerical Analysis, 26 (1989), pp. 129–151.

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

[19] J. Meijerink and H. van der Vorst, *An iterative solution method for linear systems of which the coefficient matrix is a symmetric M-matrix*, Mathematics of Computation, 31 (1977), pp. 148–162.

[20] J. M. Ortega, *Orderings for conjugate gradient preconditioning*, SIAM Journal on Optimization, 1 (1991), pp. 565–582.

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

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