ORNL/TM-13729

ornl

OAK RIDGE NATIONAL LABORATORY

LOCKHEED MARTIN

Minimal Orderings Revisited

Barry W. Peyton

MANAGED AND OPERATED BY
LOCKHEED MARTIN ENERGY RESEARCH CORPORATION
FOR THE UNITED STATES
DEPARTMENT OF ENERGY

ORNL-27 (3-96)

DISCLAIMER

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

ORNL/TM-13729

Computer Science and Mathematics Division

MINIMAL ORDERINGS REVISITED

Barry W. Peyton

Computational Mathematics and Statistics Section
Oak Ridge National Laboratory
P. O. Box 2008, Bldg. 6012
Oak Ridge, TN 37831-6367

Date Published: July 1999

Research supported by the Applied Mathematical Sciences subprogram of the Office of Energy Research, U.S. Department of Energy

Prepared by
OAK RIDGE NATIONAL LABORATORY
Oak Ridge, Tennessee 37831-6285
managed by
LOCKHEED MARTIN ENERGY RESEARCH CORP.
for the
U.S. DEPARTMENT OF ENERGY
under contract DE-AC05-96OR22464

Contents

List of Tables

Acknowledgements

The author thanks Bengt Aspvall and Jean Blair and her family for their gracious hospitality during a visit to Bergen, Norway a few years ago. In the stimulating environment provided during that visit, the ideas behind this paper began to come together.

Abstract

When minimum orderings proved too difficult to deal with, Rose, Tarjan, and Leuker instead studied minimal orderings and how to compute them (Algorithmic aspects of vertex elimination on graphs, *SIAM J. Comput.*, 5:266–283, 1976). This paper introduces an algorithm that is capable of computing much better minimal orderings much more efficiently than the algorithm in Rose et al. The new insight is a way to use certain structures and concepts from modern sparse Cholesky solvers to re-express one of the basic results in Rose et al. The new algorithm begins with any initial ordering and then refines it until a minimal ordering is obtained. It is simple to obtain high-quality low-cost minimal orderings by using fill-reducing heuristic orderings as initial orderings for the algorithm. We examine several such initial orderings in some detail.

1. Introduction

Let A be an $n \times n$ symmetric positive definite matrix, let P be an $n \times n$ permutation matrix, and let L be the Cholesky factor of PAP^T. A *minimum ordering* is any ordering P that minimizes the number of nonzero entries in L, subject to the usual assumption that no lucky cancellation occurs. Rose, Tarjan, and Leuker [14] conjectured that the problem of computing a minimum ordering is NP-complete, and later Yannakakis [17] verified this conjecture. Rose et al. [14] turned their attention instead to the easier problem of computing a minimal ordering. This paper revisits this problem: we show how any initial ordering can be refined to obtain a minimal ordering whose fill is a subset of the initial ordering's fill.

Following Rose et al., we use graphs to define minimal orderings. Let $G = (V, E)$ be the graph of PAP^T; that is, $V = \{1, 2, \ldots, n\}$, and an undirected edge $\{i, j\}$, $i \neq j$, belongs to E if and only if the (i, j)-entry of PAP^T is not zero. (Only the labeling of the vertices varies as P varies; the structure of the graph and of course the number of edges, $e = |E|$, remain the same.) Define G^+ to be the *fill graph* associated with PAP^T; that is, G^+ is the graph of $L+L^T$ under the usual assumption that no lucky cancellation occurs. Note that $G^+ = (V, E \cup F)$, where F is composed of the fill edges created by the elimination process; hence, G^+ is a *supergraph* of G. A graph is *chordal* if every cycle of length greater than three has a chord, that is, an edge joining two nonadjacent vertices in the cycle. It is well known [12, 13] that G^+ is a chordal supergraph of G. A minimum ordering P minimizes the number of edges in G^+ over all orderings; in this case, G^+ is a *minimum chordal supergraph* of G. For a *minimal chordal supergraph* $G^* = (V, E \cup F^*)$ of G, every supergraph $G' = (V, E \cup F')$ of G such that $F' \subset F^*$ is not chordal. A *minimal ordering* produces a fill graph G^+ that is a minimal chordal supergraph of G.

Broadly speaking, the primary goal of this paper is to carry a few of the key insights in Rose et al. [14] back into the sparse factorization setting in a fruitful way. We use a key result in Rose et al. [14] to lay the groundwork for a new minimal ordering algorithm. Beginning with any initial ordering, the new algorithm generates a sequence of reorderings, each removing additional fill from the current fill graph, until a minimal chordal supergraph, and hence a minimal ordering, is obtained. Several familiar concepts and algorithms from sparse Cholesky factorizations are used to formulate and implement the algorithm; these include elimination trees, supernodes, supernodal elimination trees, topological orderings, the minimum degree algorithm, and column counts. Although we assume some familiarity with these concepts and algorithms, we also include references and a minimum of background material where needed.

Both the LEX M algorithm of Rose et al. [14] and an algorithm of Ohtsuki [11] compute a minimal ordering in $O(ne)$ time. Partly because of the use of quotient graphs and minimum degree in the new algorithm, the new algorithm's time complexity

remains unknown; consequently, we rely exclusively on empirical testing to evaluate the algorithm's time efficiency. The first tests we conduct in Section 5 show that LEX M does not measure up to the new minimal ordering algorithm in either ordering quality or ordering time. The same should hold true for the algorithm of Ohtsuki. These first tests use minimum degree with internal degree and no multiple elimination (MDint) to produce the initial orderings. These initial orderings are often minimal or very close to minimal; often the minimal ordering algorithm serves merely as a relatively cheap means to verify that the initial ordering is minimal.

Many of the tests in Section 5 with other initial orderings produce similar results, though there are some differences worth observing. The orderings tested include random (Ran) orderings, METIS nested dissection (ND) orderings [6], multisection (MS) orderings [2, 5] based on METIS ND, and minimum degree with external degree and no multiple elimination (MDext) [7]. Roughly speaking, most of the MDint orderings are minimal, the MDext orderings are very close to minimal, and the MS orderings are close to minimal. The ND orderings are not as close to minimal as the other ordering heuristics, and some of the ND orderings are far from minimal. Finally, the random orderings are extremely far from minimal, but the minimal orderings obtained from random initial orderings are poor fill-reducing orderings and are expensive to compute.

The following gives an outline of this paper. Section 2 presents background material from Rose et al. [14] and from the area of sparse Cholesky factorization. Section 3 presents the main result, which uses some concepts and tools from sparse Cholesky factorization to recast one of the insights in Rose et al. In Section 4 the main result forms the basis for a new minimal ordering algorithm. Section 5 compares the new algorithm with the LEX M algorithm and experiments with various initial orderings. Section 6 summarizes and adds a few concluding remarks.

2. Background

In Section 2.1, we state a scheme for obtaining a minimal chordal supergraph from a nonminimal chordal supergraph; the scheme is implicit in a result of Rose et al. [14]. With further development and refinement, this scheme will become an algorithm for computing minimal orderings. Section 2.1 also states another key result from [14]. In Section 2.2, we give some concepts and tools from sparse Cholesky factorization that will be used to develop the new minimal ordering algorithm.

2.1. Computing minimal chordal supergraphs

Let G^* be a chordal supergraph of the graph G. A *candidate edge* $\{u, v\}$ is any fill edge such that G^* remains chordal after $\{u, v\}$ has been removed from the graph. Rose et al. [14] showed that every nonminimal chordal supergraph has a candidate edge. As an immediate consequence of this result and the definition of candidate edges, we have

the following proposition.

Proposition 1 (Rose et al. [14]). *A chordal supergraph is minimal if and only if it has no candidate edges.*

As an immediate consequence of Proposition 1, the scheme shown in Figure 1 will produce a minimal chordal supergraph. The set of candidate edges changes as edges are removed from the graph: some noncandidate fill edges may become candidate edges; some candidate edges may cease to be candidate edges.

```
Input: a chordal supergraph G* of the graph G.
while there is a candidate edge in G* do
    remove a candidate edge from the graph G*;
endwhile;
```

Figure 1: Scheme for generating a minimal chordal supergraph.

The following proposition from Rose et al. [14] characterizes the candidate edges.

Proposition 2 (Rose et al. [14]). *Let G^*, which is $(V, E \cup F)$, be a chordal supergraph of G, which is (V, E). A fill edge $\{u, v\} \in F$ of G^* is not a candidate edge if and only if there exist two vertices a and z such that a and z are both adjacent to u and v in G^*, but not adjacent to one another in G^*.*

In Section 3, we use concepts and tools from sparse Cholesky factorization to recast this characterization in the case where G^* is the fill graph G^+ associated with the graph G of PAP^T.

2.2. Concepts and tools from sparse factorization

The fill graph G^+ is obtained from the graph G of PAP^T by an elimination process that models the factorization elimination process. Let G_k be the graph obtained from G by adding every edge needed to make the vertices adjacent to k ($\text{adj}_G[k]$) a clique and then by eliminating k and the edges incident upon k. The elimination process replaces G with G_1, G with G_2, G with G_3, and so on, until it finally replaces G with G_{n-1}. The fill graph has the edges belonging to the original graph G along with the fill edges generated by the elimination process. We also define an elimination graph G_X for an arbitrary subset of vertices X. This graph is obtained by using the elimination process to eliminate in any order the vertices of X (and only the vertices of X). The resulting graph is independent of the order in which the vertices of X are removed.

For a vertex k of a graph G', let $\text{madj}_{G'}[k]$ be the neighbors of k in G' that are numbered higher than k. The parent function of the *elimination tree* (or forest) associated

with a fill graph G^+ is defined as follows: if $\text{madj}_{G^+}[k]$ is empty, then the parent of k is null and k is a root in the forest; otherwise, the parent of k is the lowest numbered member of $\text{madj}_{G^+}[k]$. The following fact [9] proves useful later on. Let $c_1, c_2, \ldots, c_t$ be the children of a vertex p in the elimination tree. Then

$$\{p\} \cup \text{madj}_{G^+}[p] = \left(\bigcup_{i=1}^{t} \text{madj}_{G^+}[c_i] \right) \cup \{p\} \cup \text{madj}_G[p]. \tag{1}$$

Note that Eq. (1) holds only for fill graphs G^+ and not for arbitrary chordal supergraphs.

A vertex a is an *ancestor* of vertex d (and d is a *descendant* of a) if a lies on the path from d to the root of d's tree in the elimination forest. The vertex a is a *proper ancestor* of vertex d (and d is a *proper descendant* of a) if a is distinct from d and an ancestor of d.

Supernodes have become a familiar tool in various computations associated with sparse factorizations. The fundamental supernode partition is commonly used and has received some attention. Liu, Ng, and Peyton [10] give an algorithm that computes the fundamental supernode partition in $O(n + e)$ time. The supernode partition defined here is similar to supernode partitions used in practice, but it does not consist of fundamental supernodes nor does it define the maximal cliques of the chordal graph. This departure from the usual supernode partitions is motivated entirely by the problem at hand; the reason for it will become apparent in the proof of our main result, presented in Section 3.

Definition 1. *Let G^+ be the fill graph associated with the graph G of PAP^T. We define a supernode partition as follows: a child-parent pair c and p in the elimination tree belong to the same supernode if and only if c is the only child of p for which* $\text{madj}_{G^+}[c] = \{p\} \cup \text{madj}_{G^+}[p]$.

For a given elimination tree, this supernode partition is unique. Note that each supernode is a path in the elimination tree from a lowest vertex to an ancestor of the lowest vertex. All references to supernodes in this paper are to those defined by Definition 1.

We will also need the *supernodal elimination tree* associated with this supernode partition. Each supernode S is a vertex in the supernodal elimination tree. Supernode P is the parent of supernode C if the parent (in the elimination tree) of the "top" vertex in C is a vertex in P. Supernode R is a root if the top vertex in R is a root vertex in the elimination tree (or forest).

Let S be a supernode. A vertex u is a *proper descendant* of S if $u \notin S$ and u is a descendant of some vertex in S in the elimination tree. Let $T[S]$ be the subtree of the elimination tree rooted at S; that is, $T[S]$ includes the vertices of S and all vertices that are proper descendants of S in the elimination tree. Let $D[S] := T[S] \setminus S$ so that it contains precisely the proper descendants of S in the elimination tree. Note that a

supernode S' is a proper descendant of a supernode S in the supernodal elimination tree if and only if the vertices of S' are proper descendants of S in the elimination tree.

3. A characterization of candidate edges

We now state and prove an alternative characterization of the candidate edges, which is the basis for the algorithm we develop in Section 4.

Proposition 3. *Let G^+ be the fill graph associated with the graph G of PAP^T. Let S be a supernode in the supernode partition given by Definition 1. Assume that (1) $u \in S$, (2) $u < v$ in the elimination order, and (3) $\{u, v\}$ is a fill edge. We then have the following: $\{u, v\}$ is a candidate edge if and only if $\{u, v\}$ is not an edge in the elimination graph $G_{D[S]}$.*

Proof: Throughout the proof, let f be the first vertex (i.e., the lowest numbered vertex) in supernode S.

Assume that $\{u, v\}$ is not a candidate edge. By Proposition 2 there exist then two vertices a and z that are adjacent to u and v in G^+ but not adjacent to each other in G^+. Now, by assumption $u \in S$, and f is the first vertex in supernode S. It follows that $\mathrm{madj}_{G^+}[u] \subseteq \{f\} \cup \mathrm{madj}_{G^+}[f]$. The lower numbered neighbors of u in G^+ must be descendants of u in the elimination tree [9, 16]. Therefore, these lower numbered neighbors belong to $T[S]$, which is $S \cup D[S]$. Since $S \subseteq \{f\} \cup \mathrm{madj}_{G^+}[f]$, we have $\mathrm{adj}_{G^+}[u] \subseteq D[S] \cup \{f\} \cup \mathrm{madj}_{G^+}[f]$. It then follows that both a and z belong to $D[S] \cup \{f\} \cup \mathrm{madj}_{G^+}[f]$. Since $\{f\} \cup \mathrm{madj}_{G^+}[f]$ is a clique in G^+, at least one of the two nonadjacent vertices a and z belongs to $D[S]$. It follows that $\{u, v\}$ is an edge in the elimination graph $G_{D[S]}$.

To prove the other direction, assume that $\{u, v\}$ is a candidate edge and that $\{u, v\}$ is a fill edge in the elimination graph $G_{D[S]}$. It suffices to derive a contradiction from these assumptions.

Since $\{u, v\}$ is a fill edge in the elimination graph $G_{D[S]}$, there exists a vertex $a \in D[S]$ that is adjacent to both u and v in G^+. For the following reasons we may assume without loss of generality that a is a child of a vertex in S. Any descendant vertex d of S has as one of its ancestors a vertex c that is a child of some vertex in S; moreover, for any child c of a vertex in S and any descendant d of c, we have

$$\mathrm{madj}_{G^+}[d] \cap (\{f\} \cup \mathrm{madj}_{G^+}[f]) \subseteq \mathrm{madj}_{G^+}[c] \cap (\{f\} \cup \mathrm{madj}_{G^+}[f]).$$

Now, since $\{u, v\}$ is a candidate edge by assumption, it follows that any pair of vertices adjacent to both u and v are adjacent to one another. Since u and v are both adjacent in G^+ to every vertex in $\{f\} \cup \mathrm{madj}_{G^+}[f] \setminus \{u, v\}$, it follows that a is adjacent in G^+ to every vertex in $\{f\} \cup \mathrm{madj}_{G^+}[f]$, so we can write $\{f\} \cup \mathrm{madj}_{G^+}[f] \subseteq \mathrm{madj}_{G^+}[a]$.

It moreover follows that a is a child of f; for if it were a child of some other vertex of S then it could not be adjacent in G^+ to f because f would then be neither an ancestor nor a descendant of a. Furthermore, $\mathrm{madj}_{G^+}[a] = \{f\} \cup \mathrm{madj}_{G^+}[f]$, because by (1), $\mathrm{madj}_{G^+}[a] \subseteq \{f\} \cup \mathrm{madj}_{G^+}[f]$.

Now, supernode S begins at vertex f; that is, no descendants of f belong to S. Consequently, existence of the child a of f for which $\mathrm{madj}_{G^+}[a] = \{f\} \cup \mathrm{madj}_{G^+}[f]$ implies the existence of another child z of f for which $\mathrm{madj}_{G^+}[z] = \{f\} \cup \mathrm{madj}_{G^+}[f]$; were there no such vertex z, vertex a would have been incorporated into the supernode S and S would not begin at f. Vertices a and z clearly are adjacent to u and v in G^+, but they are not adjacent to each other because they are siblings in the elimination tree. From Proposition 2 it follows that $\{u, v\}$ *is not* a candidate edge, contrary to our assumption that it *is* a candidate edge. The result follows from this contradiction. ■

4. A minimal ordering algorithm

Let G^+ be the fill graph associated with the graph G of PAP^T, and consider the supernode partition given by Definition 1. In the following definition, we partition the candidate edges among the supernodes. The vertices of S are listed in elimination order.

Definition 2. *Let $S = \{f = u_1, u_2, \ldots, u_r\}$ be a supernode in the supernode partition given by Definition 1. The candidate edges of S in G^+ include every candidate edge $\{u, v\}$ for which $u \in S$ and $v \geq f$.*

Proposition 3 says that the candidate edges of S are missing from the elimination graph $G_{D[S]}$. If S has any candidate edges, then some of them can be removed simply by reordering the vertices of S; what follows in Subsections 4.1 and 4.2 expands on and justifies the preceding statement. The algorithm is presented in Subsection 4.3.

4.1. Candidate edges and degrees in an elimination graph

First, we need two more definitions. Define the *degree* of a vertex v in a graph G' to be the number of neighbors of v in G', that is, $\deg_{G'}(v) := |\mathrm{adj}_{G'}[v]|$. Two vertices v and w in a graph G' are said to be *indistinguishable* if

$$\{v\} \cup \mathrm{adj}_{G'}[v] = \{w\} \cup \mathrm{adj}_{G'}[w].$$

Let $S = \{f = u_1, u_2, \ldots, u_r\}$ be a supernode in the supernode partition given by Definition 1. Again, the vertices of S are listed in elimination order. Consider the elimination graph $G_{D[S]}$. Recall that for each vertex in S the set $D[S]$ contains every descendant in the elimination tree that does not belong to S. Since vertex f has no proper descendant that belongs to S, every proper descendant of f belongs to $D[S]$.

Note also that none of the ancestors of f are included in $D[S]$. It follows [9] that

$$\{f\} \cup \mathrm{adj}_{G_{D[S]}}[f] = \{f\} \cup \mathrm{madj}_{G^+}[f]. \tag{2}$$

Now, consider the elimination graph obtained by eliminating f from $G_{D[S]}$, that is, the elimination graph G_X where $X = \{f\} \cup D[S]$. From (2) it follows that $\mathrm{madj}_{G^+}[f]$ is a clique in G_X. Since $\{u_2, \ldots, u_r\} \subseteq \mathrm{madj}_{G^+}[f]$, it follows that for i, $2 \leq i \leq r$,

$$\{u_i\} \cup \mathrm{adj}_{G_X}[u_i] \supseteq \mathrm{madj}_{G^+}[f]. \tag{3}$$

From (3) and the fact that S is a supernode in G^+, it follows that for i, $2 \leq i \leq r$,

$$\{u_i\} \cup \mathrm{adj}_{G_X}[u_i] = \mathrm{madj}_{G^+}[f]. \tag{4}$$

In other words, the vertices $u_2, \ldots, u_r$ become indistinguishable from one another after f is removed from $G_{D[S]}$ to obtain G_X, as described previously. But the vertices $f, u_2, \ldots, u_r$ are not necessarily indistinguishable from one another in $G_{D[S]}$, and this is the key to the algorithm.

Now we shift the focus back to $G_{D[S]}$. It follows directly from (4) that for every i, $2 \leq i \leq r$, we have

$$\{u_i\} \cup \mathrm{adj}_{G_{D[S]}}[u_i] \subseteq \{f\} \cup \mathrm{madj}_{G^+}[f]. \tag{5}$$

Note that since $\{f\} \cup \mathrm{madj}_{G^+}[f]$ is a clique in G^+, any pair of vertices from this clique that is not joined by an edge in the original graph is joined by a fill edge in G^+. From (2) and Proposition 3, it follows that S has no candidate edges incident upon f. It follows from (5), Proposition 3, and the preceding comment on fill edges that the candidate edges of S incident upon u_i $(2 \leq i \leq r)$ are precisely those joining u_i to any vertex in

$$(\{f\} \cup \mathrm{madj}_{G^+}[f]) \setminus (\{u_i\} \cup \mathrm{adj}_{G_{D[S]}}[u_i]). \tag{6}$$

It follows from (2) and the preceding statement that the number of candidate edges of S incident upon u_i is

$$\deg_{G_{D[S]}}(f) - \deg_{G_{D[S]}}(u_i). \tag{7}$$

4.2. Using minimum degree to remove candidate edges

From the last statement of the preceding subsection, it follows that a vertex $u_i \in S$ with minimum degree $\deg_{G_{D[S]}}(u_i)$ has the most candidate edges of S incident upon it in G^+. Select such a vertex u_i to be eliminated from $G_{D[S]}$. The set $\mathrm{adj}_{G_{D[S]}}[u_i]$ will be the monotone adjacency set of u_i no matter how the elimination process is completed. The candidate edges of S incident upon u_i in G^+ join u_i with the vertices of

$$(\{f\} \cup \mathrm{madj}_{G^+}[f]) \setminus (\{u_i\} \cup \mathrm{adj}_{G_{D[S]}}[u_i]);$$

such candidate edges exist if and only if $\deg_{G_{D[S]}}(u_i) < \deg_{G_{D[S]}}(f)$. Since the set $\mathrm{adj}_{G_{D[S]}}[u_i]$ will be the monotone adjacency set of u_i, it follows that all the candidate edges of S incident upon u_i do not appear in the new elimination graph; moreover, these candidate edges are the only edges to disappear from the monotone adjacency set of u_i. Note that no new edges that are not in G^+ are introduced since

$$\mathrm{adj}_{G_{D[S]}}[u_i] \subseteq \{f\} \cup \mathrm{madj}_{G^+}[f] \setminus \{u_i\}.$$

We then repeat the process. Let $X = \{u_i\} \cup D[S]$, and consider the elimination graph G_X. Choose from the uneliminated vertices of S a vertex u_j with minimum degree $\deg_{G_X}(u_j)$. Vertex u_j has the most candidate edges of S incident upon it in G^+ that were not filled in by the previous elimination of u_i. The vertex u_j will be eliminated from G_X; hence, the set $\mathrm{adj}_{G_X}[u_j]$ will be the monotone adjacency set of u_j no matter how the elimination process is completed. Any nonfilled candidate edges of S incident upon u_j in G^+ join u_j with the vertices of

$$(\{f\} \cup \mathrm{madj}_{G_{D[S]}}[f]) \setminus (\{u_i, u_j\} \cup \mathrm{adj}_{G_X}[u_j]);$$

such candidate edges exist if and only if $\deg_{G_X}(u_j) < \deg_{G_X}(f)$. Since the set $\mathrm{adj}_{G_X}[u_j]$ will be the monotone adjacency set of u_j in the new elimination graph, it follows that all the nonfilled candidate edges of S incident upon u_i do not appear in the new elimination graph; moreover, these candidate edges are the only edges to disappear from the monotone adjacency set. Note that no new edges that are not in G^+ are introduced since

$$\mathrm{adj}_{G_X}[u_j] \subseteq \{f\} \cup \mathrm{madj}_{G^+}[f] \setminus \{u_i, u_j\}.$$

We continue this process until all the vertices of S are removed from the original elimination graph $G_{D[S]}$. If S has any candidate edges at all, then some are removed by applying the minimum degree ordering heuristic to the vertices of S in the elimination graph $G_{D[S]}$, as we did previously. Moreover, only candidate edges of S are removed by this process. If S has no candidate edges, then the vertices of S are indistinguishable from one another in $G_{D[S]}$, and applying the minimum degree ordering heuristic to the vertices of S in the elimination graph $G_{D[S]}$ produces an arbitrary ordering of S and does not change the fill. In this case, the sequence of degrees is $\deg_{G_{D[S]}}(f)$, $\deg_{G_{D[S]}}(f)-1, \ldots,$ and $\deg_{G_{D[S]}}(f)-r+1$; moreover, if $u_{i_1}, u_{i_2}, \ldots, u_{i_r}$ is any ordering of S, then the monotone adjacency sets are $\{f\} \cup \mathrm{madj}_{G^+}[f] \setminus \{u_{i_1}\}$, $\{f\} \cup \mathrm{madj}_{G^+}[f] \setminus \{u_{i_1}, u_{i_2}\}$, $\ldots,$ and $\{f\} \cup \mathrm{madj}_{G^+}[f] \setminus \{u_{i_1}, u_{i_2}, \ldots, u_{i_r}\}$.

4.3. The algorithm

The algorithm for computing a minimal ordering, including some implementation details, appears in Figure 2. The algorithm requires an initial ordering; it will work with

any initial ordering, including a random one. The algorithm repeats the major step until there is no reduction in the factor nonzero count (i.e., the fill edges). A major step breaks into two parts:

1. symbolic processing using the current ordering, and
2. a block elimination process with minimum degree refinement on each block (supernode) to obtain a new ordering.

```
[Initial ordering: can be any ordering]

[Repeat ordering refinement step until no progress is encountered]
until the total factor nonzero count is not reduced do

    [Symbolic preprocessing using current ordering]
    Compute elimination tree and postorder it [9];
    Compute column factor nonzero counts [4];
    Compute the supernodes (Definition 1);
    Compute supernodal elimination tree and a topological ordering
    of this tree;

    [block elimination process in block topological order and with
    minimum degree refinement to obtain new refinement of old ordering]
    Begin elimination process with original graph;
    for each supernode S (in topological order) do
        until all vertices in S have been eliminated do
            Select a vertex v ∈ S whose internal degree in the
            current elimination graph is minimum among the
            uneliminated vertices in S;
            Eliminate v and form the quotient graph
            representation of the new elimination graph;
        end until;
    end for;

    Replace old ordering with new ordering;
    [Here we check new total factor nonzero count against old]
end until;
```

Figure 2: Algorithm for computing a minimal ordering, including some implementation details.

During the symbolic part of a major step, the algorithm first computes the elimination tree, and then postorders it. We use the fast algorithm in [7] to compute the elimination tree; the postordering is needed to compute the column factor nonzero counts. (Column factor nonzero counts refer to the number of nonzero entries in each column of

the Cholesky factor under the current postordering of the elimination tree.) Computing the column factor nonzero counts is achieved using the fast algorithm in [4]. With the elimination tree and column counts in hand, it is trivial to compute the supernodes of Definition 1 and the associated supernodal elimination tree. Finally, the algorithm computes a topological ordering of the supernodal elimination tree; this also is trivial.

During the elimination part of a major step, the algorithm processes the supernodes in the given topological order of the current supernodal elimination tree. The elimination process is a block elimination process; for any supernode S, the algorithm uses the minimum degree algorithm to eliminate the vertices of S together, and only after it has removed in the same fashion, supernode by supernode, the descendant vertices of S, $D[S]$. The topological ordering also ensures that no vertex from an ancestor supernode is removed before the vertices of the supernode and its descendants. In short, the analysis presented in Subsections 4.1 and 4.2 applies directly to the elimination graphs generated by the algorithm. That is, any supernode S that has candidate edges will have some of the candidate edges removed with no edges added beyond the fill generated by the current ordering. Moreover, any edges removed are candidate edges.

The elimination process generates a new ordering. During the elimination process, the algorithm accumulates the amount of fill incurred by this ordering, and when the elimination process is finished, the algorithm compares it with the amount of fill incurred by the old ordering. If there is no reduction, then the algorithm stops. From the argument in the previous paragraph, a major step of the algorithm removes only candidate edges and removes them if and only if there exist such edges in the current fill graph. Since there are a finite number of edges, the algorithm must terminate at some point, and clearly it will be at the point where there are no candidate edges in the old fill graph and no candidate edges in the new fill graph. It follows from Proposition 1 that the algorithm terminates with a minimal ordering and minimal chordal supergraph. The algorithm is, indeed, merely an elaboration of the scheme shown in Figure 1.

Note that it is important to use internal degree rather than external degree during the block elimination process. Internal degree gives priority to vertices incident upon the most candidate edges, as desired. External degree may give priority to a vertex incident upon no candidate edges even though there are vertices available that are incident upon candidate edges. Consequently, external degree could fail to detect candidate edges for a supernode that has some, while internal degree is sure to detect and remove some candidate edges from any supernode that has some.

Note also that we have adapted a minimum degree code to perform the supernode-by-supernode elimination process. The adapted code inherits several of the standard enhancements that have been incorporated into such codes [3, 7]; these include mass elimination, indistinguishable nodes, incomplete degree update, and the generalized element storage scheme. Because any two vertices from distinct supernodes must be treated as separate vertices, some opportunities for mass elimination or detecting in-

distinguishability must be passed over. On the other hand, because the elimination process needs to know degrees of vertices in one supernode at a time, we can greatly reduce the number of degree calculations needed. Our timings indicate that the gains from the latter typically far outweigh the costs from the former.

5. Test results

We wrote Fortran 77 implementations of the new minimal ordering algorithm described in the previous section and the LEX M algorithm described on pages 273 and 280–281 of Rose et al. [14]. We used Fortran compiler `F77` with compiler optimization level `-O`, and we ran the tests on a SUN Sparc 20 workstation (model 41).

The primary purpose of Subsection 5.1 is simply empirical proof of concept for the new algorithm. To achieve this goal it suffices to show that, in practice, the new algorithm can produce high-quality minimal orderings very efficiently compared with the LEX M algorithm. Since we are interested in obtaining high-quality minimal orderings as efficiently as possible, we chose minimum degree orderings to be the initial orderings in Subsection 5.1. To be consistent with the later use of internal degree in the refinement step, we use internal degree rather than the superior external degree [7] to compute the initial ordering, too. For the same reason we do not use multiple elimination; we use minimum degree with internal degree and no multiple elimination (MDint). While establishing proof of concept, we will also observe that MDint often computes minimal orderings.

We examine random (Ran) intitial orderings in Subsection 5.2, minimum degree initial orderings with external degree and no multiple elimination (MDext) in Subsection 5.3, METIS nested dissection (ND) initial orderings in Subsection 5.4, and multisection (MS) initial orderings based on METIS ND in Subsection 5.5.

5.1. Proof of concept

The large run times of LEX M limited us to relatively small matrices for our test runs. Despite their limited size, the test matrices suffice for our purposes. For more accurate comparisons we computed ten random permutations for each graph, we computed ten permuted versions of the adjacency structure for each graph, and we ran both algorithms on each of the ten permuted adjacency structures for each of the matrices. We then averaged the reported statistics over the ten runs for each matrix. We report factor floating point operations rather than fill because comparing factor flops usually emphasizes the differences between orderings more than comparing factor nonzero counts. Table 1 reports average factor flops for the initial MDint ordering, the final ordering obtained by the new algorithm, and the LEX M ordering. Table 2 reports average run times for both algorithms and the average number of major iterations taken by the new algorithm.

Matrix	MDint/new-minimal		LEX M
	Initial MDint	Final	
DIS060	11,647,635	11,647,635	20,493,469
DIS090	49,259,575	49,259,575	114,737,552
DIS120	132,932,902	132,932,902	345,508,373
NASA1824	5,587,100	5,587,100	49,813,698
NASA2910	32,173,189	32,173,189	96,484,650
NASA4704	44,214,573	44,214,573	341,965,690
SPA060	18,833,613	18,833,613	35,749,867
SPA090	81,138,522	81,138,522	186,792,889
SPA120	213,760,890	213,760,890	617,861,622
BCSSTK13	68,694,616	68,694,616	144,702,445
BCSSTK14	10,432,368	10,432,368	28,393,024
BCSSTK15	193,568,418	193,568,418	481,766,496
BCSSTK16	169,108,792	169,108,792	145,053,771
BCSSTK17	220,082,381	219,444,619	637,065,955
BCSSTK18	155,701,425	155,701,425	2,012,990,730
BCSSTK19	121,633	117,440	126,733
BCSSTK23	160,449,169	160,449,169	304,238,013
BCSSTK24	40,172,956	40,172,956	107,450,852
BCSSTK25	399,921,493	399,921,493	525,698,984
BCSSTK26	1,646,503	1,646,417	14,716,146

Table 1: Average factor floating point operations associated with MDint/new-minimal and with LEX M. Averages are taken over ten runs with randomly permuted adjacency structures.

Matrix	MDint/new-minimal				LEX M
	MDint time	Major iter.	Minimal time	Total time	time
DIS060	0.23	1.0	0.21	0.44	22.30
DIS090	0.80	1.0	0.54	1.34	134.27
DIS120	1.44	1.0	1.07	2.51	440.42
NASA1824	0.08	1.0	0.08	0.16	6.09
NASA2910	0.22	1.0	0.28	0.50	31.02
NASA4704	0.28	1.0	0.24	0.52	47.88
SPA060	0.18	1.0	0.15	0.33	18.89
SPA090	0.62	1.0	0.48	1.10	126.72
SPA120	1.17	1.0	0.92	2.09	452.17
BCSSTK13	0.24	1.0	0.15	0.39	11.88
BCSSTK14	0.10	1.0	0.09	0.19	7.87
BCSSTK15	0.54	1.0	0.28	0.82	37.15
BCSSTK16	0.40	1.0	0.43	0.83	90.07
BCSSTK17	1.02	2.1	1.80	2.82	331.14
BCSSTK18	1.15	1.0	0.70	1.85	138.01
BCSSTK19	0.02	8.1	0.17	0.19	0.37
BCSSTK23	0.53	1.0	0.18	0.71	12.84
BCSSTK24	0.13	1.0	0.19	0.32	37.98
BCSSTK25	3.35	1.0	1.58	4.93	565.62
BCSSTK26	0.11	1.3	0.12	0.23	4.43

Table 2: Average ordering times (in CPU seconds) and average number of major iterations for MDint/new-minimal and for LEX M. Averages are taken over ten runs with randomly permuted adjacency structures.

A "1.0" in column three of Table 2 means that for all ten permutations of the adjacency structure the initial MDint ordering is minimal. Consequently, for all ten permutations of the adjacency structure the new minimal ordering algorithm takes one major step. For 17 of the 20 problems, the initial MDint ordering is minimal for all ten permutations of the adjacency structure. For these problems, the initial factor flops and the final factor flops are the same. For BCSSTK17 and BCSSTK26, there is, on average, a small change: less than a 0.5% reduction in factor flops. The largest change is for the very small problem BCSSTK19, but it is still only a 3.5% reduction in factor flops. Clearly, the new minimal ordering algorithm serves most often merely to verify that the initial MDint ordering is minimal; in the other three cases, it trims away fill (and factor flops) from nearly minimal MDint orderings until minimality is achieved. In short, the MDint ordering heuristic comes very close to being a minimal ordering algorithm in our tests.

Also note in Table 1 that the MDint/new-minimal orderings consistently cost far fewer factor floating point operations than the LEX M orderings. For only one problem, BCSSTK16, does LEX M outperform MDint/new-minimal in this regard. Typically, LEX M requires anywhere from a factor of two to a factor of four more factor flops than MDint/new-minimal; sometimes LEX M requires a factor of eight or nine more factor flops than MDint/new-minimal. This superiority of MDint/new-minimal is not surprising; LEX M, which is a breadth-first search ordering, has much in common with bandwidth and profile reducing orderings, and hence a good general sparse ordering like MDint is naturally expected to reduce factor flops better than LEX M. The tests serve to confirm this expectation.

In Table 2, the total run-times of the MDint/new-minimal algorithm are, with one exception (BCSSTK19), a very small fraction of the corresponding run times of the LEX M algorithm. This is not surprising; the time complexity for LEX M is $O(ne)$, and the $O(ne)$ time complexity is fully realized by the implementation [14]. Although the time complexity of the minimum degree heuristic is unknown, the empirical efficiency of modern implementations of this heuristic is well established [3, 7]; it would be somewhat surprising to see minimum degree ordering times exceed any significant fraction of the corresponding LEX M ordering times.

There are known problems with the time efficiency of the minimum degree algorithm. For example, it is well known that a dense or near-dense row in the matrix seriously degrades the time efficiency of conventional implementations. Such rows are rare in practice and rare in most test collections; we apparently included no test problems on which the minimum degree algorithm runs very inefficiently.

In the cases where the new minimal ordering algorithm merely confirms the minimality of the initial MDint ordering, the time to confirm minimality is usually smaller than, but comparable to, the MDint ordering time. Exceptions include BCSSTK23 and BCSSTK25, for which the time to confirm minimality is unusually small, and

Matrix	Ran/new-minimal		LEX M
	Initial Ran	Final	
DIS060	4,757,731,242	37,682,529	20,493,469
DIS090	54,861,712,933	220,798,340	114,737,552
DIS120	312,439,901,383	756,179,560	345,508,373
NASA1824	1,062,532,716	42,332,133	49,813,698
NASA2910	6,447,802,052	142,705,755	96,484,650
NASA4704	18,158,402,295	384,032,920	341,965,690
SPA060	4,963,022,985	62,885,285	35,749,867
SPA090	56,608,430,168	339,173,406	186,792,889
SPA120	321,186,967,378	1,112,385,506	617,861,622
BCSSTK13	1,870,073,867	277,115,570	144,702,445
BCSSTK14	1,210,729,528	31,954,148	28,393,024
BCSSTK15	13,688,813,940	1,048,872,525	481,766,496
BCSSTK16	29,179,705,516	553,218,374	145,053,771
BCSSTK17	253,889,063,160	1,361,456,809	637,065,955
BCSSTK18	137,861,398,502	4,006,320,908	2,012,990,730
BCSSTK19	13,056,150	197,659	126,733
BCSSTK23	4,771,733,090	1,167,104,345	304,238,013
BCSSTK24	10,566,075,667	95,085,626	107,450,852
BCSSTK25	548,110,368,126	20,380,728,988	525,698,984
BCSSTK26	814,879,011	9,266,978	14,716,146

Table 3: Average factor floating point operations associated with Ran/new-minimal and with LEX M. Averages are taken over ten runs with randomly permuted adjacency structures.

NASA2910 and BCSSTK24, for which the time to confirm minimality is significantly greater than the MDint ordering time. For BCSSTK17, BCSSTK19, and BCSSTK26, the average time to compute the minimal ordering divided by the average number of major iterations gives the average time per major iteration, which for each of these problems is less than but roughly comparable to the MDint ordering time. Only BCSSTK19, with its 8.1 major iterations, has an MDint/new-minimal time (0.19) that approaches in magnitude the time for LEX M (0.37).

5.2. Random

In the previous subsection, on the whole, the MDint initial ordering did the work of computing minimal orderings, while the new minimal ordering algorithm merely detected minimality. In this subsection we look at the opposite extreme: the initial orderings are random, and the effort to achieve minimality is exerted solely by the new minimal ordering algorithm. Because of large run times, we run tests on the same test set of relatively small problems used in the previous subsection. The results of the Ran/new-minimal runs are presented in Tables 3 and 4.

The initial random orderings are very poor, as expected; the factor flops for the random orderings are one or two orders of magnitude larger than the factor flops for

Matrix	Ran/new-minimal				LEX M
	Ran time	Major iter.	Minimal time	Total time	time
DIS060	0.02	178.8	50.30	50.32	22.30
DIS090	0.05	371.4	383.54	383.59	134.27
DIS120	0.09	570.5	1665.96	1666.05	440.42
NASA1824	0.01	112.9	15.36	15.37	6.09
NASA2910	0.02	83.6	25.37	25.39	31.02
NASA4704	0.03	232.6	133.07	133.10	47.88
SPA060	0.02	188.1	52.46	52.48	18.89
SPA090	0.05	380.5	403.84	403.89	126.72
SPA120	0.09	591.8	1766.61	1766.70	452.17
BCSSTK13	0.01	188.4	52.60	52.61	11.88
BCSSTK14	0.01	69.0	8.45	8.46	7.87
BCSSTK15	0.02	291.5	190.56	190.58	37.15
BCSSTK16	0.03	133.3	84.36	84.39	90.07
BCSSTK17	0.07	410.1	1315.54	1315.61	331.14
BCSSTK18	0.07	1171.7	9608.18	9608.25	138.01
BCSSTK19	0.01	45.1	1.20	1.21	0.37
BCSSTK23	0.02	473.5	350.30	350.32	12.84
BCSSTK24	0.02	103.3	31.20	31.22	37.98
BCSSTK25	0.09	1441.3	17374.76	17374.85	565.62
BCSSTK26	0.01	120.1	16.11	16.12	4.43

Table 4: Average ordering times (in CPU seconds) and average number of major iterations for Ran/new-minimal and for LEX M. Averages are taken over ten runs with randomly permuted adjacency structures.

the corresponding final minimal orderings. The large reductions in factor flops require many major iterations; generally, hundreds of iterations are required, with over a thousand iterations required for BCSSTK18 and BCSSTK25. Though the final orderings greatly improve upon the initial random orderings, the final orderings are poor compared with the MDint orderings; moreover, the Ran/new-minimal orderings are poor compared with the LEX M orderings, with Ran/new-minimal orderings producing more factor flops than LEX M for 17 of the 20 problems. For the following matrices, the Ran/new-minimal ordering produces over twice as many factor flops as LEX M: DIS120, BCSSTK15, BCSSTK16, BCSSTK17, BCSSTK23, and BCSSTK25. The three matrices where Ran/new-minmal outperforms LEX M are small (NASA1824, BCSSTK24, and BCSSTK26), and the improvement of LEX M over Ran/new-minimal orderings for these matrices is relatively small.

It is not surprising that the run times to compute the initial random orderings are extremely small compared with the large run times to compute the associated minimal orderings. After all, a random ordering is obtained by a single $O(n \log n)$ sort. Several factors contribute to the exceptionally large run times for Ran/new-minimal orderings. First, and most obvious, is the large number of major iterations required for each problem. Second, the cost of each iteration is increased by the large amounts of fill that must be represented by the sequence of quotient graphs. Third, random orderings lead to relatively small supernodes, so the minimum degree refinement algorithm enjoys limited compression from supernodes. Fourth, the minimum degree refinement code uses the trick described by Amestoy, Davis, and Duff [1] of recompressing the quotient graphs when space is exhausted; the large amounts of fill and relatively small supernodes lead to many recompressions.

The run times for Ran/new-minimal are often poor compared with the LEX M orderings, with 17 out of 20 problems requiring more run-time than than LEX M. For many of the matrices, it requires a factor of two up to a factor of four more time. The results in this subsection, along with the rest of the results in this section, indicate that the new minimal ordering algorithm depends on a high-quality initial ordering to obtain a high-quality minimal ordering using small run time.

5.3. Minimum degree with external degree

Next, we obtained initial orderings from a minimum degree algorithm with external degree and no multiple elimination (MDext) [7]. We include it because it has become a standard by which other orderings are evaluated and because we wish to compare it with MDint. We include all the test matrices used in the previous two subsections and add some larger matrices to the test set, increasing the total from 20 to 33 matrices. Tables 5 and 6 present the results of our tests.

The results are quite similar to those obtained with initial orderings from MDint. However, among the 20 smaller problems used in the previous two subsections, only

Matrix	MDext/new-minimal		
	Avg. factor flops		Major
	Initial MDext	Final	iter.
CRY01	318,558,468	318,451,818	1.3
CRY02	4,042,557,744	4,042,253,531	1.2
CRY03	13,427,942,015	13,427,137,055	1.3
DIS060	9,854,297	9,854,297	1.0
DIS090	40,515,917	40,515,917	1.0
DIS120	112,984,002	112,984,002	1.0
NASA1824	4,922,550	4,912,521	1.2
NASA2910	23,065,112	23,042,832	2.0
NASA4704	36,171,302	36,171,302	1.0
SPA060	16,804,878	16,804,878	1.0
SPA090	66,258,263	66,258,263	1.0
SPA120	181,019,709	181,019,709	1.0
BCSSTK13	59,521,086	59,417,976	1.2
BCSSTK14	9,276,441	9,276,441	1.0
BCSSTK15	169,602,058	169,601,988	1.1
BCSSTK16	140,546,882	140,423,998	1.6
BCSSTK17	197,614,074	197,505,968	1.5
BCSSTK18	134,648,713	134,058,723	2.2
BCSSTK19	101,040	98,941	5.6
BCSSTK23	141,524,137	141,420,609	1.3
BCSSTK24	36,204,596	36,204,596	1.0
BCSSTK25	325,741,745	322,543,996	3.0
BCSSTK26	1,726,326	1,725,622	2.0
BCSSTK28	38,771,930	38,752,863	2.0
BCSSTK29	429,195,651	427,430,429	2.0
BCSSTK30	933,847,752	933,847,752	1.0
BCSSTK31	2,510,827,417	2,509,665,848	2.0
BCSSTK32	1,059,279,331	1,057,542,718	2.3
BCSSTK33	1,318,874,270	1,318,744,558	1.5
BCSSTK35	400,228,680	398,868,766	3.1
BCSSTK36	619,945,112	619,939,799	1.4
BCSSTK37	556,241,428	556,221,089	2.0

Table 5: Average factor floating point operations associated with MDext/new-minimal. Average is taken over ten runs with randomly permuted adjacency structures.

Matrix	MDext/new-minimal			
	MDext time	Major iter.	Minimal time	Total time
CRY01	0.51	1.3	0.60	1.11
CRY02	2.11	1.2	2.15	4.26
CRY03	5.46	1.3	4.91	10.37
DIS060	0.13	1.0	0.13	0.26
DIS090	0.34	1.0	0.35	0.69
DIS120	0.73	1.0	0.83	1.56
NASA1824	0.08	1.2	0.08	0.16
NASA2910	0.23	2.0	0.51	0.74
NASA4704	0.24	1.0	0.24	0.48
SPA060	0.13	1.0	0.13	0.26
SPA090	0.34	1.0	0.36	0.70
SPA120	0.72	1.0	0.80	1.52
BCSSTK13	0.30	1.2	0.19	0.49
BCSSTK14	0.13	1.0	0.09	0.22
BCSSTK15	0.77	1.1	0.35	1.12
BCSSTK16	0.43	1.6	0.71	1.14
BCSSTK17	1.13	1.5	1.43	2.56
BCSSTK18	1.13	2.2	1.60	2.73
BCSSTK19	0.01	5.6	0.12	0.13
BCSSTK23	0.65	1.3	0.25	0.90
BCSSTK24	0.16	1.0	0.19	0.35
BCSSTK25	2.48	3.0	4.12	6.60
BCSSTK26	0.11	2.0	0.14	0.25
BCSSTK28	0.21	2.0	0.53	0.74
BCSSTK29	2.00	2.0	2.78	4.78
BCSSTK30	4.20	1.0	3.71	7.91
BCSSTK31	6.03	2.0	7.60	13.63
BCSSTK32	6.76	2.3	11.71	18.47
BCSSTK33	1.38	1.5	1.53	2.91
BCSSTK35	3.54	3.1	8.94	12.48
BCSSTK36	2.08	1.4	2.80	4.88
BCSSTK37	2.43	2.0	4.42	6.85

Table 6: Average ordering times (in CPU seconds) and average number of major iterations for MDext/new-minimal. Averages are taken over ten runs with randomly permuted adjacency structures.

9 have all ten initial orderings minimal, in contrast to 17 when the initial orderings are produced by MDint. Nonetheless, for every matrix but two, the reduction in factor flops from initial to final ordering is less than 0.5%. For BCSSTK25, an average of three major iterations leads to an average of 1% reduction in factor flops; for the extremely small problem BCSSTK19, an average of 5.6 major iterations leads to an average of 2% reduction in factor flops. The initial MDext orderings are very nearly minimal and stand to gain very little from trying to squeeze out critical edges.

5.4. Nested dissection

An ND ordering finds a node bisector of the graph of A and numbers these vertices last in the ordering. It applies this numbering process recursively to the remaining pieces (i.e., connected components) of the graph. Current implementations apply this numbering process to the graph until each of the remaining pieces has fewer than some given number of vertices. (The ordering package we use subdivides no piece with 200 or fewer vertices.) Following Ashcraft and Liu [2], we call these small pieces that remain to be labeled *domains*.

We obtained initial ND orderings from the METIS ND algorithm [6]. We executed routine `METIS_NODEND` in version 3.0.3 of METIS with the default user-supplied options (`option(0)=0`). We made one change to the algorithm in METIS; we postprocessed the ordering obtained from METIS so that each domain is ordered using the constrained minimum degree algorithm (with external degrees). Constrained minimum degree was introduced by Liu [8] and has been used in the ND algorithms of Hendrickson and Rothberg [5] and Ashcraft and Liu [2]. Constrained minimum degree applies minimum degree to the vertices of a domain, using degrees in the complete elimination graph.

The results of our tests are shown in Tables 7 and 8. None of the initial ND orderings are minimal; some are quite close to minimal, while others are quite far from minimal (as measured by the percent decrease in factor flops from the initial to the final orderings). For 20 of the 32 matrices there is a decrease in factor flops of 2% or greater; for 12 of the 32 matrices there is a decrease in factor flops of 4% or greater; for 5 of the 32 matrices there is a decrease in factor flops of 11% or greater. Note that the number of major iterations is only loosely connected with the percent reduction in factor flops: for example, BCSSTK16 has a 14.10% reduction in 4.7 major iterations, while BCSSTK13 has a 0.75% reduction in 6.3 major iterations.

It is known that ND does not necessarily order the vertices of the separators in the most efficient manner [2, 5, 15]. The vertices within a separator are numbered arbitrarily by ND even though one ordering of the separator may reduce fill better than another. On a more global level, an ND ordering may create significant amounts of extra fill when it is used to order matrices arising from long, narrow structures. A cannonical example of this is ND applied to a path: minimum degree numbers the singleton separators from the ends on in creating no fill, while ND imposes an order

Matrix	ND/new-minimal			
	Avg. factor flops			Major
	Initial ND	Final	% decrease	iter.
CRY01	269,627,898	262,579,422	2.61	2.0
CRY02	1,880,205,121	1,874,577,047	0.30	2.0
CRY03	5,569,301,060	5,319,366,551	4.49	2.0
DIS060	11,401,297	11,100,794	2.64	2.3
DIS090	42,021,435	40,995,218	2.44	3.6
DIS120	109,090,527	105,289,465	3.48	2.9
NASA1824	5,781,541	5,695,467	1.49	3.0
NASA2910	23,121,841	22,688,614	1.87	5.0
NASA4704	35,646,032	34,814,447	2.33	4.1
SPA060	15,157,149	15,154,770	0.02	1.6
SPA090	55,866,936	55,600,237	0.48	2.1
SPA120	138,377,652	138,232,951	0.10	2.7
BCSSTK13	52,599,234	52,206,328	0.75	6.3
BCSSTK14	7,991,080	7,959,466	0.40	1.2
BCSSTK15	86,038,909	84,751,326	1.50	6.0
BCSSTK16	151,625,532	130,252,776	14.10	4.7
BCSSTK17	189,513,839	161,230,829	14.92	6.9
BCSSTK18	86,877,095	84,761,781	2.43	11.7
BCSSTK19	116,858	99,760	14.63	19.9
BCSSTK23	95,742,282	94,088,996	1.73	10.2
BCSSTK24	37,133,108	36,344,740	2.12	2.3
BCSSTK25	372,742,122	255,754,708	31.39	12.4
BCSSTK26	2,058,885	1,934,046	6.06	7.6
BCSSTK28	47,149,103	45,211,552	4.11	5.5
BCSSTK29	330,065,817	315,180,812	4.51	8.8
BCSSTK30	1,164,467,420	1,034,372,362	11.17	8.5
BCSSTK31	1,186,768,833	1,170,581,409	1.36	8.3
BCSSTK32	1,338,162,758	1,264,908,765	5.47	9.0
BCSSTK33	848,645,138	838,896,022	1.15	2.5
BCSSTK35	486,821,573	460,586,443	5.39	11.9
BCSSTK36	653,046,189	639,185,353	2.12	6.0
BCSSTK37	641,497,837	595,023,291	7.24	7.6

Table 7: Average factor floating point operations associated with ND/new-minimal. Average is taken over ten runs with randomly permuted adjacency structures.

Matrix	ND/new-minimal			
	ND time	Major iter.	Minimal time	Total time
CRY01	0.69	2.00	0.92	1.61
CRY02	3.09	2.00	3.33	6.42
CRY03	6.20	2.00	7.14	13.34
DIS060	0.65	2.30	0.30	0.95
DIS090	1.85	3.60	1.27	3.12
DIS120	3.73	2.90	2.31	6.04
NASA1	0.16	3.00	0.24	0.40
NASA2	0.73	5.00	1.19	1.92
NASA4	0.65	4.10	0.84	1.49
SPA060	0.62	1.60	0.21	0.83
SPA090	1.88	2.10	0.72	2.60
SPA120	3.54	2.70	2.14	5.68
BCSSTK13	0.58	6.30	0.94	1.52
BCSSTK14	0.38	1.20	0.11	0.49
BCSSTK15	1.75	6.00	1.69	3.44
BCSSTK16	0.66	4.70	2.01	2.67
BCSSTK17	2.40	6.90	5.99	8.39
BCSSTK18	3.48	11.70	8.18	11.66
BCSSTK19	0.04	19.90	0.43	0.47
BCSSTK23	0.84	10.20	2.08	2.92
BCSSTK24	0.20	2.30	0.44	0.64
BCSSTK25	5.56	12.40	15.76	21.32
BCSSTK26	0.19	7.60	0.53	0.72
BCSSTK28	0.19	5.50	1.57	1.76
BCSSTK29	5.54	8.80	12.13	17.67
BCSSTK30	7.47	8.50	30.53	38.00
BCSSTK31	11.42	8.30	32.92	44.34
BCSSTK32	9.96	9.00	46.97	56.93
BCSSTK33	3.59	2.50	2.56	6.15
BCSSTK35	3.35	11.90	34.93	38.28
BCSSTK36	1.97	6.00	11.91	13.88
BCSSTK37	3.44	7.60	17.56	21.00

Table 8: Average ordering times (in CPU seconds) and average number of major iterations for ND/new-minimal. Averages are taken over ten runs with randomly permuted adjacency structures.

on the singleton separators that creates fill. The matrix BCSSTK25 in our test set is an example of this phenomenon. It arises from a finite element model of a long narrow structure, namely, a 76-story skyscraper, and it incurs by far the greatest reduction in factor flops: 31.39%.

The times for the ND orderings are greater than those for the MDext orderings, but they are still quite reasonable. Because the ND orderings are not so nearly minimal, the number of major iterations for the initial ND orderings are much greater than the number of major iterations for the initial MDext orderings. Rising with the number of major iterations is the time to compute the minimal orderings, which can be quite substantial. Neither the number of major iterations nor the run times, however, approach in magnitude the number of major iterations or the run times, respectively, for Ran/new-minimal.

5.5. Multisection

The MS ordering algorithm was a response to difficulties encountered using the ND ordering algorithm. An MS ordering is obtained from an ND algorithm as follows. The set of separators and domains is computed as before, and the vertices of the domains are to be ordered before the vertices of the separators as before. The domains are again eliminated using constrained minimum degree. Let X be the set of vertices obtained by forming the union of all the domains. An MS ordering is then obtained by applying the minimum degree ordering (with external degree) to the quotient graph representation of the elimination graph G_X. This strategy for ordering the separators has appeared in [2, 5, 15]. Our results using an MS initial ordering appear in Tables 9 and 10.

Our results with MS ordering corroborate those reported in Ashcraft and Liu [2]. Overall, the MS ordering reduces factor flops better than either the MDext or ND orderings. For four of the matrices, the MS initial ordering is a minimal ordering. Overall, the number of major iterations lies between the number of major iterations for the MDext ordering and the number of major iterations for the ND ordering. Applying minimum degree to the elimination graph G_X causes the initial MS orderings to be much closer to minimal than the initial ND orderings were. For all but four matrices, the reduction in factor flops is under 1.0%. For three of these four matrices the reduction is small: 1.1% for NASA2910, 1.3% for BCSSTK25, and 3.4% for BCSSTK26. The only problem for which there is a large reduction is the tiny problem BCSSTK19, and here we are merely obtaining exactly the same results that we obtained for the initial ND ordering.

The MS run times are simply ND run times with the time for ordering G_X by minimum degree added in. The code was written for ease of programming and not for optimal time efficiency; still the MS run times are reasonably small. Because the major iterations are reduced in number, the total run times are generally smaller than those for ND/new-minimal.

Matrix	MS/new-minimal		
	Avg. factor flops		Major
	Initial MS	Final	iter.
CRY01	248,164,448	248,141,000	1.2
CRY02	1,877,497,131	1,877,497,131	1.0
CRY03	5,587,997,831	5,587,987,076	1.1
DIS060	9,418,207	9,417,758	1.1
DIS090	32,561,032	32,559,649	1.6
DIS120	82,128,063	82,127,446	1.5
NASA1824	5,437,036	5,433,668	1.8
NASA2910	22,946,750	22,700,805	3.7
NASA4704	35,236,673	35,193,747	2.4
SPA060	14,158,894	14,158,894	1.0
SPA090	50,515,064	50,515,064	1.0
SPA120	125,367,743	125,364,701	1.8
BCSSTK13	51,187,752	51,135,554	1.6
BCSSTK14	7,976,010	7,952,477	1.2
BCSSTK15	85,158,867	85,069,649	2.7
BCSSTK16	125,842,140	125,142,088	2.6
BCSSTK17	146,569,782	145,858,491	3.3
BCSSTK18	84,328,146	83,745,632	7.8
BCSSTK19	116,858	99,760	19.9
BCSSTK23	96,860,748	96,666,031	5.9
BCSSTK24	34,742,704	34,742,704	1.0
BCSSTK25	242,388,850	239,125,956	8.7
BCSSTK26	1,977,138	1,909,224	6.3
BCSSTK28	36,264,698	36,226,333	2.6
BCSSTK29	343,010,925	341,157,227	7.0
BCSSTK30	867,524,321	867,090,637	4.6
BCSSTK31	1,458,766,012	1,457,656,825	5.3
BCSSTK32	923,649,582	922,128,407	5.0
BCSSTK33	754,734,712	754,522,294	1.2
BCSSTK35	377,911,035	376,542,336	4.4
BCSSTK36	498,846,780	498,749,689	2.9
BCSSTK37	466,486,791	466,085,474	4.4

Table 9: Average factor floating point operations associated with MS/new-minimal. Average is taken over ten runs with randomly permuted adjacency structures.

matrix	MS/new-minimal			
	MS time	major iter.	minimal time	total time
CRY01	1.21	1.2	0.58	1.79
CRY02	4.96	1.0	1.74	6.70
CRY03	10.15	1.1	3.95	14.10
DIS060	0.80	1.1	0.14	0.94
DIS090	2.34	1.6	0.56	2.90
DIS120	4.48	1.5	1.17	5.65
NASA1	0.23	1.8	0.12	0.35
NASA2	1.05	3.7	0.86	1.91
NASA4	0.86	2.4	0.49	1.35
SPA060	0.79	1.0	0.13	0.92
SPA090	2.31	1.0	0.33	2.64
SPA120	4.55	1.8	1.47	6.02
BCSSTK13	0.76	1.6	0.23	0.99
BCSSTK14	0.49	1.2	0.11	0.60
BCSSTK15	2.23	2.7	0.85	3.08
BCSSTK16	1.13	2.6	1.11	2.24
BCSSTK17	3.38	3.3	3.03	6.41
BCSSTK18	4.42	7.8	5.74	10.16
BCSSTK19	0.07	19.9	0.47	0.54
BCSSTK23	1.08	5.9	1.16	2.24
BCSSTK24	0.40	1.0	0.19	0.59
BCSSTK25	7.48	8.7	11.42	18.90
BCSSTK26	0.28	6.3	0.43	0.71
BCSSTK28	0.45	2.6	0.70	1.15
BCSSTK29	7.19	7.0	9.89	17.08
BCSSTK30	11.21	4.6	15.79	27.00
BCSSTK31	15.71	5.3	20.45	36.16
BCSSTK32	14.94	5.0	24.57	39.51
BCSSTK33	4.61	1.2	1.22	5.83
BCSSTK35	6.24	4.4	13.15	19.39
BCSSTK36	4.12	2.9	5.77	9.89
BCSSTK37	5.87	4.4	9.90	15.77

Table 10: Average ordering times (in CPU seconds) and average number of major iterations for MS/new-minimal. Averages are taken over ten runs with randomly permuted adjacency structures.

6. Concluding remarks

We devised a new characterization of candidate edges that leads to a simple "block-restricted minimum degree" elimination process to remove candidate edges. We then devised an algorithm that removes candidate edges until a minimal chordal supergraph and a minimal ordering are obtained. Empirically, the method MDint/minimal-new improves on both the ordering quality and the ordering time of the old method LEX M.

In the past, minimal orderings were not good heuristic orderings; they did not approximate minimum orderings well [14, page 282]. The new approach in this paper deals with this shortcoming of past minimal ordering algorithms. Because it starts with any initial ordering and refines that ordering until minimality is achieved, the algorithm can turn good heuristic orderings into good minimal orderings.

However, we saw that this capability makes little or no difference when the initial ordering is produced by MDint. It also makes little difference when the initial ordering is produced by MS or MDext. Each of these ordering heuristics produces orderings that are "nearly" minimal and trying to improve them by making them minimal makes little sense in the practical setting. We did see, however, that initial ND orderings can be quite far from minimal. MS can be viewed as a way to fix this problem with ND.

The time complexity of the new minimal ordering algorithm is unknown, in part because of the use of minimum degree and quotient graphs, and in part because a bound on the number of major steps in the new algorithm is unknown. We conjecture that the number of major steps is $O(n)$.

7. References

[1] P.R Amestoy, T.A. Davis, and I.S. Duff. An approximate minimum degree ordering algorithm. *SIAM J. Matrix Anal. Appl.*, 17:886–905, 1996.

[2] C. Ashcraft and J.W.H. Liu. Robust ordering of sparse matrices using multisection. *SIAM J. Matrix Anal. Appl.*, 19:816–832, 1998.

[3] A. George and J. W-H. Liu. The evolution of the minimum degree ordering algorithm. *SIAM Review*, 31:1–19, 1989.

[4] J.R. Gilbert, E. Ng, and B.W. Peyton. An efficient algorithm to compute row and column counts for sparse Cholesky factorization. *SIAM J. Matrix Anal. Appl.*, 15:1075–1091, 1994.

[5] B. Hendrickson and E. Rothberg. Improving the run time and quality of nested dissection ordering. *SIAM J. Sci. Comput.*, 20:468–489, 1998.

[6] G. Karypis and V. Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM J. Sci. Comput.*, 20:359–392, 1998.

[7] J. W-H. Liu. Modification of the minimum degree algorithm by multiple elimination. *ACM Trans. Math. Software*, 11:141–153, 1985.

[8] J. W-H. Liu. The minimum degree ordering with constraints. *SIAM J. Sci. Statist. Comput.*, 10:1136–1145, 1989.

[9] J. W-H. Liu. The role of elimination trees in sparse factorization. *SIAM J. Matrix Anal. Appl.*, 11:134–172, 1990.

[10] J.W.H. Liu, E. Ng, and B.W. Peyton. On finding supernodes for sparse matrix computations. *SIAM J. Matrix Anal. Appl.*, 14:242–252, 1993.

[11] T. Ohtsuki. A fast algorithm for finding an optimal ordering in the vertex elimination on a graph. *SIAM J. Comput.*, 5:133–145, 1976.

[12] D.J. Rose. Triangulated graphs and the elimination process. *J. Math. Anal. Appl.*, 32:597–609, 1970.

[13] D.J. Rose. A graph-theoretic study of the numerical solution of sparse positive definite systems of linear equations. In R. C. Read, editor, *Graph Theory and Computing*, pages 183–217. Academic Press, 1972.

[14] D.J. Rose, R.E. Tarjan, and G.S. Lueker. Algorithmic aspects of vertex elimination on graphs. *SIAM J. Comput.*, 5:266–283, 1976.

[15] E. Rothberg. Robust ordering of sparse matrices: A minimum degree nested dissection hybrid. Silicon Graphics manuscript, 1996.

[16] R. Schreiber. A new implementation of sparse Gaussian elimination. *ACM Trans. Math. Software*, 8:256–276, 1982.

[17] M. Yannakakis. Computing the minimum fill-in is NP-complete. *SIAM J. Alg. Disc. Meth.*, 2:77–79, 1981.

INTERNAL DISTRIBUTION

1. B. R. Appleton
2. B. A. Carreras
3. E. F. D'Azevedo
4. T. S. Darland
5. M. R. Leuze
6–11. B. W. Peyton
12. C. H. Romine
13. B. A. Worley
14. P. H. Worley
15. T. Zacharia
16. Central Research Library
17. Laboratory Records - RC
18–19. Laboratory Records / OSTI

EXTERNAL DISTRIBUTION

20. Cleve Ashcraft, Boeing Computer Services, P. O. Box 24346, M/S 7L-21, Seattle, WA 98124-0346
21. Bengt Aspvall, Institutt for Informatikk, Universitetet i Bergen, N-5020 Bergen, Norway
22. Donald M. Austin, 6196 EECS Building, University of Minnesota, 200 Union Street SE, Minneapolis, MN 55455
23. Robert G. Babb, Oregon Graduate Center, CSE Department, 19600 N.W. Walker Road, Beaverton, OR 97006
24. David H. Bailey, NASA Ames, Mail Stop 258-5, NASA Ames Research Center, Moffet Field, CA 94035
25. Jesse L. Barlow, Department of Computer Science and Engineering, 220 Pond Laboratory, The Pennsylvania State University, University Park, PA 16802-6106
26. Edward H. Barsis, Computer Science and Mathematics, P. O. Box 5800, Sandia National Laboratory, Albuquerque, NM 87185
27. Robert E. Benner, Parallel Processing Division 1413, Sandia National Laboratories, P. O. Box 5800, Albuquerque, NM 87185
28. Jean R. S. Blair, D/EE&CS, The United States Military Academy, West Point, NY 10996
29. Ake Bjorck, Department of Mathematics, Linkoping University, S-581 83 Linkoping, Sweden
30. Roger W. Brockett, Harvard University, Pierce Hall, 29 Oxford Street, Cambridge, MA 02138
31. Bill L. Buzbee, Scientific Computing Division, National Center for Atmospheric Research, P. O. Box 3000, Boulder, CO 80307
32. Donald A. Calahan, Department of Electrical and Computer Engineering, University of Michigan, Ann Arbor, MI 48109
33. Jagdish Chandra, Army Research Office, P. O. Box 12211, Research Triangle Park, NC 27709

34. Melvyn Ciment, National Science Foundation, 1800 G Street N.W., Washington, DC 20550

35. Tom Coleman, Department of Computer Science, Cornell University, Ithaca, NY 14853

36. Paul Concus, Mathematics and Computing, Lawrence Berkeley Laboratory, Berkeley, CA 94720

37. Jane K. Cullum, IBM T. J. Watson Research Center, P. O. Box 218, Yorktown Heights, NY 10598

38. Helen Davis, Computer Science Department, Stanford University, Stanford, CA 94305

39. Craig Douglas, IBM T. J. Watson Research Center, P. O. Box 218, Yorktown Heights, NY 10598-0218

40. Iain S. Duff, Atlas Centre, Rutherford Appleton Laboratory, Chilton, Oxon OX11 0QX, England

41. Stanley Eisenstat, Department of Computer Science, Yale University, P. O. Box 2158 Yale Station, New Haven, CT 06520

42. Geoffrey C. Fox, NPAC, 111 College Place, Syracuse University, Syracuse, NY 13244-4100

43. Paul O. Frederickson, ACL, MS B287, Los Alamos National Laboratory, Los Alamos, NM 87545

44. Offir Frieder, George Mason University, Science and Technology Building, Computer Science Department, 4400 University Drive, Fairfax, VA 22030-4444

45. Robert E. Funderlic, Department of Computer Science, North Carolina State University, Raleigh, NC 27650

46. Dennis B. Gannon, Computer Science Department, Indiana University, Bloomington, IN 47401

47. W. Morven Gentleman, Division of Electrical Engineering, National Research Council, Building M-50, Room 344, Montreal Road, Ottawa, Ontario, Canada K1A 0R8

48. Alan George, Department of Computer Science, University of Waterloo, Waterloo, Ontario, Canada N2L 3G1

49. John R. Gilbert, Xerox Palo Alto Research Center, 3333 Coyote Hill Road, Palo Alto, CA 94304

50. Gene Golub, Computer Science Department, Stanford University, Stanford, CA 94305

51. Joseph F. Grcar, Division 8245, Sandia National Laboratories, Livermore, CA 94551-0969

52. William D. Gropp, Mathematics and Computer Science Division, Argonne National Laboratory, 9700 South Cass Avenue, Argonne, IL 60439

53. Eric Grosse, AT&T Bell Labs 2T-504, Murray Hill, NJ 07974

54. John L. Gustafson, Ames Laboratory, 236 Wilhelm Hall, Iowa State University, Ames, IA 50011-3020

55. Sven J. Hammarling, The Numerical Algorithms Group, Ltd., Wilkinson House, Jordan Hill Road, Oxford OX2 8DR, UK

56. Michael T. Heath, Department of Computer Science, 2304 Digital Computer Laboratory, University of Illinois, 1304 West Springfield Avenue, Urbana, IL 61801-2987

57. N. J. Higham, Department of Mathematics, University of Manchester, Gtr Manchester, M13 9PL, England

58. Dan Hitchcock, Division of Mathematical, Information, and Computational Sciences, U. S. Department of Energy, Office of Science, SC-31, 19901 Germantown Road, Room E-232, Germantown, MD 20874-1290

59. Charles J. Holland, Air Force Office of Scientific Research, Building 410, Bolling Air Force Base, Washington, DC 20332

60. Fred Howes, Division of Mathematical, Information, and Computational Sciences, U. S. Department of Energy, Office of Science, SC-31, 19901 Germantown Road, Room E-236, Germantown, MD 20874-1290

61. Robert E. Huddleston, Computation Department, Lawrence Livermore National Laboratory, P. O. Box 808, Livermore, CA 94550

62. Jenq-Neng Hwang, Department of Electrical Engineering, FT-10, University of Washington, Seattle, WA 98195

63. Bo Kagstrom, Institute of Information Processing, University of Umea, 5-901 87 Umea, Sweden

64. Malvyn Kalos, Cornell Theory Center, Engineering and Theory Center Building, Cornell University, Ithaca, NY 14853-3901

65. Hans Kaper, Mathematics and Computer Science Division, Argonne National Laboratory, 9700 South Cass Avenue, Bldg. 221, Argonne, IL 60439

66. Linda Kaufman, Bell Laboratories, 600 Mountain Avenue, Murray Hill, NJ 07974

67. Robert J. Kee, Applied Mathematics Division 8245, Sandia National Laboratories, Livermore, CA 94551-0969

68. Kenneth Kennedy, Department of Computer Science, Rice University, P. O. Box 1892, Houston, TX 77001

69. Tom Kitchens, Division of Mathematical, Information, and Computational Sciences, U. S. Department of Energy, Office of Science, SC-31, 19901 Germantown Road, Room E-235, Germantown, MD 20874-1290

70. Michael Langston, Department of Computer Science, University of Tennessee, Knoxville, TN 37996-1301

71. Richard Lau, Office of Naval Research, Code 111MA, 800 Quincy Street, Boston Tower 1, Arlington, VA 22217-5000

72. Robert L. Launer, Army Research Office, P. O. Box 12211, Research Triangle Park, NC 27709

73. Robert Leland, Sandia National Laboratories, 1424, P. O. Box 5800, Albuquerque, NM 87185-5800

74. John G. Lewis, Boeing Computer Services, P. O. Box 24346, M/S 7L-21, Seattle, WA 98124-0346

75. Heather M. Liddell, Center for Parallel Computing, Department of Computer Science and Statistics, Queen Mary College, University of London, Mile End Road, London E1 4NS, England

76. Joseph Liu, Department of Computer Science, York University, 4700 Keele Street, Downsview, Ontario, Canada M3J 1P3

77. James McGraw, Lawrence Livermore National Laboratory, L-306, P. O. Box 808, Livermore, CA 94550

78. Paul C. Messina, Mail Code 158-79, California Institute of Technology, 1201 E. California Boulevard, Pasadena, CA 91125

79. Jorge J. More, Mathematics and Computer Science Division, Argonne National Laboratory, 9700 South Cass Avenue, Argonne, IL 60439

80. David Nelson, Associate Director, Computational and Technology Research, Office of Advanced Scientific Computing, U. S. Department of Energy, Office of Science, SC-30, 19901 Germantown Road, Room E-219, Germantown, MD 20874-1290

81. Esmond G. Ng, NERSC, MS 50F, Lawrence Berkeley National Laboratory, 1 Cyclotron Road, Berkeley, CA 94720

82. Dianne P. O'Leary, Computer Science Department, University of Maryland, College Park, MD 20742

83. James M. Ortega, Department of Computer Science, Thornton Hall, University of Virginia, Charlottesville, VA 22901

84. Merrell Patrick, National Science Foundation, 1800 G Street NW, Washington, DC 20550

85. Dan Pierce, Boeing Computer Services, P. O. Box 24346, M/S 7L-21, Seattle, WA 98124-0346

86. James C. T. Pool, Deputy Director, Caltech Concurrent Supercomputing Facility, California Institute of Technology, MS 158-79, Pasadena, CA 91125

87. David A. Poplawski, Department of Computer Science, Michigan Technological University, Houghton, MI 49931

88. Padma Raghavan, Department of Computer Science, University of Tennessee, Knoxville, TN 37996-1301

89. Daniel A. Reed, Computer Science Department, University of Illinois, Urbana, IL 61801

90. John K. Reid, Numerical Analysis Group, Central Computing Department, Atlas Centre, Rutherford Appleton Laboratory, Didcot, Oxon OX11 0QX, England

91. Donald J. Rose, Department of Computer Science, Duke University, Durham, NC 27706

92. Ahmed H. Sameh, Department of Computer Science, University of Minnesota, 200 Union Street SE, Minneapolis, MN 55455

93. Jorge Sanz, IBM Almaden Research Center, Department K53/802, 650 Harry Road, San Jose, CA 95120

94. Robert Schreiber, Hewlett-Packard Laboratories, 1501 Page Mill Road, M/S 3L-5, Palo Alto, CA 94304-1126

95. David S. Scott, Intel Scientific Computers, 15201 NW Greenbrier Parkway, Beaverton, OR 97006

96. The Secretary, Department of Computer Science and Statistics, The University of Rhode Island, Kingston, RI 02881

97. Douglas R. Shier, Martin Hall, O-21, Clemson University, Clemson, SC 29634-1907

98. Horst D. Simon, Mail Stop 50B-4230, Lawrence Berkeley National Laboratory, 1 Cyclotron Road, Berkeley, CA 94720

99. Tony Skjellum, Dept of Computer Science, Mississippi State University, P. O. Drawer CS, Mississippi State, MS 39762-5623

100. Burton Smith, Tera Computer Company, 400 North 34th Street, Suite 300, Seattle, WA 98103

101. Danny C. Sorensen, Department of Mathematical Sciences, Rice University, P. O. Box 1892, Houston, TX 77251

102. Rick Stevens, Mathematics and Computer Science Division, Argonne National Laboratory, 9700 South Cass Avenue, Argonne, IL 60439

103. G. W. Stewart, Computer Science Department, University of Maryland, College Park, MD 20742

104. Paul N. Swarztrauber, National Center for Atmospheric Research, P. O. Box 3000, Boulder, CO 80307

105. Wei Pai Tang, Department of Computer Science, University of Waterloo, Waterloo, Ontario, Canada N2l 3G1

106. Udaya B. Vemulapati, Department of Computer Science, University of Central Florida, Orlando, FL 32816-0362

107. Robert G. Voigt, National Science Foundation, Room 417, 1800 G Street NW, Washington, DC 20550

108. Michael D. Vose, 107 Ayres Hall, Department of Computer Science, University of Tennessee, Knoxville, TN 37996-1301

109. Phuong Vu, Cray Research, Inc., 19607 Franz Road, Houston, TX 77084

110. Daniel D. Warner, Martin Hall, O-203, Clemson University, Clemson, SC 29634-1907

111. Mary F. Wheeler, Department of Mathematical Sciences, Rice University, P. O. Box 1892, Houston, TX 77251

112. Andrew B. White, Computing Division, Los Alamos National Laboratory, Los Alamos, NM 87545