

ORNL/TM-10937

OAK RIDGE  
NATIONAL  
LABORATORY

MARTIN MARIETTA

## A Partitioning Strategy for Parallel Sparse Cholesky Factorization

G. A. Geist  
E. Ng

OAK RIDGE NATIONAL LABORATORY  
CENTRAL RESEARCH LIBRARY  
CIRCULATION SECTION  
OAK RIDGE, TN  
**LIBRARY LOAN COPY**  
DO NOT TRANSFER TO ANOTHER PERSON  
If you wish someone else to use this  
report, send to name with report and  
the library will arrange a loan.  
OCT 1989 1 11

OPERATED BY  
MARTIN MARIETTA ENERGY SYSTEMS, INC.  
FOR THE UNITED STATES  
DEPARTMENT OF ENERGY

Printed in the United States of America. Available from  
National Technical Information Service  
U.S. Department of Commerce  
5285 Port Royal Road, Springfield, Virginia 22161  
NTIS price codes—Printed Copy: A03; Microfiche A01

This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor any agency thereof, nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof.

ORNL/TM - 10937

Engineering Physics & Mathematics Division

Mathematical Sciences Section

**A PARTITIONING STRATEGY FOR PARALLEL SPARSE  
CHOLESKY FACTORIZATION**

*G. A. Geist*

*E. Ng*

Oak Ridge National Laboratory  
Mathematical Sciences Section  
P.O. Box 2009, Building 9207A  
Oak Ridge, Tennessee 37831-8083

Date Published: November, 1988

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

Prepared by the  
Oak Ridge National Laboratory  
Oak Ridge, Tennessee 37831  
operated by  
Martin Marietta Energy Systems, Inc.  
for the  
U.S. DEPARTMENT OF ENERGY  
under Contract No. DE-AC-05-84OR21400



3 4456 0283944 5



## Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
<b>2</b>	<b>Elimination Trees</b>	<b>1</b>
<b>3</b>	<b>Existing Partitioning Schemes</b>	<b>4</b>
<b>4</b>	<b>The Partitioning Algorithm</b>	<b>10</b>
<b>5</b>	<b>Experimental Results</b>	<b>14</b>
<b>6</b>	<b>References</b>	<b>19</b>



# A PARTITIONING STRATEGY FOR PARALLEL SPARSE CHOLESKY FACTORIZATION

G. A. Geist

E. Ng

## Abstract

This paper presents a solution to the problem of partitioning the work for sparse matrix factorization on a multiprocessor system. The goal of this partitioning strategy is to achieve load balancing and a high degree of concurrency among the processors while reducing the amount of processor-to-processor data communication. The task assignment strategy is based on the structure of the elimination tree for a given ordering, and can be applied to arbitrarily unbalanced trees. This is important because popular fill-reducing ordering methods, such as the minimum degree algorithm, often produce unbalanced elimination trees. Results from the Intel iPSC/2 are presented for various finite-element problems using both nested dissection and minimum degree orderings.



## 1. Introduction

In this paper we address the problem of partitioning the computational work in the factorization of sparse matrices for a multiprocessor system. More specifically, we consider the solution of  $Ax = b$ , where  $A$  is sparse, symmetric and positive definite. This system of equations is solved using the Cholesky factorization  $A = LL^T$ , where  $L$  is lower triangular. We partition the columns of  $L$  among the processors rather than the individual elements of the matrix because this level of granularity is well suited for most of the multiprocessors commercially available today [3,5]. Given this choice of granularity, the partitioning problem becomes how to assign columns of  $L$  to the individual processors so that the work load is balanced, and a high degree of concurrency is achieved among processors. It is also desirable to minimize the interaction between processors, which takes the form of message passing for distributed-memory computers or synchronization for shared-memory computers. While no particular architecture is assumed in this work, the results are particularly well suited to distributed-memory parallel computers such as hypercube multiprocessors.

The partitioning scheme we present can be thought of as a generalization of the *subtree-to-subcube* mapping scheme [8] to arbitrarily unbalanced elimination trees. The subtree-to-subcube scheme is designed to work with nested dissection orderings and with the number of processors equal to a power of 2 while the general scheme has no restrictions on the number of processors or ordering method.

An outline of the paper is as follows. In Section 2, we describe elimination trees and how they are used as a tool to identify parallelism. We survey existing partitioning schemes for the above problem, including subtree-to-subcube mapping, in Section 3 and describe our partitioning scheme in Section 4. Finally, some typical results for finite-element problems are presented in Section 5.

## 2. Elimination Trees

We assume the reader is familiar with basic graph theory notions and the correspondence between undirected graphs and the structure of symmetric matrices. For background in this area the reader is referred to [7]. In particular, the methods in this paper depend heavily on the notion of elimination trees and their properties. An elimination

tree corresponding to the structure of the Cholesky factor  $L$  is defined as follows. For each column  $j \leq n$ , if column  $j$  has off-diagonal nonzeros, then let  $\gamma(j)$  be the row subscript of the first off-diagonal nonzero in column  $j$ ; otherwise, let  $\gamma(j) = 0$ . An elimination tree of  $L$  is a tree with  $n$  vertices labelled from 1 to  $n$  and the parent of vertex  $j$  is  $\gamma(j)$ . The labelling of the vertices in an elimination tree corresponds to the columns of  $L$ , and the structure of the tree contains useful information pertaining to the factorization process.

We illustrate this process with an example. The Cholesky factor  $L$  of a matrix and its elimination tree are displayed in Figures 1 and 2, respectively. Consider the first

$$L = \begin{pmatrix} \times & & & & & \\ & \times & & & & \\ \times & & \times & & & \\ & & & \times & \times & \\ \times & \times & \times & \times & \times & \\ & \times & & \times & \times & \times \end{pmatrix}$$

Figure 1: Structure of a Cholesky factor.

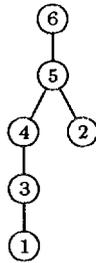


Figure 2: The elimination tree associated with the Cholesky factor in Figure 1.

step of Cholesky factorization of  $A$ . Suppose the first column of  $L$  has been computed. Since  $l_{21}$  is zero, the second column of  $L$  is not affected by column 1 of  $L$ . When column 2 of  $L$  is computed, only columns 5 and 6 are affected. These dependencies are expressed in the elimination tree in Figure 2. The elimination of column  $i$  in the matrix affects only those columns that are ancestors of vertex  $i$  in the elimination tree. In other words, columns in disjoint subtrees are independent during Cholesky factorization. The observation above and several other properties of elimination trees

are given in [14].

In the example, since nodes 1 and 2 are on different independent branches of the tree and thus do not affect each other, columns 1 and 2 of the Cholesky factor can be computed in parallel. Conversely, node 5 is the parent of nodes 2 and 4, and therefore column 5 cannot be computed until columns 2 and 4 are finished. A graphical interpretation of the factorization can be illustrated using the elimination tree. Computing a column of the Cholesky factor corresponds to removing or eliminating that node from the tree. At the first step, any or all of the leaf nodes can be eliminated. This creates a new set of leaf nodes in the tree, which can now be eliminated, and so on.

In general, the leaf nodes in the elimination tree denote all the independent columns of the sparse matrix, and the paths up the elimination tree to the root specify the column dependencies. Thus the elimination tree provides precise information about the column dependencies of  $L$  and hence can be used to partition columns of the sparse matrix among different processors.

The structure of the elimination tree depends on the structure of the matrix. Since the structure of the matrix changes under symmetric permutations, the structure of the elimination tree also changes. Ordering methods commonly used do not take the structure of the elimination tree into consideration. Instead, they rely on heuristics that reduce fill in the Cholesky factor [7]. While these methods produce good orderings in terms of fill and operation counts, they may not produce orderings that lend themselves to parallel computation. A desirable tree structure for parallel execution is short and wide. A wide tree indicates a high degree of concurrency can be exploited, and a short tree has a small parallel execution time. Nested dissection orderings [2,6] often produce trees that are wide and short, and for some very regular problems such as  $k \times k$  grids, Some nested dissection orderings produce balanced binary trees. However, the elimination trees associated with minimum degree orderings are often unbalanced. In Figures 3 and 4 the elimination trees are displayed for a nested dissection ordering and a minimum degree ordering, respectively, on a  $7 \times 7$  grid.

If the elimination tree is not balanced, then there are two obvious approaches to the partitioning problem. First, the tree could be manipulated in the hope that it will become more balanced, or at least have more branches, without increasing fill. A second approach is to use the unbalanced tree but to partition it in such a way

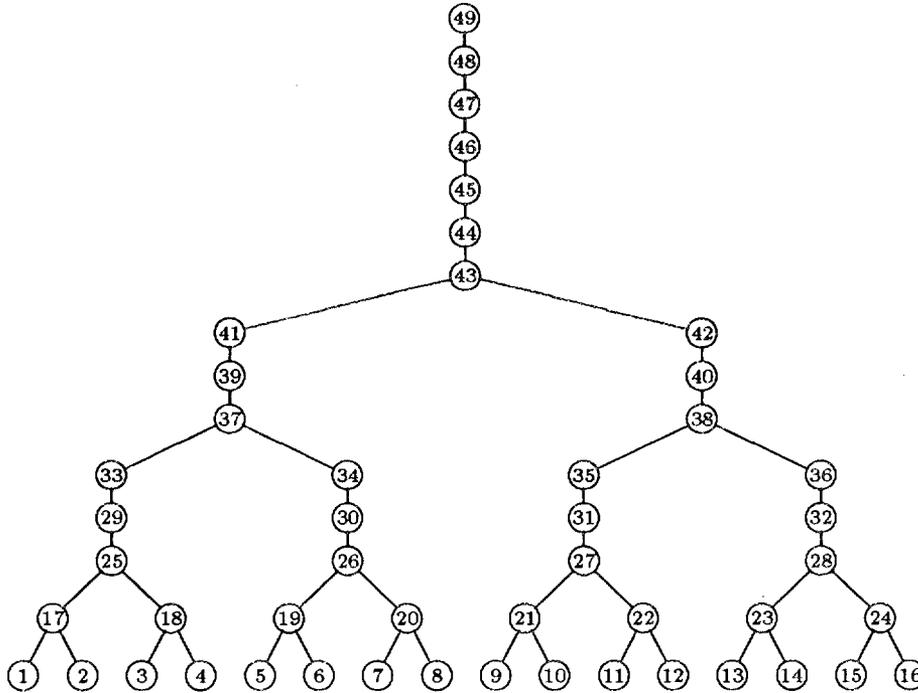


Figure 3: The elimination tree associated with a nested dissection ordering on a  $7 \times 7$  grid.

that the computational load across the processors is roughly equal. The first approach was taken by Liu in [12,13], where special tree rotation algorithms are described that attempt to reduce the tree height while preserving fill.

There are advantages and disadvantages to the tree rotation approach. In the ideal case, the rotated tree becomes shorter and nicely balanced. Unfortunately this behavior is the exception. For some trees the algorithm is unable to reduce the height of the tree. In most cases the algorithm reduces the height of the tree significantly, but the structure of the tree remains unbalanced. An example illustrating this fact is given in Figure 5 in which Liu's strategy is applied to the elimination tree in Figure 4.

### 3. Existing Partitioning Schemes

Several partitioning methods have been suggested for use on multiprocessors with a hypercube topology. In the dense case good load balance is achieved when the columns of the matrix are assigned to the processors using the following formula: column  $i$  is assigned to processor  $(i-1) \bmod p$ . This form of partitioning is called *wrap mapping* [1].

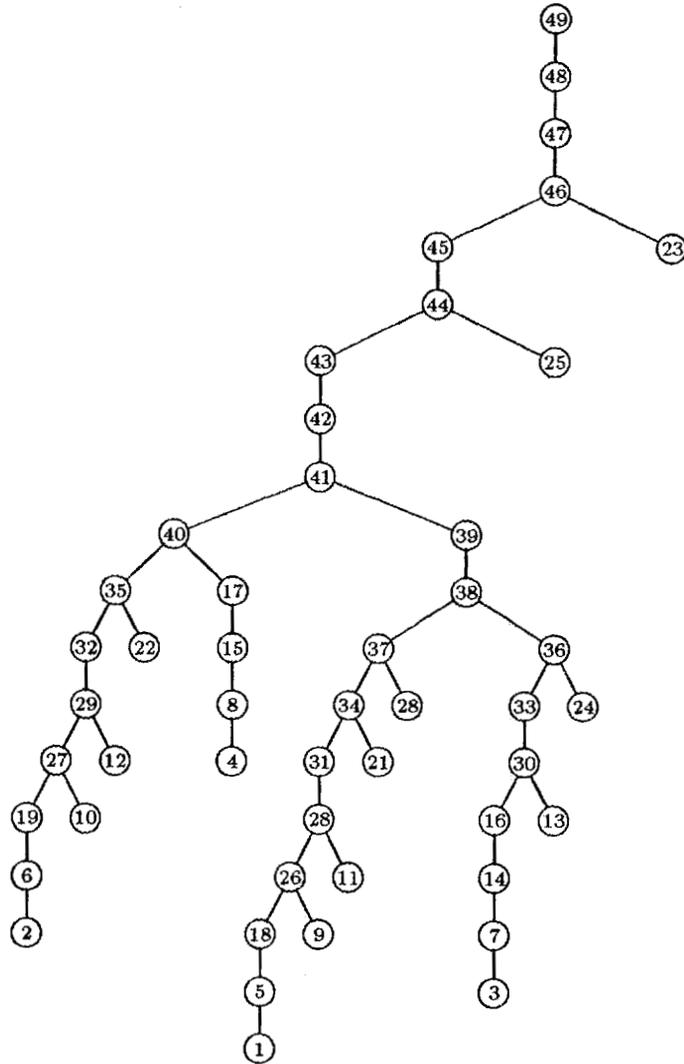


Figure 4: The elimination tree associated with a minimum degree ordering on a  $7 \times 7$  grid.

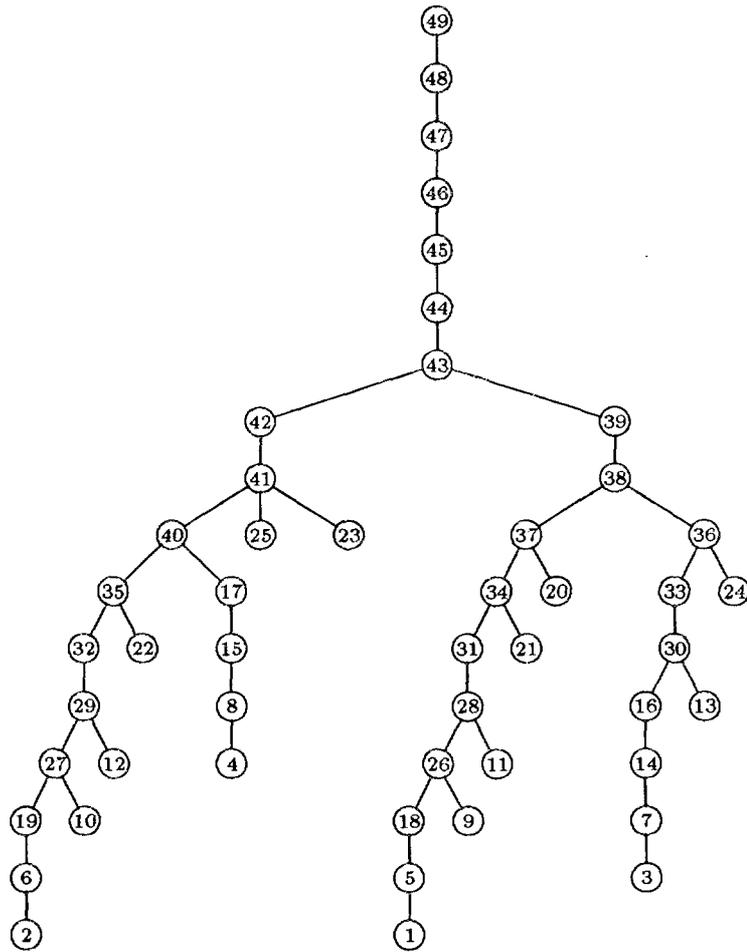


Figure 5: The elimination tree obtained when tree rotations are applied to the elimination tree in Figure 4.



structure of the elimination tree and the topology of the hypercube. Although a balanced binary tree is ideal for this method, any tree associated with a nested dissection ordering can be used. Similarly, the method assumes the number of processors,  $p$ , is a power of two, although modifications can be made to the method to avoid this restriction. Starting at the root, the method assigns the columns in the initial chain of nodes in a wrap fashion to all processors. Since the elimination tree is produced from a nested dissection ordering, the initial chain splits into two chains. These in turn split into two and so on to the bottom of the tree. The method uses this fact in its assignments. It divides the processors into two equal groups and wrap maps each of the chains to each group. The method continues recursively until only one processor remains in each of  $p$  groups. It then assigns the portion of the tree below each group to that processor. Figure 7 shows an example of subtree-to-subcube partitioning on four processors of a  $7 \times 7$  grid ordered by nested dissection. For a  $k \times k$  grid ordered by

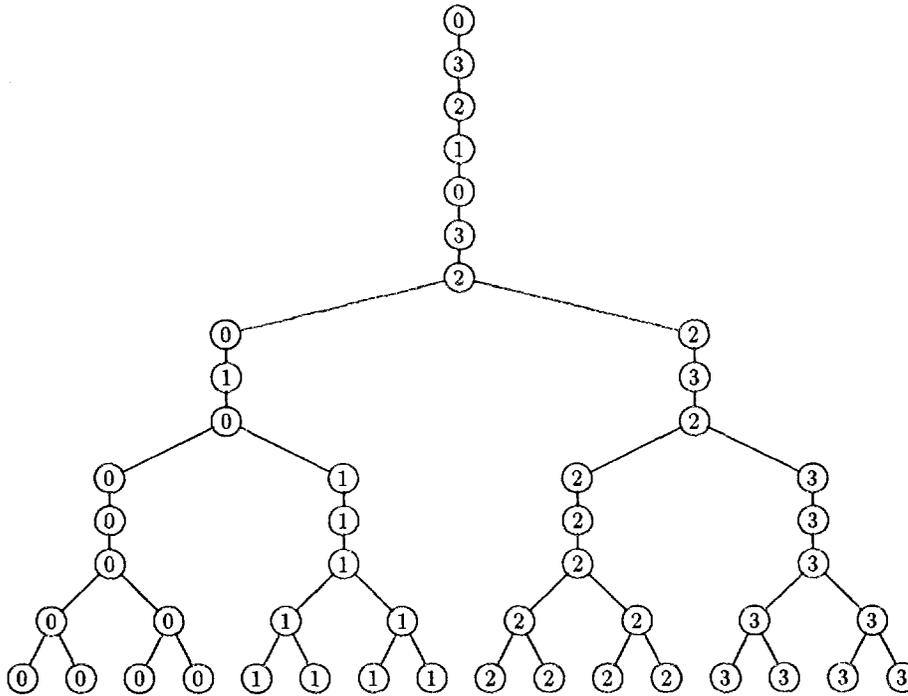


Figure 7: A subtree-to-subcube mapping for the elimination tree in Figure 3. The labelling refers to the processor to which a column is assigned.

nested dissection, George, Liu and Ng proved that the subtree-to-subcube mapping has the minimum amount of interprocessor communication [8]. The major disadvantage of

this partitioning method is that it requires a balanced elimination tree to work well. Otherwise, a large load imbalance results and performance suffers accordingly.

For wrap and sparse-wrap partitioning methods the time required to perform the backward substitution using a row-oriented data structure was sometimes as large as the time to do the factorization! This is due to the randomness of the communication pattern, which forces the parallel backward solution to behave like a sequential algorithm. Recall that in the serial case the triangular solution is an order of magnitude faster than the factorization. Using the subtree-to-subcube mapping, the communication requirement for each solution component is limited to only a subtree in the elimination tree, and thus the relative proportions of the two phases of the solution have returned to normal.

Compared to minimum degree and banded orderings, the nested dissection ordering produces the most balanced elimination trees. Since balanced trees are desired when using the subtree-to-subcube partitioning, a parallel nested dissection algorithm has been developed for use on 2-dimensional finite element problems [4]. The combination of this nested dissection algorithm and the subtree-to-subcube partitioning is compared to our new partitioning scheme in Section 5. The main drawback to the nested dissection ordering algorithm is the tendency to produce a large amount of fill and consequently a large amount of computational work for general problems. Of the three ordering methods mentioned above, minimum degree ordering algorithms often generate the least fill. Table 1 gives the amount of fill produced by Liu's multiple minimum degree algorithm (MMD) and by the parallel nested dissection algorithm described in [4]. The test problems are a sequence of finite-element problems defined on an L-shaped domain. The MMD ordering was performed on one processor and so its quality is unaffected by the number of processors used in the factorization. On the other hand, the quality of the parallel nested dissection ordering depends on the number of processors used during the ordering. This is because the graph is divided into two equal pieces recursively until  $p$  pieces are generated, then a regular nested dissection [7] is performed on the subgraph in each processor.

In general the MMD ordering produces a very unbalanced elimination tree. Liu's tree rotation algorithm can sometimes improve the balance, but seldom to the point that a subtree-to-subcube partitioning can be applied effectively (see Figures 4 and 5).

$n$	MMD	ND	
		$p = 8$	$p = 16$
2,233	46,624	50,275	50,967
2,614	57,276	61,637	63,357
3,025	67,912	73,647	75,696
3,466	83,116	86,717	88,972
3,937	95,374	101,190	103,719
4,438	111,493	116,954	119,529
4,969	125,706	134,710	136,914
5,530	146,089	151,897	155,440
6,121	164,054	173,906	175,902

Table 1: Offdiagonal nonzero counts for minimum degree ordering and nested dissection orderings.

#### 4. The Partitioning Algorithm

It is our intent to work with arbitrarily unbalanced elimination trees to allow the user flexibility in his choice of ordering methods. Our approach uses some additional properties of elimination trees. The elimination tree can be generated from the structure of the reordered matrix  $A$  before the Cholesky factor  $L$  is known [10,14]. Also in [14] is an algorithm to predict the amount of fill that will occur in  $L$  using only the structure of the reordered matrix  $A$  and the elimination tree. Given the number of nonzeros in each column and the elimination tree, it is possible to calculate the number of floating-point operations performed on each column of  $L$ . An efficient algorithm for generating these operation counts is shown in Figure 8. We will use these operation counts as nodal weights in our scheme.

Given an arbitrary tree and  $p$  processors, our task is to find the smallest set of branches such that this set can be partitioned into exactly  $p$  subsets, all of which contain approximately the same amount of work. Over this class of solutions we wish to maximize the operation counts in the set of branches.

Our strategy involves a breadth first search of the elimination tree, cutting off branches and applying a heuristic bin packing algorithm to the set of branches. The procedure is applied iteratively until the work load across all processors meets a user defined tolerance. That is, the iterative procedure will be terminated when the difference in work load between any two processors is less than a user specified parameter. Each processor is then assigned the set of branches in a particular bin. The remaining

```
for  $j := 1$  to  $n$  do
   $nz(j) := 1$ 
end for

for  $i := 1$  to  $n$  do
   $marker(i) := 0$ 
  for each  $k$  with  $a_{ik} \neq 0$  and  $k > i$  do
     $j := k$ 
    while  $marker(j) \neq i$  and  $j < i$  do
       $nz(j) := nz(j) + 1$ 
       $marker(j) := i$ 
       $j := parent(j)$ 
    end while
  end for
end for

for  $i := 1$  to  $n$  do
   $marker(i) := 0$ 
end for

for  $i := 1$  to  $n$  do
   $fcnt(i) := fcnt(i) + nz(i)$ 
  for each  $k$  with  $a_{ik} \neq 0$  and  $k > i$  do
     $j := k$ 
    while  $marker(j) \neq i$  and  $j < i$  do
       $fcnt(i) := fcnt(i) + nz(j)$ 
       $marker(j) := i$ 
       $nz(j) := nz(j) - 1$ 
       $j := parent(j)$ 
    end while
  end for
end for
```

Figure 8: Algorithm for predicting the number of flops required to generate each column of  $L$ .

nodes are assigned to all processors in a wrap around manner.

The key to this strategy is knowing how large each of the branches is without searching down it each time. The relative size of the branches determines which parts of the tree need pruning and which parts should be taken as a whole. We accomplish this by using a weighted elimination tree. Each node of the tree is given a weight corresponding to the sum of the weights of its children and the number of floating-point operations performed on that column of  $L$ . The algorithm in Figure 9 describes a method for generating these weights.

```
for  $i := 1$  to  $n$  do
   $nodewt(i) := 0$ 
end for
for  $i := 1$  to  $n$  do
   $nodewt(i) := nodewt(i) + fcnt(i)$ 
  if (  $parent(i) \neq 0$  )
     $nodewt(parent(i)) := nodewt(parent(i)) + nodewt(i)$ 
  end if
end for
```

Figure 9: Algorithm to generate tree weights.

Each node of the tree corresponds to one column of  $L$ , but the weight of a node is the sum of the floating-point operations to factor that column and the weight of all of its children. Thus, the weight of a given node is the number of floating-point operations needed to eliminate all of the nodes below and including that node. The weight of the root node in the tree is always the total number of floating-point operations required to factor the matrix.

The partitioning algorithm is shown in Figures 10 and 11. As an initial step the first  $p$  branches of the tree are selected and their weights are placed in the  $p$  bins. If the weights in the bins meet the user specified balance, then the partitioning algorithm terminates. Otherwise, the largest branch in the set of branches is searched until it splits. Assuming that the branch splits into only two smaller branches (and this is almost always the case), the number of branches in the set has increased by one and the largest weight is replaced by two smaller weights whose sum is less than the original weight. This new set of weights is placed in the  $p$  bins using the heuristic rule of placing

```
ratio := 0
ip := p - 1
While( ratio ≤ tol )
  ip := ip + 1
  partition tree into ≥ ip branches
  build heap of branch op-counts (max on top)
  initialize bin heap := 0
  While( op-heap not empty )
    add top of op-heap to top of bin-heap
    reheap bin-heap (min on top)
    delete top of op-heap and reheap
  end while
  find max bin in bin-heap
  ratio := minbin/maxbin
end while
assign nodes in bins to proc.
assign rest nodes in wrapped manner
```

Figure 10: Partitioning Algorithm.

```
if( first call )
  initialize linked list as first branches of tree
end if
While( size-of-list < ip )
  find node in list with max ops
  find next branch in subtree with root node
  add next branch and all its siblings to list
  delete node from the list
end while
return list of branches
```

Figure 11: Algorithm to partition tree.

the largest weight in the bin with the smallest sum. Finding the largest and smallest weights is accomplished using two heaps [9]. The weights of the nodes selected for partitioning are arranged into a heap with the maximum on top. Similarly, the weights of the  $p$  bins are also ordered into a heap, but with the minimum weight on top. Both heaps are updated when the maximum in the node heap is deleted from that heap and added to the minimum in the bin heap. Since the weights are organized as heaps, this updating can be done in  $\log q$  time, where  $q$  is the number of entries in a heap. (The use of heaps is one of the main reasons the partitioning algorithm is so efficient.) This process is continued until the node heap is empty. Once all the weights are assigned to a bin, the load balance criterion is again checked. If the test is not met the algorithm continues to prune the tree until it is met or until no more branches can be found. An example of a final mapping using this scheme on an unbalanced tree is shown in Figure 12. This particular mapping produces the same amount of work on each processor.

The stopping criterion can be used to trade off load balance for less communication. Much communication occurs in the top portion of the tree where it is wrap mapped. If the load balance criterion is set very strict, then the algorithm will have to search further down the tree to meet it than if the criterion is set more leniently. Subsequently, the portion of the tree that is wrap mapped is larger in the former, which induces a larger amount of communication during the factorization. Our experience is that performance is more sensitive to load balance than to the amount of communication so we have set the criterion to favor load balance. Some experiments in the next section will display this effect.

## 5. Experimental Results

This section presents some experimental results on the performance of our partitioning algorithm. The test problems were based on a sequence of finite-element problems defined on L-shaped domains. A detailed description of the problems can be found in [7]. Each of the problems was initially ordered using the MMD algorithm before applying our partitioning algorithm.

Table 2 contains the times (in seconds) for performing the Cholesky factorization on 8 and 16 processors of an Intel iPSC/2 using the sparse-wrap mapping. As mentioned in Section 2, Liu has developed an algorithm for reducing the height of elimination trees



under the constraint that no additional fill is introduced [11,13]. To determine how effective the height reducing heuristic is, we applied Liu’s algorithm to the MMD ordering and then performed the factorization. The factorization times are also reported in Table 2.

$n$	original tree		tree with reduced height	
	$p = 8$	$p = 16$	$p = 8$	$p = 16$
2,233	5.793	4.147	5.754	4.111
2,614	7.421	5.189	7.317	5.117
3,025	9.121	6.414	8.854	6.159
3,466	11.569	8.125	11.600	7.897
3,937	13.228	9.169	13.225	9.001
4,438	16.187	10.781	16.157	11.271
4,969	17.621	12.031	17.763	11.869
5,530	21.885	14.252	21.646	14.127
6,121	25.123	16.648	24.943	16.525

Table 2: Factorization times (in seconds) using MMD and sparse-wrap mapping.

The partitioning algorithm requires the user to specify a load balance criterion. Table 3 shows the effect of varying this criterion between 25% and 5% for 8 processors. For our tests it was set to 20%; that is, the partitioning algorithm terminates when

$n$	maximum load variation				
	25%	20%	15%	10%	5%
2,233	5.289	5.286	5.242	5.250	5.246
2,614	6.276	6.277	6.474	6.491	6.463
3,025	7.630	7.648	7.677	7.837	7.896
3,466	9.922	9.788	9.826	9.783	10.220
3,937	11.415	11.394	11.442	11.445	11.656
4,438	13.497	13.505	13.512	13.627	13.639
4,969	15.617	15.404	15.443	15.438	15.454
5,530	18.600	18.450	18.451	18.594	18.775
6,121	21.335	21.334	21.612	21.602	21.745

Table 3: Factorization times (in seconds) when load balance criterion is varied.

the numbers of floating-point operations on the busiest and the most idle processors differ by less than 20%. Tables 4 and 5 give the times in seconds to do the partitioning and the factorization for each of the test problems using 8 and 16 processors. The partitioning strategy was applied to the tree associated with the initial MMD ordering

and also to the tree with reduced height.

$n$	original tree		tree with reduced height	
	$p = 8$	$p = 16$	$p = 8$	$p = 16$
2,233	5.243	3.964	5.284	3.878
2,614	6.314	4.813	6.280	4.821
3,025	7.812	5.697	7.657	5.556
3,466	9.935	7.083	9.787	6.920
3,937	11.399	8.098	11.395	7.998
4,438	13.706	9.651	13.503	9.522
4,969	15.387	10.519	15.397	10.524
5,530	18.703	12.766	18.455	12.798
6,121	21.349	14.332	21.337	14.323

Table 4: Factorization times (in seconds) using MMD and partitioning strategy.

$n$	original tree		tree with reduced height	
	$p = 8$	$p = 16$	$p = 8$	$p = 16$
2,233	0.100	0.150	0.100	0.140
2,614	0.100	0.140	0.100	0.140
3,025	0.130	0.130	0.130	0.150
3,466	0.130	0.160	0.130	0.160
3,937	0.140	0.200	0.140	0.180
4,438	0.170	0.240	0.170	0.230
4,969	0.200	0.210	0.190	0.230
5,530	0.210	0.260	0.220	0.250
6,121	0.240	0.260	0.230	0.260

Table 5: Times (in seconds) to do the partitioning using MMD.

From Tables 4 and 5, we conclude that the partitioning strategy can be implemented very efficiently. The time required to determine the partitioning is negligible compared to the factorization time. Comparing the results in Tables 2 and 4, we see that the column-to-processor mapping using the partitioning strategy can be quite effective. For our set of test problems, the factorization times using the partitioning strategy are always smaller than those using the sparse-wrap mapping. The decrease in factorization times is primarily due to a reduction in the amount of communication required using the mapping from the partitioning strategy.

For comparison, we have provided in Table 6 the factorization times (in seconds)

using the parallel nested dissection ordering and the induced subtree-to-subcube mapping. Comparing Tables 4 and 6, the factorization times for the two mapping schemes

$n$	$p = 8$	$p = 16$
2,233	5.225	3.564
2,614	7.229	4.948
3,025	9.070	6.038
3,466	10.520	6.999
3,937	12.718	8.347
4,438	15.652	10.241
4,969	17.788	11.496
5,530	20.840	13.809
6,121	23.500	15.163

Table 6: Factorization times (in seconds) using a parallel nested dissection ordering and the induced subtree-to-subcube mapping.

are nearly equal. It is important to point out that, unlike the subtree-to-subcube mapping, our partitioning strategy is designed to handle an elimination tree associated with any fill-reducing ordering. While the MMD ordering appears to be much better than the parallel nested dissection ordering in terms of fill (see Table 1), the subtree-to-subcube mapping induced by parallel nested dissection attempts to minimize the communication load throughout the computation. The reduced computation in MMD and the reduced communication in subtree-to-subcube have cancelled each other in the timing comparisons. However, we now describe how to reduce the communication in our scheme.

For a multiprocessor system with  $p$  processors, our partitioning algorithm identifies  $p$  independent sets of columns and assigns one set to each processor. The remaining columns (denoted by  $S$ ) are assigned to the processors using a wrap mapping. Thus, one drawback about our approach is that the remaining set of columns  $S$  may be large. Consequently, the amount of communication required to process the columns in  $S$  may be large. One way to reduce this communication overhead at the possible expense of disrupting the load balance is to apply the subtree-to-subcube mapping to  $S$ . The algorithm incorporating this idea can be described as follows. We first apply the partitioning strategy to the elimination tree to identify the independent sets of columns that are assigned to the processors. We then work back up the tree starting at the roots of these subtrees. In order to implement this idea, we keep a list of

processors for each path in  $S$ . Initially, a list contains only the processor assigned to the subtree. As we move up each path, the nodes are assigned to processors in that list in a wrapped fashion. When multiple paths coalesce, their lists are merged. When continued to the root of the tree, this scheme will produce a subtree-to-subcube mapping of  $S$ . However, the overall algorithm is more complicated, and its efficient implementation is under investigation.

## 6. References

- [1] G. A. Geist and M. T. Heath. *Parallel Cholesky factorization on a hypercube multiprocessor*. Technical Report ORNL-6211, Oak Ridge National Laboratory, Oak Ridge, Tennessee, 1985.
- [2] J. A. George. Nested dissection of a regular finite element mesh. *SIAM J. Numer. Anal.*, 10:345-363, 1973.
- [3] J. A. George, M. T. Heath, J. W-H. Liu, and E. G-Y. Ng. Solution of sparse positive definite systems on a shared memory multiprocessor. *Internat. J. Parallel Programming*, 15:309-325, 1986.
- [4] J. A. George, M. T. Heath, J. W-H. Liu, and E. G-Y. Ng. *Solution of sparse positive definite systems on a hypercube*. Technical Report ORNL/TM-10865, Oak Ridge National Laboratory, 1988.
- [5] J. A. George, M. T. Heath, J. W-H. Liu, and E. G-Y. Ng. Sparse Cholesky factorization on a local-memory multiprocessor. *SIAM J. Sci. Stat. Comput.*, 9:327-340, 1988.
- [6] J. A. George and J. W-H. Liu. An automatic nested dissection algorithm for irregular finite element problems. *SIAM J. Numer. Anal.*, 15:1053-1069, 1978.
- [7] J. A. George and J. W-H. Liu. *Computer Solution of Large Sparse Positive Definite Systems*. Prentice-Hall Inc., Englewood Cliffs, New Jersey, 1981.
- [8] J. A. George, J. W-H. Liu, and E. G-Y. Ng. Communication results for parallel sparse Cholesky factorization on a hypercube. *Parallel Computing*, 1988. (to appear).

- [9] D. E. Knuth. *The art of computer programming, volume 3: Sorting and searching*. Addison-Wesley, Reading, Mass., 1973.
- [10] J. W-H. Liu. A compact row storage scheme for Cholesky factors using elimination trees. *ACM Trans. on Math. Software*, 12:127-148, 1986.
- [11] J. W-H. Liu. Equivalent sparse matrix reordering by elimination tree rotations. *SIAM J. Sci. Stat. Comput.*, 9:424-444, 1988.
- [12] J. W-H. Liu. *A graph partitioning algorithm by node separators*. Technical Report CS-88-01, Dept. of Computer Science, York University, Downsview, Ontario, 1988.
- [13] J. W-H. Liu. *Reordering sparse matrices for parallel elimination*. Technical Report CS-87-01, Dept. of Computer Science, York University, 1987.
- [14] J. W-H. Liu. *The role of elimination trees in sparse factorization*. Technical Report CS-87-12, Dept. of Computer Science, York University, 1987.

**INTERNAL DISTRIBUTION**

- |        |                   |        |  |
|--------|-------------------|--------|--|
| 1.     | B. R. Appleton    | 32.    | P. H. Worley   |
| 2.     | J. B. Drake       | 33.    | A. Zucker  |
| 3-7.   | G. A. Geist       | 34.    | J. J. Dorning (Consultant)                           |
| 8-9.   | R. F. Harbison    | 35.    | R. M. Haralick (Consultant)                          |
| 10.    | M. T. Heath       | 36.    | Central Research Library                             |
| 11-15. | J. K. Ingersoll   | 37.    | ORNL Patent Office                                   |
| 16.    | M. R. Leuze       | 38.    | K-25 Plant Library                                   |
| 17-21. | F. C. Maienschein | 39.    | Y-12 Technical Library<br>Document Reference Station |
| 22-24. | E. G. Ng          | 40.    | Laboratory Records - RC                              |
| 25.    | G. Ostrouchov     | 41-42. | Laboratory Records Department                        |
| 26.    | C. H. Romine      |        |  |
| 27-31. | R. C. Ward        |        |  |

**EXTERNAL DISTRIBUTION**

43. Dr. Donald M. Austin, Office of Scientific Computing, Office of Energy Research, ER-7, Germantown Building, U.S. Department of Energy, Washington, DC 20545
44. Dr. Robert G. Babb, Department of Computer Science and Engineering, Oregon Graduate Center, 19600 N.W. Walker Road, Beaverton, OR 97006
45. Lawrence J. Baker, Exxon Production Research Company, P.O. Box 2189, Houston, TX 77252-2189
46. Dr. Jesse L. Barlow, Department of Computer Science, Pennsylvania State University, University Park, PA 16802
47. Dr. Chris Bischof, Mathematics and Computer Science Division, Argonne National Laboratory, 9700 South Cass Avenue, Argonne, IL 60439
48. Prof. Ake Bjorck, Department of Mathematics, Linkoping University, Linkoping 58183, Sweden

49. Dr. James C. Browne, Department of Computer Sciences, University of Texas, Austin, TX 78712
50. Dr. Bill L. Buzbee, Scientific Computing Division, National Center for Atmospheric Research, P.O. Box 3000, Boulder, CO 80307
51. Dr. Donald A. Calahan, Department of Electrical and Computer Engineering, University of Michigan, Ann Arbor, MI 48109
52. Dr. Tony Chan, Department of Mathematics, University of California, Los Angeles, 405 Hilgard Avenue, Los Angeles, CA 90024
53. Dr. Jagdish Chandra, Army Research Office, P.O. Box 12211, Research Triangle Park, North Carolina 27709
54. Dr. Eleanor Chu, Department of Computer Science, University of Waterloo, Waterloo, Ontario, Canada N2L 3G1
55. Prof. Tom Coleman, Department of Computer Science, Cornell University, Ithaca, NY 14853
56. Dr. Paul Concus, Mathematics and Computing, Lawrence Berkeley Laboratory, Berkeley, CA 94720
57. Prof. Andy Conn, Department of Combinatorics and Optimization, University of Waterloo, Waterloo, Ontario, Canada N2L 3G1
58. Dr. Jane K. Cullum, IBM T. J. Watson Research Center, P.O. Box 218, Yorktown Heights, NY 10598
59. Dr. George Cybenko, Computer Science Department, University of Illinois, Urbana, IL 61801
60. Dr. George J. Davis, Department of Mathematics, Georgia State University, Atlanta, GA 30303
61. Dr. Jack J. Dongarra, Mathematics and Computer Science Division, Argonne National Laboratory, 9700 South Cass Avenue, Argonne, IL 60439

62. Dr. Iain Duff, CSS Division, Harwell Laboratory, Didcot, Oxon OX11 0RA, England
63. Prof. Pat Eberlein, Department of Computer Science, SUNY/Buffalo, Buffalo, NY 14260
64. Dr. Stanley Eisenstat, Department of Computer Science, Yale University, P.O. Box 2158 Yale Station, New Haven, CT 06520
65. Dr. Lars Elden, Department of Mathematics, Linkoping University, 581 83 Linkoping, Sweden
66. Dr. Howard C. Elman, Computer Science Department, University of Maryland, College Park, MD 20742
67. Dr. Albert M. Erisman, Boeing Computer Services, 565 Andover Park West, Tukwila, WA 98188
68. Dr. Peter Fenyves, General Motors Research Laboratory, Department 15, GM Technical Center, Warren, MI 48090
69. Prof. David Fisher, Department of Mathematics, Harvey Mudd College, Claremont, CA 91711
70. Dr. Geoffrey C. Fox, Booth Computing Center 158-79, California Institute of Technology, Pasadena, CA 91125
71. Dr. Paul O. Frederickson, Computing Division, Los Alamos National Laboratory, Los Alamos, NM 87545
72. Dr. Fred N. Fritsch, L-300, Mathematics and Statistics Division, Lawrence Livermore National Laboratory, P.O. Box 808, Livermore, CA 94550
73. Dr. Robert E. Funderlic, Department of Computer Science, North Carolina State University, Raleigh, NC 27650
74. Dr. Dennis B. Gannon, Computer Science Department, Indiana University, Bloomington, IN 47405

75. Dr. David M. Gay, Bell Laboratories, 600 Mountain Avenue, Murray Hill, NJ 07974
76. Dr. C. William Gear, Computer Science Department, University of Illinois, Urbana, Illinois 61801
77. Dr. W. Morven Gentleman, Division of Electrical Engineering, National Research Council, Building M-50, Room 344, Montreal Road, Ottawa, Ontario, Canada K1A 0R8
78. Dr. Alan George, Vice President, Academic and Provost, Needles Hall, University of Waterloo, Waterloo, Ontario, Canada N2L 3G1
79. Dr. John Gilbert, Xerox Palo Alto Research Center 3333 Coyote Hill Road Palo Alto, CA 94304
80. Dr. Jacob D. Goldstein, The Analytic Sciences Corporation, 55 Walkers Brook Drive, Reading, MA 01867
81. Prof. Gene H. Golub, Department of Computer Science, Stanford University, Stanford, CA 94305
82. Dr. Joseph F. Grcar, Division 8331, Sandia National Laboratories, Livermore, CA 94550
83. Dr. Per Christian Hansen, Copenhagen University Observatory, Øster Voldgade 3, DK-1350 Copenhagen K, Denmark
84. Dr. Don E. Heller, Physics and Computer Science Department, Shell Development Co., P.O. Box 481, Houston, TX 77001
85. Dr. F. J. Helton, GA Technologies, P.O. Box 81608, San Diego, CA 92188
86. Dr. Charles J. Holland, Air Force Office of Scientific Research, Building 410, Bolling Air Force Base, Washington, DC 20332
87. Dr. Robert E. Huddleston, Computation Department, Lawrence Livermore National Laboratory, P.O. Box 808, Livermore, CA 94550

88. Dr. Ilse Ipsen, Department of Computer Science, Yale University, P.O. Box 2158  
Yale Station, New Haven, CT 06520
89. Ms. Elizabeth Jessup, Department of Computer Science, Yale University, P.O. Box  
2158, Yale Station, New Haven, CT 06520
90. Prof. Barry Joe, Department of Computer Science, University of Alberta, Ed-  
monton, Alberta, Canada T6G 2H1
91. Dr. Harry Jordan, Department of Electrical and Computer Engineering, Univer-  
sity of Colorado, Boulder, CO 80309
92. Dr. Bo Kagstrom, Institute of Information Processing, University of Umea, 5-901  
87 Umea, Sweden
93. Dr. Hans Kaper, Mathematics and Computer Science Division, Argonne National  
Laboratory, 9700 South Cass Avenue, Argonne, IL 60439
94. Dr. Linda Kaufman, Bell Laboratories, 600 Mountain Avenue, Murray Hill, NJ  
07974
95. Dr. Robert J. Kee, Applied Mathematics Division 8331, Sandia National Labo-  
ratories, Livermore, CA 94550
96. Ms. Virginia Klema, Statistics Center, E40-131, MIT, Cambridge, MA 02139
97. Dr. Richard Lau, Office of Naval Research, 1030 E. Green Street, Pasadena, CA  
91101
98. Dr. Alan J. Laub, Department of Electrical and Computer Engineering, Univer-  
sity of California, Santa Barbara, CA 93106
99. Dr. Robert L. Launer, Army Research Office, P.O. Box 12211, Research Triangle  
Park, North Carolina 27709
100. Dr. Charles Lawson, Applied Mathematics Group, Jet Propulsion Laboratory,  
California Institute of Technology, M/S 506-232, 4800 Oak Grove Drive, Pasadena,  
CA 91103

101. Prof. Peter D. Lax, Director, Courant Institute of Mathematical Sciences, New York University, 251 Mercer Street, New York, NY 10012
102. Dr. John G. Lewis, Boeing Computer Services, P.O. Box 24346, M/S 7L-21, Seattle, WA 98124-0346
103. Dr. Heather M. Liddell, Director, Center for Parallel Computing, Department of Computer Science and Statistics, Queen Mary College, University of London, Mile End Road, London E1 4NS, England
104. Dr. Joseph Liu, Department of Computer Science, York University, 4700 Keele Street, Downsview, Ontario, Canada M3J 1P3
105. Dr. Franklin Luk, Electrical Engineering Department, Cornell University, Ithaca, NY 14853
106. James G. Malone, General Motors Research Laboratories, Warren, Michigan 48090-9055
107. Dr. Thomas A. Manteuffel, Computing Division, Los Alamos National Laboratory, Los Alamos, NM 87545
108. Dr. Bernard McDonald, National Science Foundation, 1800 G Street, NW, Washington, DC 20550
109. Dr. Paul C. Messina, California Institute of Technology, Mail Code 158-79, Pasadena, CA 91125
110. Dr. Cleve Moler, Ardent Computers, 550 Del Ray Avenue, Sunnyvale, CA 94086
111. Dr. Brent Morris, National Security Agency, Ft. George G. Meade, MD 20755
112. Dr. Dianne P. O'Leary, Computer Science Department, University of Maryland, College Park, MD 20742
113. Maj. C. E. Oliver, Office of the Chief Scientist, Air Force Weapons Laboratory, Kirtland Air Force Base, Albuquerque, NM 87115
114. Dr. James M. Ortega, Department of Applied Mathematics, University of Virginia, Charlottesville, VA 22903

115. Prof. Chris Paige, Department of Computer Science, McGill University, 805 Sherbrooke Street W., Montreal, Quebec, Canada H3A 2K6
116. Dr. John F. Palmer, NCUBE Corporation, 915 E. LaVieve Lane, Tempe, AZ 85284
117. Prof. Roy P. Pargas, Department of Computer Science, Clemson University, Clemson, SC 29634-1906
118. Prof. Beresford N. Parlett, Department of Mathematics, University of California, Berkeley, CA 94720
119. Prof. Merrell Patrick, Department of Computer Science, Duke University, Durham, NC 27706
120. Dr. Robert J. Plemmons, Departments of Mathematics and Computer Science, North Carolina State University, Raleigh, NC 27650
121. Dr. Alex Pothén, Department of Computer Science, Pennsylvania State University, University Park, PA 16802
122. Dr. John K. Reid, CSS Division, Building 8.9, AERE Harwell, Didcot, Oxon, England OX11 0RA
123. Dr. John R. Rice, Computer Science Department, Purdue University, West Lafayette, IN 47907
124. Dr. Garry Rodrigue, Numerical Mathematics Group, Lawrence Livermore Laboratory, Livermore, CA 94550
125. Dr. Donald J. Rose, Department of Computer Science, Duke University, Durham, NC 27706
126. Dr. Ahmed H. Sameh, Computer Science Department, University of Illinois, Urbana, IL 61801
127. Dr. Michael Saunders, Systems Optimization Laboratory, Operations Research Department, Stanford University, Stanford, CA 94305

128. Dr. Robert Schreiber, Department of Computer Science, Rensselaer Polytechnic Institute, Troy, NY 12180
129. Dr. Martin H. Schultz, Department of Computer Science, Yale University, P.O. Box 2158 Yale Station, New Haven, CT 06520
130. Dr. David S. Scott, Intel Scientific Computers, 15201 N.W. Greenbrier Parkway, Beaverton, OR 97006
131. Dr. Lawrence F. Shampine, Mathematics Department, Southern Methodist University, Dallas, TX 75275
132. Dr. Kermit Sigmon, Department of Mathematics, University of Florida, Gainesville, FL 32611
133. Dr. Danny C. Sorensen, Mathematics and Computer Science Division, Argonne National Laboratory, 9700 South Cass Avenue, Argonne, IL 60439
134. Prof. G. W. Stewart, Computer Science Department, University of Maryland, College Park, MD 20742
135. Dr. Kosmo D. Tatalias, Atlantic Aerospace Electronics Corporation, 6404 Ivy Lane, Suite 300, Breenbelt, MD 20770-1406
136. Prof. Charles Van Loan, Department of Computer Science, Cornell University, Ithaca, NY 14853
137. Dr. Robert G. Voigt, ICASE, MS 132-C, NASA Langley Research Center, Hampton, VA 23665
138. Dr. Andrew B. White, Computing Division, Los Alamos National Laboratory, Los Alamos, NM 87545
139. Dr. Arthur Wouk, Army Research Office, P.O. Box 12211, Research Triangle Park, North Carolina 27709
140. Dr. Margaret Wright, Bell Laboratories, 600 Mountain Avenue, Murray Hill, NJ 07974

141. Dr. A. Yerebin, Department of Numerical Mathematics of the USSR Academy of Sciences, Gorki Street 11, Moscow, 103905, USSR
142. Office of Assistant Manager for Energy Research and Development, U.S. Department of Energy, Oak Ridge Operations Office, P.O. Box 2001 Oak Ridge, TN 37831-8600
- 143-152. Office of Scientific & Technical Information, P.O. Box 62, Oak Ridge, TN 37831