(2)

NASA Contractor Report 189634
ICASE Report No. 92-13

ICASE

A PERFORMANCE STUDY OF SPARSE CHOLESKY FACTORIZATION ON INTEL iPSC/860

M. Zubair
M. Ghose

Contract No. NAS1-18605
March 1992

Institute for Computer Applications in Science and Engineering
NASA Langley Research Center
Hampton, Virginia 23665-5225

Operated by the Universities Space Research Association

National Aeronautics and
Space Administration

Langley Research Center
Hampton, Virginia 23665-5225

A PERFORMANCE STUDY OF SPARSE CHOLESKY FACTORIZATION ON INTEL iPSC/860

M. Zubair[1] and M. Ghose
Department of Computer Science
Old Dominion University
Norfolk, VA

ABSTRACT

The problem of Cholesky factorization of a sparse matrix has been very well investigated on sequential machines. A number of efficient codes exist for factorizing large unstructured sparse matrices, for example, codes from Harwell Subroutine Library [4] and Sparspak [7]. However, there is a lack of such efficient codes on parallel machines in general, and distributed memory machines in particular. Some of the issues which are critical to the implementation of sparse Cholesky factorization on a distributed memory parallel machine are: *ordering*, *partitioning and mapping*, *load balancing*, and *ordering of various tasks* within a processor. Addressing these issues optimally for unstructured sparse matrices is a challenging task. In this paper we focus on the effect of various partitioning schemes on the performance of sparse Cholesky factorization on the INTEL iPSC/860. We also propose a new partitioning heuristic for structured as well as unstructured sparse matrices, and compare its performance with the other schemes.

[1]Research was supported in part by the National Aeronautics and Space Administration under NASA Contract No. NAS1-18605 while the author was in residence at the Institute for Computer Applications in Science and Engineering (ICASE), NASA Langley Research Center, Hampton, VA 23665.

1 Introduction

The problem of Cholesky factorization of a sparse matrix has been very well investigated on sequential machines. A number of efficient codes exist for factorizing large unstructured sparse matrices, for example, codes from Harwell Subroutine Library [4] and Sparspak [7]. However, there is a lack of such efficient codes on parallel machines in general, and distributed memory machines in particular. This is partly because these machines are relatively new and there is not much experience to solve unstructured problems on these machines. However, there has been reasonable success in putting unstructured Euler codes on a distributed memory parallel machine [3]. (For these codes, in contrast to unstructured sparse factorization codes, ordering of various tasks such as computation and communication, within each processor is not an issue.)

Some of the issues which are critical to the implementation of sparse Cholesky factorization on a distributed memory parallel machine are: *ordering*, *partitioning and mapping*, *load balancing*, and *ordering of various tasks* within a processor. Addressing these issues optimally is a challenging task. For example, it is not clear what is a good ordering scheme for parallel factorization. (Recently, reordering schemes for parallel factorization have been suggested in the literature, for example [12]. But, we are not aware of any performance figures for these orderings on an actual distributed memory parallel machine.) The problem becomes more complex because a solution obtained at a step may influence the solution at the next step. For example, an ordering which minimizes fill may lead to an unbalanced load.

In the past, some attempts have been made to implement the Cholesky factorization for structured sparse matrices on the INTEL iPSC/2. George et al. have described an implementation of the fan-out algorithm [9]. Recently, Ashcraft et al. [1] have presented a compute-ahead implementation of fan-in the algorithm. Although, relative performance of their implementation is better than the basic fan-in and fan-out implementation, the absolute performance is far from desirable. For example, the factorization time reported for a 75×75 grid problem using compute ahead fan-in is 1.561 seconds on a 64-processor machine. In megaflops this is approximately 0.075 *mflops* per node. This is significantly low when one considers the performance of the existing sequential codes on RISC based workstations. The MA27 code on IBM RS/6000, 41MHz machine, gives 11 MFlops for medium sized matrices [15]. Ashcraft et al. [2] have compared the communication requirement of distributed multifrontal schemes with the fan-out and fan-in schemes. It should be noted that most of the above mentioned studies have been done for structured sparse matrices arising from regular grids. For these problems, ordering, partitioning, and load balancing do not pose a significant problem. Nested dissection is used for ordering these problems as it gives optimal-order fill and well-balanced elimination trees [7]. The partitioning scheme suggested in [8] for regular grid problems results in good load balancing and low communication cost. For unstructured sparse matrices not many results have been reported.

Venugopal and Naik have recently studied partitioning and scheduling methodology for unstructured sparse matrix factorization on distributed memory machines [13]. However, they do not report any performance results of their studies on an actual machine.

We feel that there is much more to be done before one can possibly get good performance for sparse Cholesky factorization on a distributed memory parallel machine. Our effort is a step in that direction. In this paper, we study the effect of various partitioning schemes on the performance of sparse Cholesky factorization on the INTEL iPSC/860. We also propose a new partitioning heuristic for structured as well as unstructured sparse matrices, and compare its performance with the other schemes. The distributed factorization algorithm which has been implemented is a variation of the distributed fan-out algorithm. The distributed fan-out algorithm is known to have greater interprocessor communication costs than the other distributed algorithms [11]. We still selected this algorithm because (i) it is simple to implement, and (ii) the focus of our research was to study the effect of partitioning on the performance of factorization.

The rest of the paper is organized as follows. In the next section we review the basic Cholesky algorithm for solving a linear system of equations. Section 3 gives a brief description of various partitioning schemes including the proposed heuristic. In Section 4 we briefly describe the implementation of factorization algorithm. The experimental results are discussed in Section 5. Finally in Section 6 we give the conclusions.

2 Background

Consider a system of linear equations,

$$Ax = b,$$

where A is an $n \times n$ symmetric positive definite matrix, b is a known vector and x is the unknown vector to be computed. One way to solve such linear systems is to compute the Cholesky factorization of matrix A,

$$A = LL^T,$$

where L is a lower triangular matrix. Then x is computed by solving the triangular systems

$$Ly = b, and$$

$$L^T x = y.$$

We now briefly discuss the basic steps involved in the solution of such a system on a distributed memory parallel machine.

(i) *Ordering :* Find an ordering P of the sparse matrix A so that the Cholesky factor L of PAP^T suffers little fill and at the same time reduces the parallel time.

(ii) *Preprocessing:* It consists of three parts. The first part is to determine the structure of L. The second part is to obtain the partitioning and mapping, that is the distribution of the columns of A amongst various processors of the machine. The third part is to create the required data structures for each processor for the numeric factorization step.

(iii) *Numeric Factorization:* Compute the Cholesky factor L of PAP^T.

(iv) *Triangular Solution:* Solve $Ly = Pb$ and $L^T z = y$, and then set $x = P^T z$.

In this paper, we have focussed on the effect of partitioning on the performance of numeric factorization. We discuss them in a little more detail.

Partitioning. Given a graph G of n nodes associated with the $L + L^T$ matrix, find p ($n >> p$) partitions of the graph with (i) large number of intra-partition edges and very few inter-partition edges, and (ii) nearly equal computational load for each partition.

Numeric Factorization. The distributed algorithms proposed in the literature are based on the column-oriented Cholesky factorization. Following [8, 14], the basic column-oriented algorithm can be expressed as

```
begin
  for j = 1 to n do
  begin
     for k = 1 to j-1 do
        cmod(j,k)
     cdiv(j)
  end
end
```

where,
cmod(j,k) : is modification of column j by column k ($k < j$), and is also referred as the update computation.
cdiv(j) : is division of column j by a scalar, and is also referred as the factorize computation.

The three basic distributed algorithms reported in the literature are distributed fan-in, distributed fan-out and distributed multifrontal. We do not discuss these algorithms here. For a detailed discussion of these algorithms along with their advantages and disadvantages, one may refer to [11]. For this paper, we have implemented a variation of the distributed fan-out algorithm.

3 Partitioning Schemes

In this section we first briefly discuss some of the existing partitioning schemes which have been investigated in this paper, and finally we describe a new partitioning heuristic for structured as well as unstructured sparse matrices.

3.1 Subtree-to-Subcube

This scheme was suggested by George et al. [8] for regular grid problems. An example illustrating the partitioning and mapping of a 7×7 grid is shown in Figure 1. For details one can refer to [8].

3.2 Contiguous column

In this scheme contiguous columns of a sparse matrix are assigned to a partition such that there is a uniform distribution of columns amongst various partitions.

3.3 Contiguous column with uniform operation count

In this scheme contiguous columns of a sparse matrix are assigned to a partition such that the number of operations required to factorize columns in a partition is nearly equal for all the partitions.

3.4 Wrap-around

In this scheme ith column of a sparse matrix is assigned to $(i-1) \bmod p$ partition, where p is the total number of partitions.

3.5 Heuristic

We first describe how to form a single partition using this scheme. Consider the Cholesky factorization of a given sparse symmetric positive definite matrix A into LL^T. Assume that the matrix A has already been ordered by some permutation. Let $G(F)$ be the graph associated with $F = L + L^T$. In the discussion here and the rest of the paper also, we use the term node of a graph G and a column of sparse matrix A interchangeably. Pick a node (column) of G, which does not depend on any other nodes (columns) for factorization, as the first node of the partition. We now do a breadth first search on G starting with this node. At any level of the breadth first search we have a set of visited nodes. Out of this set we select only those nodes which satisfy some criterion. The rest of the nodes are rejected, and the breadth first search is continued from the selected set of nodes. This process can be viewed as *pruning* of the breadth first search tree. We stop when either the computation load corresponding to the partition reaches a fixed threshold, or there are no more unvisited nodes.

Before forming the next partition, we mark all the nodes which were selected for the previous partition as visited. Note that all the nodes which were rejected are marked unvisited. In case a starting node (which does not depend on any other nodes for factorization) is not found for the new partition, an arbitrary node from the unvisited nodes is selected. It is possible that after forming all the partitions there are some free nodes, that is nodes which have not been included in any partition. These nodes are distributed such that a free node is assigned to a partition which has most of its neighbors.

Selection criterion. A node is included in the partition if most of its neighbors have already been included in the partition. The selection criterion is made stronger with the addition of more nodes in the partition. Consequently, at later stages of the partition formation, fewer and fewer nodes from a visited set of nodes are selected. Concretely, a node i is included in the partition if

$$\frac{indeg(i)}{outdeg(i)} > \alpha \frac{opc}{mop} \tag{1}$$

where,

indeg(i).: for a given partition it is the number of nodes within the partition that are adjacent to node i,

outdeg(i).: for a given partition it is the number of nodes outside the partition that are adjacent to node i,

opc.: is the number of *flops* associated with the current partition,

mop.: is the number of *flops* needed to factorize the complete sparse matrix divided by the number of partitions, and

α: is a tunable parameter with value greater than equal to zero.

Note that for $\alpha = 0$ there is no pruning of the breadth first search tree. We illustrate the partitioning heuristic with the help of an example. Consider the graph $G(F)$ of Figure 2a. We indicate the operation count associated with a node i by $c(i)$. The total operation count for the example of Figure 2a is 75. Let us assume that we are interested in forming two partitions. Thus the value of *mop* in Eq.(1) is 38. The partition formation by the heuristic scheme with $\alpha = 0$ is illustrated in Figure 2b. During the formation of the first partition we pick node 1 as the starting node. Note that node 1 does not depend on any other node for factorization. The operation count associated with this node is 3. Thus the value of *opc* (which is the number of floating point operations corresponding to nodes currently in the partition) is initialized to 3. At the next level we select nodes 4 and 7 (See Eq. (1)). The value of *opc* at this point becomes 24. In the next level, we first select node 5 which makes *opc* = 33, and then node 6 is selected which takes the value of *opc* to 44. Since the value of *opc* at this point is greater than the value of *mop*, the formation of first partition is stopped. Similarly the second partition is formed starting with node 2.

4 Implementation

Our factorization scheme can be considered as a compute-ahead implementation of the fan-out algorithm. In our algorithm description, which follows next, we make a distinction between two types of computations for a typical column at a processor. The first is the *update* computation which is the modification of a column by other columns, and the other is the *factorize* computation which is the division of a column by a scalar. The factorize computation is done on a column when it is completely updated by all the required columns. The complete algorithm can be best explained informally by considering a ready queue at each processor. The ready queue is initialized with the column numbers which do not require any update, and are ready for factorization. The program at a node can be described as follows.

```
Algorithm: Distributed factorizaton
10    while ready queue is not empty do
          • factorize the first ready column available in the queue.
          • send the factorized column to off-processors.
          • update the required local columns by the factorized column.
          • update the ready queue, that is a column which has been
          completely updated is inserted in the ready queue.
      end while
```

```
if all local columns have been factorized and sent then stop.
if a column is received then
    • update the dependent local columns.
    • update the ready queue.
end if
go to 10.
```

The detailed code for the implementation can be found in [16].

5 Experimental Results

We did a series of experiments to evaluate the performance of various partitioning schemes on INTEL iPSC/860. The experiments were done on both structured and unstructured sparse matrices. Table 1 lists three structured matrices arising from nine-point finite difference operators on square grids. In the first set of experiments, we evaluated the performance of five different partitioning schemes for these matrices. The five partitioning schemes are: (i) wrap around (*wr*), (ii) contiguous column with uniform distribution of columns (*cc*1), (iii) contiguous column with uniform distribution of operation counts (*cc*2), (iv) subtree-to-subcube (*ss*), and (v) heuristic (*hr*). For all the schemes, except subtree-to-subcube, we used the minimum degree ordering given in sparspak [7] for ordering the sparse matrices. For subtree-to-subcube partitioning the matrices were ordered using nested dissection ordering [7]. Table 2 summarizes the total factorization time in seconds (*tt*) for different partitioning schemes for three types of matrices with varying number of processors. It is clear from the table that the performance of *wr* and *cc*1 is not comparable with the other three schemes. This can be explained by further examining the performance of a 75×75 grid problem on a 4-processor machine. We observed the distribution of *operation count*, *computation* time (the time spent on computation on a processor), and *total time* on all the four processors. These observations are tabulated in Tables 3a, 3b and 3c. It is interesting to observe from these tables that the reason for the bad performance of *wr* is different from the one for *cc*1. The *wr* scheme results in uniform distribution of operation counts, but the time spent in communication is relatively greater. On the other hand, the *cc*1 scheme results in non-uniform distribution of operation counts which is mainly responsible for the its bad performance.

Another observation can be made from Table 2, that is, as the number of processors is increased the performance of *cc*2 and *hr* becomes better than that of *ss*. The worse performance of *ss* could also be due to the use of distributed fan-out algorithm for numeric factorization. The distributed fan-out algorithm is known for not exploiting the subtree-to-subcube mapping effectively [11]. We also give the performance of various schemes in *mflops* on a 16 processor machine (see Table 4). For 100×100 grid problem we obtained 9.65 *mflops* which is around 0.6 *mflops* per processor.

The second set of experiments was done on unstructured matrices from Harwell Boeing Collection [5] (see Table 5). For these matrices we compare the performance of *cc*2 with *hr*. The results are summarized in Table 6. For each scheme, we have listed the computation time (*ct*), and the total time (*tt*). We have also listed the performance in *mflops* for the two schemes. We observe that the performance of *hr* is better than that of *cc*2. To understand this behavior, we observed the distribution of operation counts, computation time, and total time over the processors of the machine. We summarize these results in Table 7. For both the schemes we have listed the standard deviation in,
(i) operation count as a fraction of average operation count on a processor (*sdopc*),
(ii) computation time as a fraction of average computation time on a processor (*sdct*), and
(iii) total time as a fraction of average total time on a processor (*sdtt*).
It is obvious from this table that both the schemes result in balanced computation and communication. The performance of *cc*2 is bad because it results in large volume of communication traffic as compared to the *hr* scheme.

6 Conclusion

In this paper, we have studied the effect of various partitioning schemes on the performance of sparse factorization on INTEL iPSC/860 for structured as well as unstructured sparse matrices. We show that the proposed partitioning heuristic works for both structured and unstructured sparse matrices. The absolute performance of the factorization step was not that impressive. We believe it can be improved by implementing a distributed algorithm which (i) maximize the performance at each processor by exploiting the cache behavior, and (ii) orders the computations at each proc ·ssor to minimize the c·mmunication overheads. One such algorithm, in our opinion, is a distributed multifrontal scheme.

References

[1] C. Ashcraft, S.C. Eisenstat, J.W.H. Liu, B.W. Peyton, A.H. Sherman, "A Compute-Ahead Implementation of the Fan-In Sparse Distributed Factorization Scheme," *Tech. Report CS-90-03*, Dept of Computer Science, York Univ., 1990.

[2] C. Ashcraft, S.C. Eisenstat, J.W.H. Liu, B.W. Peyton, A.H. Sherman, "A Comparison of Three Column-based Distribution Sparse Factorization Schemes," *Research Report YALEU/DCS/RR-810*, Yale University, 1990

[3] R. Das, D. Mavriplis, J. Saltz, S. Gupta and R. Ponnusamy, " The Design and Implementation of a Parallel Unstructured Euler Solver using Software Primitives," AIAA Paper Number 92-0562, 1992.

[4] I.S. Duff, A.M. Erisman and J.K. Reid, "Direct Methods for Sparse Matrices", Clarendon Press Oxford, 1986.

[5] I. S. Duff, R. G. Grimes and J. G. Lewis, "Sparse Matrix Test Problems", *ACM Trans. on Mathematical Software*, Vol 15, No. 1, pp. 1-14, 1989.

[6] J. A. George, "Nested dissection of a regular finite element mesh," *SIAM J. Numer. Anal.*, 10, 1973.

[7] A. George and J. Liu, "*Computer Solution of Large Sparse Positive Definite Systems*", Prentice Hall, 1981.

[8] A. George, J. Liu and E. Ng, "Communication reduction in parallel sparse Cholesky factorization on a hypercube. In M. T. Heath, editor," *Hypercube Multiprocessors*, 576-586, SIAM Press, 1987.

[9] A. George, M. Heath, J.W. Liu and E. Ng, "Sparse Cholesky factorization on a local memory multiprocessor,"*SIAM J. Sci. Stat. Computing*,9,pp.327-340, 1988.

[10] J. R. Gilbert and E. Zmijewski, "A Parallel Graph Partitioning Algorithm for a Message Passing Multiprocessor," *Internat. J. Parallel Programming*, 16:427-449, 1987.

[11] M.T. Heath, E. Ng, and B.W. Peyton, "Parallel algorithms for sparse linear systems," *Parallel Algorithms for Matrix Computation*,SIAM Press, pp.83-124, 1990.

[12] J. Liu, "Reordering sparse matrices for parallel elimination ," *Parallel Computing*, 11:73-91, 1989.

[13] S. Venugopal and V.K. Naik, "Effects of Partitioning and Scheduling Sparse Matrix Factorization on Communication and Load Balance," *Proceedings of Supercomputing '91*, 1991.

[14] E. Zmijewski, "Sparse Cholesky Factorization on a Multiprocessor ," *Ph.D. Thesis*, Dept. of Computer Science, Cornell University, 1987.

[15] M. Zubair "Performance of multifrontal code on RS/6000," *unpublished manuscript*, 1991.

[16] M. Zubair and M. Ghose, "A performance study of sparse cholesky factorization on Intel ipsc/860", *Technical Report*, Dept. of Computer Science, Old Dominion Univ., 1992.

Table 1. List of structured matrices arising from nine-point finite difference operators on a square grid.

grid problem	order	nonzeros	operation count
50 × 50	2500	12202	2032374
75 × 75	5625	27677	7227520
100 × 100	10000	49402	17562662

Table 2. Performance of various partitioning schemes for structured matrices.

grid problem	np	tt (sec)				
		wr	cc1	cc2	ss	hr
50 × 50	2	1.28	0.95	0.92	0.92	0.94
	4	1.33	0.82	0.67	0.73	0.66
	8	1.65	0.82	0.44	0.70	0.44
	16	1.87	0.84	0.29	0.64	0.36
75 × 75	2	4.23	3.04	2.88	2.88	3.36
	4	4.81	2.51	2.13	1.92	2.04
	8	5.36	2.45	1.29	1.58	1.29
	16	7.31	2.52	0.82	1.52	0.96
100 × 100	2	7.96	7.31	6.89	6.87	8.22
	4	5.44	5.97	4.97	4.43	4.91
	8	4.21	5.72	2.97	3.45	2.83
	16	3.53	8.26	1.85	3.28	1.82

Table 3a. Distribution of operation counts for a 75 × 75 grid problem on a 4-processor machine.

part. scheme	p0	p1	p2	p3
wr	1804438	1804649	1813194	1805239
cc1	1266845	1856797	1266953	2836925
cc2	1777056	1803289	1784693	1862482
ss	1830051	1805154	1814518	1777797
hr	1820510	1818536	1807575	1780899

Table 3b. Distribution of computation time (in sec) for a 75 × 75 grid problem on a 4-processor machine.

part. scheme	p0	p1	p2	p3
wr	1.61	1.80	1.59	1.76
cc1	0.84	1.30	0.84	2.38
cc2	1.19	1.38	1.24	1.40
ss	1.66	1.65	1.67	1.66
hr	1.32	1.36	1.34	1.22

Table 3c. Distribution of total time (in sec) for a 75 × 75 grid problem on a 4-processor machine.

part. scheme	p0	p1	p2	p3
wr	4.81	4.81	4.81	4.81
cc1	0.91	1.40	0.91	2.51
cc2	1.33	1.61	1.38	2.13
ss	1.92	1.92	1.92	1.92
hr	1.99	2.03	2.04	1.51

Table 4. Performance in *mflops* of various partitioning schemes for a 16-processor machine

grid problem	mflops				
	wr	cc1	cc2	ss	hr
50 × 50	1.09	2.42	7.00	3.18	5.65
75 × 75	0.99	2.87	8.81	4.75	7.53
100 × 100	4.98	2.13	9.49	5.35	9.65

Table 5. List of Harwell Boeing test matrices.

matrix	order	nonzeros	operation count
bcsstk16	4884	147631	184196735
bcsstk17	10974	219812	214447177
bcsstk18	11948	80519	162705482
bcsstk28	4410	111717	40562546

Table 6. Performance of Harwell Boeing test matrices.

matrix	cc2		hr		mflops	
	ct	tt	ct	tt	cc2	hr
bcsstk16	8.77	41.15	7.88	32.07	4.48	5.74
bcsstk17	11.07	37.53	10.20	26.95	5.71	7.96
bcsstk18	9.30	26.47	7.18	21.97	6.15	7.41
bcsstk28	2.21	9.00	1.75	5.97	4.5	6.79

Table 7. Load distribution of Harwell Boeing test matrices.

matrix	sdopc		sdct		sdtt	
	cc2	hr	cc2	hr	cc2	hr
bcsstk16	0.01	0.01	0.07	0.05	0.37	0.42
bcsstk17	0.01	0.02	0.07	0.04	0.32	0.31
bcsstk18	0.01	0.03	0.09	0.03	0.22	0.10
bcsstk28	0.02	0.03	0.07	0.04	0.35	0.23

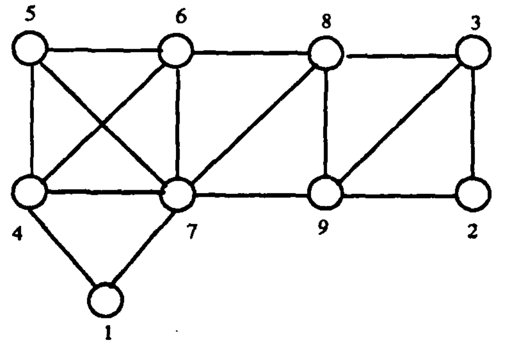
1	25	2	43	3	28	4
17	26	18	44	19	29	20
5	27	6	45	7	30	8
37	38	39	46	40	41	42
9	31	10	47	11	34	12
21	32	22	48	23	35	24
13	33	14	49	15	36	16

Ordering

0	0	0	2	1	1	1
0	0	0	3	1	1	1
0	0	0	0	1	1	1
0	1	2	1	3	0	1
2	2	2	2	3	3	3
2	2	2	3	3	3	3
2	2	2	0	3	3	3

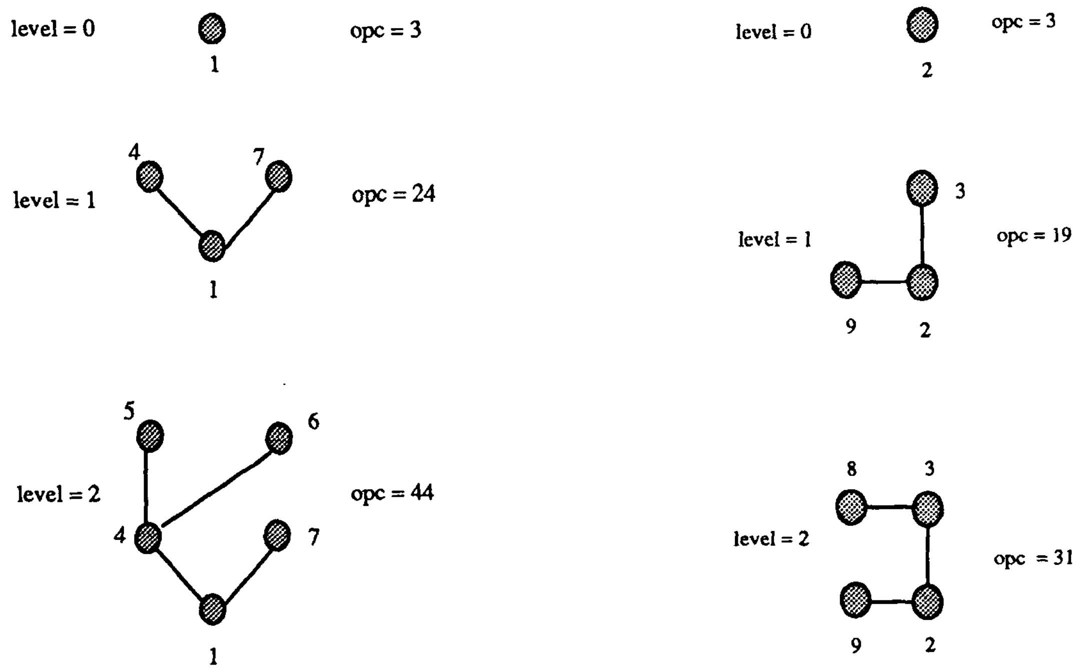
Partitioning

Figure 1. Nested dissection ordering and subtree-to-subcube partitioning on 7×7 grid.



c(1) = 3
c(2) = 3
c(3) = 7
c(4) = 8
c(5) = 9
c(6) = 11
c(7) = 13
c(8) = 12
c(9) = 9

Figure 2a. The graph G(F). (c(i) is the operation count associated with node i)



Formation of first partition

Formation of second partition

Figure 2b. Various stages of partition formation.
(opc gives the current operation count of a partition)

REPORT DOCUMENTATION PAGE	*Form Approved* *OMB No. 0704-0188*

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE March 1992	3. REPORT TYPE AND DATES COVERED Contractor Report

4. TITLE AND SUBTITLE A PERFORMANCE STUDY OF SPARSE CHOLESKY FACTORIZATION ON INTEL iPSC/860	5. FUNDING NUMBERS C NAS1-18605 WU 505-90-52-01
6. AUTHOR(S) M. Zubair M. Ghose	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Institute for Computer Applications in Science and Engineering Mail Stop 132C, NASA Langley Research Center Hampton, VA 23665-5225	8. PERFORMING ORGANIZATION REPORT NUMBER ICASE Report No. 92-13
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) National Aeronautics and Space Administration Langley Research Center Hampton, VA 23665-5225	10. SPONSORING/MONITORING AGENCY REPORT NUMBER NASA CR-189634 ICASE Report No. 92-13

11. SUPPLEMENTARY NOTES

Langley Technical Monitor: Michael F. Card
Final Report

Submitted to Frontiers '92: The 4th Symposium on the Mass. Parallel Computation

12a. DISTRIBUTION/AVAILABILITY STATEMENT Unclassified - Unlimited Subject Category 61	12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

The problem of Cholesky factorization of a sparse matrix has been very well investigated on sequential machines. A number of efficient codes exist for factorizing large unstructured sparse matrices, for example, codes from Harwell Subroutine Library [4] and Sparspak [7]. However, there is a lack of such efficient codes on parallel machines in general, and distributed memory machines in particular. Some of the issues which are critical to the implementation of sparse Cholesky factorization on a distributed memory parallel machine are: ordering, partitioning and mapping, load balancing, and ordering of various tasks within a processor. Addressing these issues optimally for unstructured sparse matrices is a challenging task. In this paper we focus on the effect of various partitioning schemes on the performance of sparse Cholesky factorization on the INTEL iPSC/860. We also propose a new partitioning heuristic for structured as well as unstructured sparse matrices, and compare its performance with the other schemes.

14. SUBJECT TERMS Sparse Cholesky factorization; distributed memory systems	15. NUMBER OF PAGES 17
	16. PRICE CODE A03

17. SECURITY CLASSIFICATION OF REPORT Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT

NSN 7540-01-280-5500

Standard Form 298 (Rev 2-89)
Prescribed by ANSI Std Z39-18
298-102