SPECTRAL NESTED DISSECTION*

ALEX POTHEN†, HORST D. SIMON‡ AND LIE WANG§

Abstract. We describe a spectral nested dissection algorithm for computing orderings appropriate for parallel factorization of sparse, symmetric matrices. The algorithm makes use of spectral properties of the Laplacian matrix associated with the given matrix to compute separators. We evaluate the quality of the spectral orderings with respect to several measures: fill, elimination tree height, height and weight balances of elimination trees, and clique tree heights. Spectral orderings compare quite favorably with commonly used orderings, outperforming them by a wide margin for some of these measures. These results are confirmed by computing a multifrontal numerical factorization with the different orderings on a Cray Y-MP with eight processors.

Keywords. graph partitioning, graph spectra, Laplacian matrix, ordering algorithms, parallel orderings, parallel sparse Cholesky factorization, sparse matrix, vertex separator

AMS(MOS) subject classifications. 65F50, 65F05, 65F15, 68R10

1. Introduction.

The parallel solution of large, sparse, symmetric positive definite systems requires the prior computation of a good parallel ordering of the matrix such that it can be factored efficiently. In this paper, we describe a spectral algorithm for computing good parallel orderings.

We employ a divide and conquer scheme, known as nested dissection, for computing the ordering. At a step of nested dissection, a vertex separator (a set of vertices whose removal disconnects the graph into two parts) in the adjacency graph of the matrix is computed, and the vertices in the separator are numbered last. Then the vertices in the two parts are numbered recursively by the same strategy. Nested dissection and generalizations have been described by George [9], Lipton and Tarjan [17], and Gilbert [12].

Thus the computation of a nested dissection ordering requires the identification of a good separator of the adjacency graph. A good separator in this context should satisfy two criteria: first, the size (number of vertices) of the separator should be small, and second, the two parts obtained by removing the separator should have roughly equal sizes. The first requirement controls the fill, and the second ensures load balance in the factorization among the processors, since the two parts are mapped to two subsets of processors of equal cardinality.

In earlier work [25], we had described a *spectral algorithm* for computing vertex separators of graphs. With the given matrix (and its adjacency graph) we associate a matrix called the Laplacian matrix, and compute initially an edge separator using the components of an

---

* Written Jan 1992.

† Computer Science Department, The Pennsylvania State University, Whitmore Lab, University Park, PA 16802 (pothen@cs.psu.edu, na.pothen@na-net.ornl.gov). The research of this author was supported by NSF grant CCR-9024954 and by U. S. Department of Energy grant DE-FG02-91ER25095.

‡ Numerical Aerodynamic Simulation (NAS) Systems Division, Mail Stop T045-1, NASA Ames Research Center, Moffett Field, CA 94035-1000 (simon@nas.nasa.gov). The author is an employee of Computer Sciences Corporation. This work was supported through NASA Contract NAS 2-12961.

§ Computer Science Department, The Pennsylvania State University, Whitmore Lab, University Park, PA 16802 (wangl@cs.psu.edu). The research of this author was supported by NSF grant CCR-9024954 and by U. S. Department of Energy grant DE-FG02-91ER25095.

eigenvector of the Laplacian matrix. A maximum matching in a bipartite subgraph induced by the edge separator can then be used to obtain a vertex separator. In this paper, we report a *spectral nested dissection algorithm* that recursively employs the spectral separator algorithm to obtain parallel orderings. We have tested the performance of the spectral nested dissection algorithm on a set of large, sparse matrices, and have compared spectral orderings with two commonly employed orderings. We also report the comparative performance of a sparse matrix factorization algorithm on a Cray Y-MP/8 when spectral nested dissection and other orderings are used to preorder the matrix.

This paper is organized as follows. Section 2 contains background information on a number of topics: the spectral approach to the graph partitioning problem, elimination trees, supernodes, and clique trees. The spectral separator and nested dissection algorithms are described in Section 3. Experimental results on the quality of spectral nested dissection orderings, comparisons with other ordering algorithms, and running times of a numerical factorization code with the different ordering algorithms are included in Section 4. The measures used to study the quality of the orderings include fill, elimination tree height, weight and height balance of the elimination tree, and clique tree height. Our conclusions and future research issues are described in Section 5.

2. Background. Given a symmetric matrix A, we associate a matrix Q called the Laplacian matrix with it. Because it is usual to associate the Laplacian matrix with a graph G, we can consider G to be the adjacency graph of A in what follows.

Let $G = (V, E)$ be an undirected graph on $|V| = n$ vertices. The $n \times n$ adjacency matrix $B = B(G)$ has element $b_{v,w}$ equal to one if $(v, w) \in E$, and zero otherwise. By convention $b_{v,v}$ is zero for all $v \in V$. The rows and columns of the matrices associated with a graph are indexed by the vertices of the graph, their order being arbitrary. Let $deg(v)$ denote the degree of vertex v, and define D to be the $n \times n$ diagonal matrix with $d_{v,v} = deg(v)$. The matrix $Q = Q(G) = D - B$ is the *Laplacian matrix* of G.

The spectral properties of Q have been studied by several authors; see the two recent surveys by Mohar [22, 23]. It is well-known that Q is a singular M-matrix. Let the eigenvalues of Q be ordered $\lambda_1 = 0 \leq \lambda_2 \leq \ldots \lambda_n$. An eigenvector corresponding to λ_1 is $\underline{e}$, the vector of all ones. The multiplicity of the zero eigenvalue is equal to the number of connected components in the graph. If G is connected, then the second smallest eigenvalue λ_2 is positive. We call an eigenvector $\underline{y}$ corresponding to λ_2 a *second eigenvector*.

Fiedler [7, 8] has studied the properties of the second eigenpair λ_2, $\underline{y}$. He has investigated the partitions of G generated by the components of the eigenvector $\underline{y}$. One of his results of interest in this paper is the following

THEOREM 2.1. *Let G be a connected graph, and let $\underline{y}$ be an eigenvector corresponding to λ_2. For a real number $r \geq 0$, define $V_1(r) = \{v \in V : y_v \geq -r\}$. Then the subgraph induced by $V_1(r)$ is connected. Similarly, for a real number $r \leq 0$, let the set $V_2(r) = \{v \in V : y_v \leq |r|\}$. The subgraph induced by $V_2(r)$ is also connected.*

In both sets V_1 and V_2, it is necessary to include all vertices with zero components for the theorem to hold. A corollary to this result is that if $y_v \neq 0$ for all $v \in V$, then the subgraphs induced by the sets $P = \{v \in V : y_v > 0\}$ and $N = \{v \in V : y_v < 0\}$ are both connected subgraphs of G.

In [25], two lower bounds on the sizes of the smallest vertex separators in a graph in terms of the eigenvalues of the Laplacian matrix $Q(G)$ were obtained. These results were obtained by applications of the Courant-Fischer-Poincarè minimax principle and the Hoffman-Wielandt theorem. The reader is referred to this paper for further details, an analysis of the model grid problem, and for a survey of spectral approaches to the partitioning problem.

In related work, spectral edge separator algorithms have been applied to the graph partitioning problem in the context of an MIMD implementation of an explicit Euler solver for CFD problems on unstructured grids [28, 29], and have been shown to have superior performance compared to other partitioning methods.

We now briefly mention a few recent developments concerning the partitioning and ordering problems. Liu [19] has described a hybrid approach that combines aspects of minimum-degree and nested dissection for computing orderings. Miller, Teng, and Vavasis [21] have introduced a geometric approach to computing vertex separators for a class of graphs embeddable in d dimensions. Agrawal, Gilbert, and Klein [1] have implemented a nested dissection approach which produces orderings that are optimal within logarithmic factors with respect to various measures. In this approach, separators are computed by solving a multicommodity flow problem as discussed by Leighton and Rao [15] and others. It remains to be seen if the latter two approaches could yield practical sparse matrix ordering algorithms.

Elimination trees, supernodes, and clique trees. In comparing the quality of different orderings in Section 4, we will refer to the elimination tree and a clique tree associated with a given ordering. Both of these are fundamental data structures in sparse matrix algorithms, and for completeness, we provide a brief discussion.

Let L denote the Cholesky factor of the given symmetric positive definite matrix A, and let $G(L)$ denote the (undirected) adjacency graph of L. The *elimination tree* of L (and of A under the given ordering) is a directed tree $T = (V, E_T)$, whose vertices are the columns of L, with a directed edge $(v, w) \in E_T$ if and only if the lowest-numbered row index of a subdiagonal nonzero in the v-th column of L is w. (The edge is directed from v to w.) The vertex w is the *parent* of v, and v is a *child* of w.

We define the *higher adjacency set* $hadj(v)$ to be the set of all vertices x adjacent to v in $G(L)$ such that x is numbered higher than v. A vertex w is the parent of vertex v in the elimination tree if and only if the lowest-numbered vertex in $hadj(v)$ is w. We refer the reader for additional details about elimination trees to a comprehensive survey provided by Liu [20].

The multifrontal algorithm for sparse Cholesky factorization makes use of groups of columns called supernodes to achieve vectorization and thereby high computational performance during the numerical factorization. Supernodes can be identified from the elimination tree by merging vertices according to their $hadj$ sets. Formally, a group of vertices $v_0, \ldots, v_k$ forms a *supernode* if
(i) the vertices form a path in the elimination tree,
(ii) all interior vertices of the path have only one child, and
(iii) $hadj(v_i) = \{v_{i+1}\} \cup hadj(v_{i+1})$, for $i = 0, \ldots, k-1$.
(This has been called a fundamental supernode by Ashcraft et al. [2]; slightly different defi-

nitions of supernodes have also been used.) A supernode corresponds to a dense submatrix with columns having a nested nonzero structure given by condition (iii) above.

It is well-known that $G(L)$ is a chordal graph, and thus has all of its maximal cliques of the form $v \cup hadj(v)$ for some vertex v. The *clique intersection graph* is an edge-weighted graph whose vertices are the maximal cliques of $G(L)$, with an edge joining two maximal cliques if the intersection of the two cliques is not empty. The weight of this edge is the number of vertices in the intersection of the two cliques. A *clique tree* is a maximum-weight spanning tree of the clique intersection graph (viewed as a rooted tree with an appropriate choice of a root clique). In [16], a more restrictive definition of a clique tree is given, which will not be repeated here. Clique trees have been applied in various contexts in sparse matrix algorithms: in addition to the above paper, see [4, 13, 26, 27].

3. The spectral nested dissection algorithm. In this section, we first describe a spectral algorithm to compute vertex separators that was designed in [25], and then discuss how the separator algorithm can be recursively used to compute elimination orderings.

The vertex separator algorithm is shown in Figure 1. It initially partitions the vertex set of the graph G into two subsets with l and $n - l$ vertices, for a prescribed value of l. This is accomplished by means of the vertex ordering given by the components of a second eigenvector of the Laplacian matrix of G. It then computes the edge separator between the two subsets, and finds a minimum vertex cover in the bipartite graph induced by the edge separator to obtain the desired vertex separator.

---

1. Compute an eigenvector $\underline{x}_2$ to a desired tolerance ϵ;
   find the l-th smallest value x_l of its components;
2. Partition the vertices of G into two sets:
   $A' = \{\text{vertices with } x_v \leq x_l\}$; $\quad B' = V \setminus A'$;
   If $|A'| > l$, move enough vertices with components equal to x_l
   from A' to B' to make $|A'| = l$;
3. Let A_1 be the set of vertices in A' adjacent to some vertex in B';
   Let B_1 be the set of vertices in B' adjacent to some vertex in A';
   Compute $H = (A_1, B_1, E_1)$, the bipartite subgraph induced by the vertex sets A_1, B_1;
4. Find a minimum vertex cover S of H by a maximum matching;
   Let $S = A_s \cup B_s$, where $A_s \subseteq A_1$, $B_s \subseteq B_1$;
   S is the desired vertex separator, and separates G into subgraphs with
   vertex sets $A = A' \setminus A_s$, $B = B' \setminus B_s$.

FIG. 1. *The spectral partitioning algorithm.*

The value of l can be chosen to make the initial part sizes as desired. We make two choices in the next Section: one is to make l the index of the median value of the eigenvector components. This leads to a partition with roughly equal sizes for the final parts A and B. Another choice is to make l equal to the number of vertices with components of one sign. If there are no vertices with zero components, then by Theorem 2.1 we know that the two subgraphs induced by the initial partition are connected.

$i := 0$; $G_0 := G$;
$num := 0$; make the vector $perm$ the empty vector;
While $num < |V|$ **do**
    Find the connected components $C_1, C_2, \ldots, C_k$ of G_i;
    for $j := 1, \ldots, k$ **do**
        if $|C_j| > size$ **then**
            Copy C_j to a local adjacency structure;
            Compute a vertex separator S_j of C_j by the spectral separator algorithm,
            using a second eigenvector computed to a tolerance ε_j;
        else
            $S_j := C_j$;
        end if
        $G' := G_i \setminus S_j$;
        $num := num + |S_j|$;
        Append S_j to $perm$;
    end for
    $G_{i+1} := G'$; $i := i + 1$;
end while
Reverse the ordering of vertices in $perm$.

FIG. 2. *The spectral nested dissection ordering algorithm.*

The spectral nested dissection algorithm is shown in Figure 2. It computes the vertex ordering stored in a vector *perm* by repeatedly calling the spectral separator algorithm to obtain vertex separators in subgraphs, until the subgraphs are small enough. The value of the parameter *size* determines when a subgraph is small enough to be ordered arbitrarily. It is necessary to choose the tolerance to which a second eigenvector of a subgraph is computed in accordance with the number of vertices in the subgraph to ensure the desired precision and speed in the eigenvector computations.

4. Experimental Results. In this section, we report experimental results obtained from the *Spectral Nested Dissection Ordering Algorithm*, and provide comparisons with the *Multiple Minimum Degree* (MMD), and the *Sparspak Nested Dissection* (AND) ordering algorithms.

We consider two variants of the spectral ordering algorithm, SND1 and SND2, that differ in the way in which the components of the second eigenvector are partitioned. SND1 partitions the components according to the median value, and SND2 partitions components according to their signs. The minimum degree ordering is currently the most popular ordering because of the low fill that it incurs. We report results using Liu's [18] *Multiple Minimum Degree Ordering Algorithm* (MMD), since it is an efficient implementation of this ordering. The *Sparspak Nested Dissection Algorithm* (AND) from Sparspak [10] uses breadth-first-search from a pseudo-peripheral vertex to obtain vertex separators from level structures.

The spectral algorithms make use of the Lanczos algorithm to compute a second eigen-

Problem	n	$\|A\|$	Description
GR 80×80.9	6,400	25,122	80×80 nine point grid
PWR8235	8,235	12, 674	Power problem from F. Alvarado
BARTH4	6,091	17,401	Connectivity matrix, four element air foil, see [3, 14, 29]
BARTH	6,691	29,748	Irregular mesh for one element airfoil
SKIRT.2	7,928	59,379	Shuttle aft skirt model, see [5] largest connected component
SRBEDDY	10,429	46,585	Shuttle rocket booster model, by E. Pramono
BARTH5	15,606	45,878	Connectivity matrix of four element air foil, refined grid see [3, 14, 29]
SPHERE6	16,386	49,152	Triangulation of the sphere based on a program by J. Richardson, TMC
PWT	36,519	144,794	Connectivity matrix, see [5] NASA Ames pressurized wind tunnel

TABLE 1
Test problems.

vector. The starting vector $\underline{w}$ was chosen with $w_i = i - (n+1)/2$, which makes it orthogonal to the first eigenvector of the Laplacian matrix, the vector of all ones. For the results of SND1 and SND2, the following termination criterion for the Lanczos iteration was used. Let

$$\varepsilon = \begin{cases} \frac{1}{2\|\underline{w}\|_2}, & \text{if } \frac{1}{\|\underline{w}\|_2} \leq 2^{-6} \\ 2^{-6}, & \text{otherwise} \end{cases} . \tag{1}$$

The iterations are terminated when the usual bound on the difference between the true and computed eigenvalues (equal to the product of the 2-norm of the residual vector and the last component of an appropriate eigenvector of the tridiagonal matrix) was no larger than ε, or 600 Lanczos steps had been performed. Explicit orthogonalization against the first eigenvector was employed in the Lanczos algorithm, as explained in [25].

Our programs were written in Fortran, and all experiments were run on a Cray Y-MP/8. We tested the algorithms on several problems from the Boeing-Harwell collection [6], and on finite-element problems obtained from NASA Ames Research Center. Here we report results for the largest problems from our test set, and these are shown in Table 1. Here n is the order of the matrix, and $|A|$ denotes the number of nonzeros in the strict lower triangle of the matrix A.

4.1. Quality of the orderings. We use several criteria to measure the quality of the parallel orderings produced by the four ordering algorithms. As in the sequential context, fill is an important measure of the storage and the arithmetic required for the factorization. A simple and effective criterion of the measure of concurrency in the factorization is the height of the elimination tree [20]. However, it is well-accepted by now that the 'shape' of the elimination tree is also an important criterion for the execution time of the parallel factorization. In general, a better 'balanced' elimination tree can be partitioned and mapped to a set of processors to obtain good efficiencies in the factorization. In the next paragraph, we describe one approach to computing measures of the balance of the elimination tree. Another measure useful especially on a massively parallel SIMD machine is the height of a clique tree. It should be noted that unlike the elimination tree, several clique trees may be obtained from a given ordering. However, experimentally the variation in the heights is small for the sizes of the problems that we have worked with, and hence we report results for clique trees produced by a clique tree algorithm described in [16].

Now we describe two measures of the balance of a rooted tree T on n vertices. It is more convenient to define the *imbalance* rather than the balance of T; the smaller the value of the imbalance, the better balanced the tree. Let $value(v)$ be a nonnegative value assigned to each vertex of T. We define $imbalance(v)$, the imbalance of a vertex v as

$$imbalance(v) = \begin{cases} 0, & \text{if } v \text{ is a leaf,} \\ \max\{value(w) | parent(w) = v\} - & \\ \qquad \min\{value(w) | parent(w) = v\}, & \text{otherwise.} \end{cases}$$

We obtain the *imbalance* of a tree T by summing $imbalance(v)$ over all the vertices of T.

We choose the variable $value(v)$ in two different ways. If $value(.) \equiv height(.)$, where $height(v)$ is the length of the longest path from a leaf of T to v, then we obtain the height imbalance of T. A second choice is $value(.) \equiv weight(.)$, where $weight(v)$ is the number of vertices in the subtree rooted at v (including v). Now we obtain the weight imbalance of a tree. Note that more sophisticated measures can be used for $value(v)$: for instance, it could be chosen as the number of arithmetic operations associated with v.

The definition of $imbalance(.)$ that we have used merits some comment. If a vertex v has only one child, then from the definition given above, $imbalance(v) = 0$. A moment's consideration of the elimination tree of the model grid problem ordered by optimal nested dissection should convince the reader that the definition is appropriate for such a vertex. This tree has each separator forming a long path, with the lowest numbered vertex in the separator having two children, the latter being the highest numbered vertices in the separators at the next level. Thus according to our definition, both a complete binary tree and a path have zero *imbalance*; the former is more appropriate than the latter as a task graph in parallel computation, and this can be inferred from their heights. However, both trees can be mapped in a parallel processor with little *imbalance* in computational work. This observation also points to the importance of using both height and balance properties of the elimination tree to evaluate the quality of a parallel ordering.

4.2. Comparison of the orderings. We now present our results in a series of Tables: Table 2 reports fill, Table 3 compares elimination tree heights, and Table 4 compares height and weight balances of elimination trees. Clique tree heights are reported in Table 5. All the problems in this collection except for the power network problem PWR8325 arise in the context of finite-element discretizations. The anomalous behavior of power network problems is well-known in the literature. The power problems are almost tree-like, with average vertex degrees bounded by about 1.5; as a consequence, they suffer little fill, and have a large number of supernodes with small average supernode sizes.

Problem	SND1	SND2	MMD	AND
GR 80×80.9	170,038	170,038	183,301	183,071
PWR8235	67,157	54,734	36,991	163,590
BARTH4	127,386	116,758	110,786	194,883
BARTH	139,581	130,349	103,795	214,235
SKIRT.2	394,286	325,799	341,656	443,185
SRBEDDY	326,708	315,436	317,171	379,620
BARTH5	391,398	363,361	356,722	618,873
SPHERE6	670,691	657,510	879,325	665,557
PWT	1,405,520	1,334,909	1,554,077	1,592,935

Table 2
Comparison of fill.

Problem	SND1	SND2	MMD	AND
GR 80×80.9	235	235	410	299
PWR8235	158	149	152	549
BARTH4	233	201	366	488
BARTH	237	236	303	401
SKIRT.2	393	349	903	559
SRBEDDY	296	291	1,302	472
BARTH5	349	336	424	810
SPHERE6	583	576	835	607
PWT	600	565	1,187	821

Table 3
Comparison of elimination tree height.

	Weight Imbalance				*Height Imbalance*			
Problem	SND1	SND2	MMD	AND	SND1	SND2	MMD	AND
GR 80×80.9	2.31	2.31	17.1	17.8	1.53	1.53	2.70	2.13
PWR8235	122	131	232	824	8.74	7.80	13.0	65.6
BARTH4	2.64	6.82	31.8	27.5	1.53	1.96	7.17	4.23
BARTH	3.03	8.63	26.9	37.8	1.81	2.41	4.61	5.36
SKIRT.2	19.8	12.8	81.4	48.5	2.84	3.08	16.5	8.54
SRBEDDY	5.44	7.08	175	28.7	1.33	1.73	24.9	5.46
BARTH5	6.99	20.6	61.2	118	3.87	4.68	11.5	11.8
SPHERE6	7.98	13.0	29.1	35.0	3.74	4.01	4.79	6.38
PWT	29.5	32.8	263	168	6.78	7.66	17.3	21.2

Table 4
Comparison of elimination tree balances. All numbers have been scaled by a thousand for convenience.

Problem	SND1	SND2	MMD	AND
GR 80×80.9	13	13	24	15
PWR8235	20	21	32	53
BARTH4	17	18	34	20
BARTH	17	18	23	24
SKIRT.2	18	17	34	22
SRBEDDY	15	14	48	19
BARTH5	19	19	22	25
SPHERE6	22	20	18	21
PWT	24	26	27	24

TABLE 5
Comparison of clique tree heights.

The spectral orderings, SND1 and SND2, incur fill that is quite comparable with the minimum-degree ordering MMD on this problem set. There are four problems where MMD has the least fill, three for which both spectral algorithms have lower fill than MMD, and two where SND2 has least fill. SND2 incurs less fill than SND1 on this set of problems. The Sparspak nested dissection ordering AND usually suffers the greatest fill, and it can be substantially larger than the three other orderings for some problems.

The elimination tree heights in Table 3 show that the spectral orderings have lower elimination tree heights than MMD and AND. Again, SND2 leads to lower heights than SND1, and on the average problem in this set, the MMD tree height is twice that of the SND2 height. The maximum value of this ratio is 4.5 for SRBEDDY. Notice that there are problems where AND leads to a smaller elimination tree height, and sometimes substantially so, than the MMD ordering.

The balance statistics reported in Table 4 show that the spectral orderings have the best balance properties. (The numbers in this Table have been scaled by a thousand for convenience.) Not surprisingly, SND1 has the edge over SND2 in terms of both height and weight balance, and the spectral orderings perform much better than MMD and AND. The average ratio of the weight balance of the MMD and SND2 orderings is seven, with the maximum value almost twenty five, again for the SRBEDDY problem.

Table 5 reports the clique tree heights obtained from the clique tree algorithm in [16] for the different orderings. Since the clique tree height behaves as $\Theta(\log n)$ for graphs with good separators, it is not surprising that the heights and the differences among them are much smaller than that for elimination tree heights. (Elimination tree height for graphs with good separators from two-dimensional problems behaves as $\Theta(\sqrt{n})$.) From the table, we can see that the spectral orderings give shorter clique trees than the other two orderings for most problems.

4.3. Numerical factorization results. The ultimate test of the quality of an ordering is the running time of a factorization algorithm which employs that ordering. Hence we report results of numerical factorization on a Cray Y-MP with eight processors for three of the test problems. The numerical factorization code is a multifrontal code developed at

Cray by Yang [30], who kindly ran the experiments for us with the spectral orderings that we supplied.

The multifrontal code makes use of supernodes to achieve vectorization and thereby high Megaflop rates during the numerical factorization. On a vector machine like the Cray, the larger the supernodes, the higher should be the expected degree of vectorization. Hence the average supernode size is often increased by coalescing two or more supernodes together at the expense of additional fill. A parameter *Relaxation* controls the additional fill permitted in each coalesced supernode. From now on, the word supernode will indicate a coalesced supernode.

There are two more parameters *Grouping*, and *Blocking* that control the concurrency in the multifrontal factorization. We explain these parameters in the next few paragraphs.

The set of supernodes is divided into two subsets: each supernode in the first subset is computed by a single processor, and all processors cooperate to compute each supernode in the second subset. The former is called the *outer-parallel subset*, and the latter, the *inter-parallel subset*. The value of *Grouping* determines the partition: If the update matrix of a supernode is no greater than *Grouping*, then the supernode is placed in the *outer-parallel subset*, otherwise, it is placed in *inter-parallel subset*. Thus the larger the value of *Grouping*, the larger the *outer-parallel subset*.

All supernodes in the *outer-parallel subset* are organized into groups by some grouping algorithm(see [30]). The update matrices of a group of supernodes in the *outer-parallel subset* are stored in the same block of the shared memory, and will be assigned to a single processor which is responsible for the computations associated with them. The parameter *Blocking* specifies the size of a block. It should be clear that the value of *Blocking* can not be smaller than the value of *Grouping*.

The parallelism in the factorization process is related to the number of CPU's and how large the parameters are. Larger values of *Grouping* will include more supernodes into the *outer-parallel subset*. Zero values for *Grouping and Blocking* ensure that all supernodes belong to the *inter-parallel subset*. In the initial part of the factorization, this leads to poor processor utilization since all processors are involved in computing each supernode, and these supernodes are small in size. Towards the end of the factorization, the number of supernodes are fewer, and these are relatively large in size, and hence processor utilization will be good. On the other hand, extremely large values of *Grouping* and *Blocking* put all supernodes into the *outer-parallel subset*. Initially this leads to good processor utilization, since a large number of small supernodes are assigned to each processor; but towards the end, the few large supernodes that remain to be factored will be computed by individual processors, instead of being computed by all processors concurrently.

Thus the concurrency in the factorization strongly depends on these parameters. Unfortunately, no recommendations have been made in [30] for optimal values of these parameters: in general, these parameters depend in a complex manner on the size of the supernodes and the structure of the supernodal elimination tree. Hence this is a good context for testing the influence of balance in the elimination trees on parallel factorization times.

Table 6 compares the numerical factorization times for three of the larger problems in our test set with the SND1 and MMD orderings on one CPU of a Cray Y-MP; Table 7

SND1					
Problem	0	64	128	256	512
SRBEDDY	0.4367	0.2806	0.2554	0.2467	0.2463
BARTH5	0.6236	0.3593	0.3299	0.3242	0.3261
PWT	1.8590	1.2310	1.1580	1.1340	1.1330
MMD					
SRBEDDY	0.4248	0.2816	0.2538	0.2457	0.2432
BARTH5	0.5803	0.3548	0.3272	0.3101	0.3112
PWT	1.9600	1.4420	1.4010	1.3540	1.3370

TABLE 6

Factorization times (in seconds) on one processor of a Cray Y-MP/8 under different values of relaxation.

compares parallel factorization times of the two orderings on all eight CPUs of the Y-MP.

The results for one processor are comparable for both SND1 and MMD orderings. On BARTH5, MMD incurs less fill than SND1, and these are reflected in the factorization times as well. On PWT, SND1 suffers less fill than MMD, and again, this is reflected in the times required for the factorization. Also, as the value of *Relaxation* is increased, the supernode sizes increase, and the factorization times decrease, indicating better vectorization.

When we consider the results for eight processors, it is immediately apparent that the height and balance of the elimination tree play a crucial role in the performance. Note that for SND1, as the parameters *Grouping and Blocking* increase, the execution times gracefully decrease, with at most a slight increase in times for the largest values of these parameters. The substantial decrease in running times as these parameters attain reasonable values should be noted. Also, the parallel factorization times are smaller with the SND1 ordering rather than with the MMD ordering.

The dependence of execution times on the parameters *Grouping and Blocking* is quite irregular when the MMD ordering is used. Notice that for the SRBEDDY and BARTH5 problems, these initially decrease and then substantially increase, as these parameters increase in value.

These results point to the fact that the elimination tree height and balance are will be even more important for good efficiencies for parallel factorization on massively parallel machines.

4.4. Execution time of spectral nested dissection. In order to give an estimate of the running time of the spectral nested dissection algorithm, we have implemented a version of SND1 for fast execution on a single processor of the Cray Y-MP/8. We emphasize that this version is only experimental at this time, and its main purpose is to demonstrate that spectral nested dissection can generate orderings in time roughly the same order of magnitude as MMD. This version running on the Cray, which we will call here SND-C, is derived from SND1, but uses a fast version of the Lanczos algorithm described in detail in [24] (also used in [28]). Another difference is that in SND-C there is no matching employed to compute the vertex separators from the edge separators; instead, the smaller endpoint set of the edge separator is taken to be the vertex separator.

SND1						
	0	512	0	512	0	512
Grouping/Blocking	SRBEDDY		BARTH5		PWT	
0/0	0.4109	0.2358	0.6345	0.3491	1.7130	0.8662
2000/3001	0.0853	0.0657	0.1113	0.0723	0.4383	0.2991
5000/5001	0.0691	0.0453	0.0943	0.0558	0.3189	0.2208
8000/8001	0.0721	0.0471	0.0897	0.0540	0.3559	0.1825
10000/10001	0.0787	0.0504	0.0949	0.0535	0.2739	0.1764
MMD						
Grouping/Blocking	SRBEDDY		BARTH5		PWT	
0/0	0.4091	0.2362	0.6181	0.3472	1.5320	0.9150
2000/3001	0.0891	0.0705	0.0976	0.0640	0.4043	0.3333
5000/5001	1.1670	2.0610	1.2430	0.0835	0.3476	0.2697
8000/8001	0.4042	2.5850	0.1824	0.1806	0.3299	0.2489
10000/10001	0.4115	0.9257	0.1160	0.2799	0.3224	0.2467

Table 7

Factorization times (in seconds) on 8 processors of a Cray Y-MP/8 under the relaxation 0 and 512.

Tolerance	Nonzeros in L	Time (sec.)
0.1	725,784	3.06
0.01	701,977	6.53
0.001	678,331	14.3
0.0001	652,730	21.8

Table 8

Execution times (on one processor) of a Cray Y-MP/8 and fill for the SND-C implementation when different tolerances are used for SPHERE6.

We first point out that spectral nested dissection permits a tradeoff between the running time of the algorithm, and the quality of the computed ordering, as measured by the amount of fill in the Cholesky factor L. This tradeoff is controlled by the tolerance ε, which has been set for SND1 and SND2 in (1). Generally, a smaller tolerance will result in a longer execution time, but in less fill, and vice versa. Table 8 demonstrates this behavior using the SPHERE6 matrix.

We see from Table 8 that the choice of ε in (1) is a compromise between a very short execution time and an improved ordering. When the tolerance in SND-C is set to 0.01, then on a subset of the sample problems the execution times in Table 9 are obtained. These are compared to the execution time for MMD on a single processor of the Cray Y-MP/8.

The fill-in for SND-C for the problems in Table 9 is similar to the values reported for SND1 in Table 2. The execution times for SND-C are a factor of five to twelve times slower than for MMD. However, these results are from an initial implementation of the SND algorithm, and there are several enhancements that could potentially improve the running

Matrix	SND-C	MMD
BARTH4	3.18	0.31
BARTH	4.04	0.38
SKIRT.2	8.31	0.99
SRBEDDY	6.63	0.49
BARTH5	9.44	0.83
SPHERE6	6.54	0.86

TABLE 9
Execution times (in secs) for SND-C and MMD algorithms on one processor of a Cray Y-MP/8.

time of the algorithm.

We list a few of these potential enhancements now. It is clear that the running time of the algorithm depends on the accuracy with which a second eigenvector is computed. This tolerance should be set in a dynamic manner, with high accuracy demanded of the large graphs to be partitioned at the beginning of the nested dissection, and low accuracies required of the smaller subgraphs towards the end. Indeed, subgraphs containing fewer vertices than an appropriately chosen threshold could be ordered using minimum degree or some other fast heuristic. Another enhancement is to compute the eigenvectors in parallel by means of a a parallel Lanzcos algorithm.

It should be borne in mind that the history of the development of the MMD algorithm shows [11] that a number of incremental improvements over a long period of time have given the MMD algorithm the speed it has today. We anticipate that the enhancements listed in the preceding paragraph and others will make spectral nested dissection faster. Furthermore, a user would be willing to employ spectral nested dissection even if its running time is greater than that of minimum-degree, say by a factor of five, if the quality of the resulting ordering leads to a faster net factorization time. There are many contexts such as numerical optimization where a matrix with the same structure but with different numerical values must be factored repeatedly. In such contexts, the costs of computing a good ordering can be amortized over the number of factorizations.

5. Conclusions. We have described a spectral nested dissection algorithm (SND) to compute orderings appropriate for parallel factorization of sparse, symmetric matrices. The algorithm makes use of spectral properties of the Laplacian matrix associated with the given matrix to compute separators. This algorithm has the potential of computing qualitatively different separators than previous combinatorial algorithms, since it makes use of global information about the adjacency graph, while the latter use local information about neighborhoods of vertices.

Our experimental results show that SND computes orderings that compare favorably with the Multiple-Minimum degree (MMD) and Sparspak nested dissection (AND) algorithms. The fill incurred by the spectral algorithm is quite close to that suffered by the MMD algorithm, and much smaller than the AND algorithm. We have also compared these algorithms with respect to four measures of good parallel orderings: elimination tree height, height and weight balances of the elimination trees, and clique tree height. The spectral algorithm clearly outperforms the MMD and AND algorithms with respect to these measures. For instance, for the large problems in our test set, elimination tree heights of the spectral orderings are almost half that obtained from the MMD orderings. For the measures of balance, the ratios are even more favorable towards the spectral algorithm.

We have also compared the performance of the orderings within the context of a supernodal multifrontal factorization code (developed at Cray) on a Cray Y-MP with eight processors. The spectral orderings lead to faster numerical factorization than the MMD orderings. Factorization times from spectral orderings also depend gracefully on the parameters that control the concurrency in the code. This behavior is important, since optimal *a priori* choices of these parameters are difficult to make. Factorization times from the MMD orderings show an irregular dependence on these parameters.

A preliminary implementation of SND requires about five to twelve times the running time of the MMD algorithm. In future work, we intend to make SND faster by incorporating several enhancements including the parallel computation of the eigenvectors. Spectral orderings will be compared with the other orderings for numerical factorization on distributed-memory multiprocessors. A better understanding of the theoretical underpinnings of the spectral separator algorithm is another priority.

Acknowledgements. We thank Dawson Deuermeyer (Cray Research Inc.), Edddy Pramono (NASA Ames/CSC), Tim Barth (NASA Ames), and Steve Hammond (NASA Ames/RIACS) for making available several of the large structural analysis and unstructured grid problems; and John Richardson (Thinking Machines Corporation) for the use of his program for the triangulation of the sphere. We also thank Chao W. Yang and Phuong Vu (Cray Research Inc.) for help with obtaining data on the performance of the multifrontal algorithm on the Cray Y-MP/8.

REFERENCES

[1] A. Agrawal, J. R. Gilbert, and P. N. Klein, *Provably good nested dissection orderings*. Talk presented at the IMA Workshop on Sparse Matrix Computations, Oct 1991.

[2] C. C. Ashcraft, R. G. Grimes, J. G. Lewis, B. W. Peyton, and H. D. Simon, *Progress in sparse matrix methods for large linear systems on vector supercomputers*, Int. J. Supercomputer Appl., 1 (1987), pp. 10–30.

[3] T. Barth and D. Jespersen, *The design and application of upwind schemes on unstructured meshes*, in Proceedings of the 27th Aerospace Sciences Meeting, 1989. Paper AIAA 89-0366.

[4] J. R. S. Blair and B. W. Peyton, *On finding minimum-diameter clique trees*, Tech. Rep. TM 11850, Mathematical Sciences Section, Oak Ridge National Lab, Oak Ridge, TN, 1991.

[5] D. Deuermeyer, *Large-scale solutions in structural analysis*, CRAY Channels, 12 (1990), pp. 15–17.

[6] I. S. Duff, R. G. Grimes, and J. G. Lewis, *Sparse matrix test problems*, ACM TOMS, 15 (1989), pp. 1 – 14.

[7] M. Fiedler, *Algebraic connectivity of graphs*, Czech. Math. J., 23 (1973), pp. 298–305.

[8] ——, *A property of eigenvectors of non-negative symmetric matrices and its application to graph theory*, Czech. Math. J., 25 (1975), pp. 619–633.

[9] J. A. George, *Nested dissection of a regular finite element mesh*, SIAM J. Numer. Anal., 10 (1973), pp. 345–363.

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

[11] ——, *The evolution of the minimum degree algorithm*, SIAM Rev., 31 (1989), pp. 1–19.

[12] J. R. Gilbert, *Graph Separator Theorems and Sparse Gaussian Elimination*, PhD thesis, Stanford University, 1980.

[13] J. R. Gilbert and R. Schreiber, *Highly parallel sparse Cholesky factorization*, Tech. Rep. CSL-90-7, Xerox Palo Alto Research Center, Palo Alto, CA, 1990.

[14] S. Hammond and T. Barth, *On a massively parallel Euler solver for unstructured grids*, in Research Directions in Parallel CFD, H. D. Simon, ed., MIT Press, Cambridge, 1992. To appear.

[15] F. T. Leighton and S. Rao, *An approximate max-flow min-cut theorem for uniform multicommodity flow problems with applications to approximation algorithms*, in Proc. of the 29th Annual Symposium on Foundations of Computer Science, IEEE, 1988, pp. 422–431.

[16] J. G. Lewis, B. W. Peyton, and A. Pothen, *A fast algorithm for reordering sparse matrices for parallel factorization*, SIAM J. Sci. Stat. Comput., 10 (1989), pp. 1146–1173. (Special section on Sparse Matrix Algorithms for Supercomputers).

[17] R. J. Lipton, D. J. Rose, and R. E. Tarjan, *Generalized nested dissection*, SIAM J. Numer. Anal., 16 (1979), pp. 346–358.

[18] J. W. H. Liu, *Modification of the minimum degree algorithm by multiple elimination*, ACM Trans. on Math. Software, 11 (1985), pp. 141–153.

[19] ———, *The minimum degree ordering with constraints*, SIAM J. Sci. Stat. Comput., 10 (1989), pp. 198–219.

[20] ———, *The role of elimination trees in sparse factorization*, SIAM J. Mat. Anal. Appl., 11 (1990), pp. 134–172.

[21] G. L. Miller, S. H. Teng, and S. A. Vavasis, *A unified geometric approach to graph separators.* preprint, 1991.

[22] B. Mohar, *The Laplacian spectrum of graphs*, in Sixth Intern. Conf. on Theory and Applic. of Graphs, Kalamazoo, Michigan, 1988.

[23] ———, *Laplace eigenvalues of graphs—a survey.* preprint, 1991.

[24] B. N. Parlett, H. D. Simon, and L. Stringer, *Estimating the largest eigenvalue with the Lanczos algorithm*, Math. Comp., 38 (1982), pp. 153–165.

[25] A. Pothen, H. D. Simon, and K.-P. Liou, *Partitioning sparse matrices with eigenvectors of graphs*, SIAM J. Mat. Anal. Appl., 11 (1990), pp. 430–452.

[26] A. Pothen and C. Sun, *Compact clique tree data structures in sparse matrix factorizations*, in Large-Scale Numerical Optimization, Proceedings in Applied Mathematics, Society for Industrial and Applied Mathematics, 1990, pp. 180–204.

[27] ———, *A distributed multifrontal algorithm using clique trees*, Tech. Rep. CS-91-24, Computer Science, Penn State, Aug 1991.

[28] H. D. Simon, *Partitioning of unstructured problems for parallel processsing*, Computing Systems in Engineering, 2 (1991), pp. 135–148.

[29] V. Venkatakrishnan, H. D. Simon, and T. Barth, *A MIMD implementation of a parallel Euler solver for unstructured grids*, Tech. Rep. RNR-91-024, NASA Ames Research Center, Moffett Field, CA 94035, 1991. Submitted to the The Journal of Supercomputing.

[30] C. W. Yang, *A parallel multifrontal method for sparse symmetric definite linear systems on the Cray Y-MP*, in Proc. of Fifth SIAM Conference on Parallel Processing for Scientific Computing, 1991.