LAPACK WORKING NOTE 64 (UT CS-93-203) DISTRIBUTED SPARSE GAUSSIAN ELIMINATION AND ORTHOGONAL FACTORIZATION

PADMA RAGHAVAN

Abstract. We consider the solution of a linear system $Ax = b$ on a distributed memory machine when the matrix A has full rank and is large, sparse and nonsymmetric. We use our Cartesian nested dissection algorithm to compute a fill-reducing column ordering of the matrix. We develop algorithms that use the associated separator tree to estimate the structure of the factor and to distribute and perform numeric computations. When the matrix is nonsymmetric but square, the numeric computations involve Gaussian elimination with row pivoting; when the matrix is overdetermined, row-oriented Householder transforms are applied to compute the triangular factor of an orthogonal factorization. We compare the fill incurred by our approach to that incurred by well known sequential methods and report on the performance of our implementation on the Intel iPSC/860.

Key words. parallel algorithms, sparse linear systems, sparse matrix factorization, Gaussian elimination, orthogonal factorization, nested dissection

AMS(MOS) subject classifications. 65F, 65W

1. Introduction.

Consider the solution of a system of linear equations $Ax = b$, where A is an $m \times n$ matrix. When $m = n$, the matrix can be converted to upper triangular form U by Gaussian elimination. When $m > n$, an orthogonal decomposition $A = QR$ can be computed to obtain the upper triangular factor R. In either case, the upper triangular matrix is used to obtain a solution to the linear system.

When the matrix A is sparse, the numeric computations are preceded by a symbolic step to order the columns so that the factor suffers low *fill*, i.e., few of the zeroes of A become nonzero. The column ordering can be induced by applying a fill-reducing heuristic such as Multiple Minimum Degree or Automatic Nested Dissection [8] to either the graph of A^TA or of $(A + A^T)$. Given that our main object is the parallel solution of the linear system, we compute a column order of A using Cartesian nested dissection of A^TA; this nested dissection can be applied in parallel [11, 17]. We formulate parallel algorithms to estimate the structure of the factor and to perform numeric computations using a separator tree that is available after nested dissection. Our current work thus completes a suite of algorithms for parallel solution of linear systems using direct methods [11, 12, 17].

Numeric computations for both Gaussian elimination and orthogonal factorization are similar when viewed as $A_i = M_iA_{i-1}$ for $i = 1, 2, \ldots, n$; $A_0 = A$ and the matrices M_i are suitably defined so that $A_n = M_nM_{n-1} \cdots A$ is upper triangular. A_n corresponds to U in the case of Gaussian elimination and R in the case of orthogonal transformations. At first glance, preservering sparsity during factorization and estimating the structure of the factor for the sparse solution of a square indefinite system with Gaussian elimination and partial pivoting may seem rather different from computing the factor R for an orthogonal factorization when the matrix is overdetermined. Fortunately, for a certain class of sparse matrices these issues can be resolved in much the same manner for both numeric schemes. A graph theoretic interpretation of the computations clearly reveals the similarities.

We present a graph-theoretic characterization in Section 2. In Section 3 we formulate algorithms for distributed memory machines and we use the graph model to establish correctness. In Section 4 we compare the fill incurred by our scheme to that incurred by a traditional sequential method using either Multiple Minimum Degree or Automatic Nested Dissection. We also report on execution times and rates observed for our implementation on the Intel iPSC/860. Section 5 contains concluding remarks.

The following assumptions are used. The matrix A has m rows, n columns and rank n. The matrix also has a *zero-free* diagonal; this assumption is justified since the rows of a sparse matrix with full column rank can be reordered to make the diagonal zero-free. It is also assumed that the matrix A is permuted according to a a column ordering selected to reduce fill; typically such a column ordering is obtained by an ordering for $A^T A$ [7, 9]. We use the letter F to denote the factor; F corresponds to U for Gaussian elimination and to R for orthogonal factorization.

2. Graph Model of Factorization. We assume the reader is familiar with basic graph theoretic concepts; a good reference is [15]. The graph theoretic model of nonsymmetric sparse matrix factorization evolved in the process of determining the exact nonzero structure of the factor. Although several variants are possible, our presentation is based on a characterization developed in [10] which is best suited for the development of our algorithms in Section 3.

The *bipartite graph of* A is denoted by $H(A)$ and has m row vertices and n column vertices. The row vertices are labeled $\hat{1}, \hat{2}, \ldots, \hat{m}$ and the column vertices are labeled $1, 2, \ldots, n$. The graph has an edge $(\hat{r}, c)$ from row vertex $\hat{r}$ to column vertex c for each $A_{r,c} \neq 0$.

We define a sequence of bipartite graphs H_i that will help us model the structure of the matrices A_i for $i = 0, 1, \ldots, n$. Recall that a matrix A_i results from a sequence of either elementary or orthogonal transformations of A. We use the term *bipartite row-merge graph* to denote any H_i. Define $H_0 = H(A)$; construct H_i from H_{i-1} by adding edges $(\hat{r}, c)$ for each $\hat{r} \in \mathcal{R}_{\mathcal{H}_\rangle}$ and each $c \in \mathcal{C}_{\mathcal{H}_\rangle}$ where $\mathcal{R}_{\mathcal{H}_\rangle} = \{\hat{\triangledown} : (\hat{\triangledown}, \rangle) \in \mathcal{H}_{\rangle-\infty}\}$ and $\mathcal{C}_{\mathcal{H}_\rangle} = \bigcup_{\hat{\triangledown} \in \mathcal{R}_{\mathcal{H}_\rangle}} \{\rfloor : \rfloor \geq \rangle \wedge (\hat{\triangledown}, \rfloor) \in \mathcal{H}_{\rangle-\infty}\}$. In other words, edges are added so that *all* row vertices adjacent to column vertex i are made adjacent to *all* columns that are adjacent to *any* such row. In the graph H_i, $\mathcal{C}_{\mathcal{H}_\rangle}$ is the set of column vertices adjacent to row vertex $\hat{\imath}$ for any row in $\mathcal{R}_{\mathcal{H}_\rangle}$. For example, $\mathcal{C}_{\mathcal{H}_\infty} = \{\infty, \in, \ni, \triangle\}$ and $\mathcal{R}_{\mathcal{H}_\infty} = \{\hat{\infty}, \hat{\ni}\}$ for the following matrix:

$$\begin{pmatrix} x & x & & & \\ & x & & & \\ x & & x & x & \\ & & x & x & \\ & x & & & x \\ & & & x & x \end{pmatrix}.$$

We now show that the graphs H_i do indeed model the structure of U for Gaussian elimination with row-pivoting and of R for orthogonal factorization. Consider Gaussian elimination with row pivoting; at the i-th step, the matrix M_i corresponds to the product of row permutation matrix (for pivoting) and a suitable elementary matrix. At this step, the rows that can be selected to become a pivot row, i.e., candidate pivot rows, are those with a nonzero in column i. Now fill due to applying a Gauss transform can occur only in these candidate pivot rows and only in columns that are nonzero in some pivot row. The process of constructing H_i from H_{i-1} correctly models this fact and from induction, it follows that the structure of U is contained in H_n. Consider now the row-oriented application of Householder transforms. At the i-th step, the matrix M_i corresponds to the Householder matrix that zeroes the subdiagonal elements of column i and replaces all rows containing a nonzero in column i with suitable linear combinations of their old values. With respect to nonzero structure, this amounts to replacing rows with a nonzero in column i with the union of structures of all such rows. This is exactly the manner in which graph H_i is constructed from H_{i-1} and so it follows that H_n corresponds to the structure of the factor R. Let f_{i*} denote the structure of the i-th row of either factor. Observe that $f_{i*} \subseteq \mathcal{C}_{\mathcal{H}_\rangle}$ where $\mathcal{C}_{\mathcal{H}_\rangle}$ is as defined earlier with respect to H_i. An interesting fact is that the structure estimated by this process is actually quite tight for matrices with the combinatorial *Hall property* [10].

3. Algorithms. We now present our distributed algorithms to compute the structure of the factor and to perform numeric computations for either scheme, i.e, Gaussian elimination or orthogonal factorization. A column order of A based on nested dissection of the graph of $A^T A$ is central to the formulation of our algorithms. The nested dissection ordering results in a *separator tree* whose nodes represent separators in the graph of $A^T A$; the separators in turn consist of columns of $A^T A$ and hence columns of A. We present a "generic" bottom-up algorithm for distributed computation on the separator tree. We then use the graph model of the previous section to establish correctness with respect computing the structure of the factor and the actual factor.

We begin with a description of the separator tree. The tree has as many nodes as the number of separators of $G(A^T A)$. A node in this tree is called a *representative vertex* since it denotes a set of columns of A that compose a separator in $G(A^T A)$. Let $\mathcal{S} = \{\rangle_\infty, \rangle_\in \cdots \rangle_\|\}$ be a separator; now $i_{j+1} = i_j + 1$ for $j = 2, 3, \ldots k$ and each i_j denotes the new number of some column of A. The set of columns in $\mathcal{S}$ form a chain; we call higher numbered columns ancestors of lower numbered ones, and we use $rep(\mathcal{S})$ to denote the smallest numbered vertex in $\mathcal{S}$. If $\mathcal{S}$ dissects some component G into components $G_1, G_2, \ldots G_l$, then the first separator of each of these components is made the child of $\mathcal{S}$. In this work we consider separator trees in which each representative vertex has at most two children, i.e., a separator creates at most two components.

Our distributed algorithms are based on the following assignment of disjoint processor subsets to disjoint separator subtrees. We assume that the total number of processors, denoted by P, is a power of two and we use the symbols $\pi_0, \ldots, \pi_{P-1}$ to denote processors. Let p processors be assigned to subtree rooted at $rep(\mathcal{S})$. Then if $p > 1$, each subtree rooted at a child of $rep(\mathcal{S})$ is assigned $p/2$ processors. This partition is applied recursively, starting at the root. At level $l = \log_2 P$ from the root, each subtree is assigned a single processor. We use $L(\pi_i)$ to denote the columns in this subtree which we call the local phase subtree for processor π_i.

Both symbolic and numeric algorithms consist of each processor's performing the following distributed algorithm; the actual computation depends on the context, i.e, it is the merging of column subscripts for structure estimation, and either Gaussian elimination or Householder transformation for numeric factorization. Each processor performs computations in its local phase subtree independently and in parallel. Let the subtree at $rep(\mathcal{S})$ be assigned the set of processors $P_{\mathcal{S}}$; then all processors in $P_{\mathcal{S}}$ cooperate to perform computations associated with $rep(\mathcal{S})$. In other words, at representative nodes at levels $(log_2 P - 1)$ to the root, processors in each processor subset cooperate to perform computations; the processor subsets are disjoint and their sizes double at each level, with $P/2$ subsets of size 2 at level $(log_2 P - 1)$ to a single subset of size P at the root.

A column to processor map can be naturally derived from the partitioning of the separator tree. Columns in a local phase subtree are assigned to the corresponding processor; columns in a separator $\mathcal{S}$ are wrap-mapped to processors in $P_{\mathcal{S}}$ when $rep(\mathcal{S})$ assigned to the set of processors $P_{\mathcal{S}}$. However, the problem of mapping rows to processors still remains. Rows must be mapped to processors so that distributed tree-structured computation as described in the preceding paragraph would indeed be correct. Our mapping scheme is derived from the column to processor map. Let $\pi(i)$ denote the processor to which column i is mapped. For any row r let $first(r)$ denote the lowest numbered column in which row r contains a nonzero; we map row r to processor $\pi(first(r))$. We also refer to this mapping by saying that row r is associated with the separator subtree rooted at $rep(\mathcal{S}_{\{)\nabla\int\sqcup(\nabla)})$.

We next show that the generic distributed tree-structured algorithm is indeed correct, i.e., computations in disjoint subtrees are data disjoint and hence can be computed in parallel. The result is established in Lemma 3.1 using the bipartite graphs defined earlier. The main source of parallelism is that due to sparsity by which the structure of H_i depends only on some, instead of all, of the previous bipartite graphs. This partial dependence is related to the separators of the graph of $A^T A$.

Lemma 3.1. *Consider a separator $\mathcal{S}$, with the subtree rooted at $rep(\mathcal{S})$ denoted by $\mathcal{T}_{\mathcal{S}}$. Let $\mathcal{R}(\mathcal{T}_{\mathcal{S}})$ denote the set of rows associated with columns in $T_{\mathcal{S}}$. For any $i \in \mathcal{S}$, let $\mathcal{R}_{\mathcal{H}_)}$ denote the set of rows in the bipartite graph H_{i-1} that contain a nonzero in column i. Then $\mathcal{R}_{\mathcal{H}_)} \subseteq \mathcal{R}(\mathcal{T}_{\mathcal{S}})$.*

Proof: Recall that the separator tree consists of separators of $G(A^T A)$. Observe that columns of a row of A form a clique in $G(A^T A)$. Assume that there exists a row r such that $r \in \mathcal{R}_{\mathcal{H}_)}$ but $r \notin \mathcal{R}(\mathcal{T}_{\mathcal{S}})$. Then, there must exist some row q of A such that $A_{q,k} \neq 0$ and $A_{q,i} \neq 0$ where $k < i$ and $first(q) = k$. Let $\underline{\mathcal{S}}$ be the separator containing k; now by the row assignment scheme, q is contained in $\mathcal{R}(\mathcal{T}_{\underline{\mathcal{S}}})$. Since $k < i$, the trees $T_{\mathcal{S}}$ and $T_{\underline{\mathcal{S}}}$ must be disjoint. Let $\overline{\mathcal{S}}$ be the smallest common ancestor of these two trees. Now $\overline{\mathcal{S}}$ cannot be a separator in $G(A^T A)$ since i and k belong to the same clique in $G(A^T A)$. The proof follows from this contradiction. ■

Computing the row structure of the factor. A sequential algorithm to compute f_{i*}, the structure of the i-th row of the factor F, follows from the construction of the bipartite graphs H_i in Section 2. In a distributed algorithm, sets that contain f_{i*} can be computed by a simple merging scheme on the separator tree. Let $struc(k)$ denote the set of columns having nonzeroes in row k; likewise, if X is a set, let $struc(X)$ denote the union of column sets containing nonzeroes over all rows in X. For a separator $\mathcal{S}$, denote by $newrows(\mathcal{S})$, the set of rows whose first nonzero is in any of the columns in $\mathcal{S}$; these are a set of rows that have not been processed earlier. Also, let $children(\mathcal{S})$ denote the separators that are immediate descendants of $\mathcal{S}$ in the separator tree.

Lemma 3.2. *Define $struc(\mathcal{S}) = \int\sqcup\nabla\sqcap\rfloor(\rfloor\langle\rangle\updownarrow\lceil\nabla\rceil\backslash(\mathcal{S})) \cup \int\sqcup\nabla\sqcap\rfloor(\backslash\rceil\sqsupseteq\nabla\wr\sqsupseteq\int(\mathcal{S})) \setminus \{| : | < \nabla\rceil\surd(\mathcal{S})\}$. For any $i \in \mathcal{S}$, define $struc(i)$ as $struc(\mathcal{S}) \setminus \{| : | < \rangle\}$. Then, $f_{i*} \subseteq struc(i)$.*

Proof: Follows from Lemma 3.1 and induction. The result obviously holds when $\mathcal{S}$ is a leaf separator, i.e., when $children(\mathcal{S}) = \phi$. ¿From this basis and Lemma 3.1, the result follows for any column. ■

The *struc* sets corresponding to each separator are computed in parallel using the generic distributed tree-structured algorithm; correctness follows from the lemmas above. Each processor initially computes the structure of the separators in its local tree independently and in parallel, without any communication. Based on its data, each processor π_i also computes the partial structure of separators on the path from its local phase subtree to the root. The complete structure of thse separators is obtained by applying the distributed tree structured algorithm with the computation being that of merging sets. Consider the total size of the *struc* sets over all separators in a local phase subtree; let $\mathcal{C}_\mathcal{L}$ denote the largest over all processors. Let $\mathcal{C}_\mathcal{D}$ denote the maximum size of a *struc* set in the distributed phase. Observe that the merge at a separator involving some p processors would require $\log_2 p$ communication steps. This results in a computational complexity at a processor of $O(\mathcal{C}_\mathcal{L} + \mathcal{C}_\mathcal{D}(\log_\in \mathcal{P})^\in)$, with a communication overhead of $O((\log_2 P)^2)$ messages for a volume of $O(\mathcal{C}_\mathcal{D}(\log_\in \mathcal{P})^\in)$.

Computing the factor. The rows of the factor are computed by processing each separator; processing a separator $\mathcal{S} = \{\rangle, \rangle + \infty, \ldots, \rangle + \text{II}\}$ completes the formation of rows $i, i+1, \ldots i+q$ of the factor.

When $\mathcal{S}$ is a leaf separator define $\mathcal{R}_\mathcal{S}$ as the set of rows whose first nonzero is in any of the columns in $\mathcal{S}$, i.e. the set $newrows(\mathcal{S})$. We treat the set of rows in $\mathcal{R}_\mathcal{S}$ as forming a dense submatrix in columns given by $struc(\mathcal{S})$. The numeric computation associated with $\mathcal{S}$ corresponds to Gaussian elimination (with row pivoting) or row-oriented Householder transforms [18, 5] to form rows of the factor F for values in $\mathcal{S}$. At the end of this process, the first $|\mathcal{S}|$ rows are completely processed. Let $\tilde{\mathcal{R}}_\mathcal{S}$ denote the remaining rows.

When $\mathcal{S}$ is not a leaf separator, we define $\mathcal{R}_\mathcal{S}$ as the union of the sets of rows that remain at children separators along with the set of rows that are new for this separator. Let the children separators be $\mathcal{S}_|$ and $\mathcal{S}_{||}$, then $\mathcal{R}_\mathcal{S} = \tilde{\mathcal{R}}_{\mathcal{S}_|} \cup \tilde{\mathcal{R}}_{\mathcal{S}_{||}} \cup \backslash\rceil\sqsupseteq\nabla\wr\sqsupseteq\int(\mathcal{S})$. Dense matrix operations are once again applied as described earlier.

In the case when a separator is assigned to a set of processors $\mathcal{P}$, the dense matrix operations are applied in a distributed manner. For a treatment of distributed dense matrix computations, the reader is referred to [3, 16]. We use schemes that are row-wrap mapped. For Gaussian elimination, a pivot row is selected as the one with the largest diagonal element. In the case of computing R for overdetermined systems, Joseph Liu [14] observed that intermediate arithmetic work can be reduced by attempting to reduce the number of rows that are propagated up the tree. In other words, an attempt is made to reduce the size of the set $\tilde{\mathcal{R}}_\mathcal{S}$. Whenever $u = |\mathcal{R}_\mathcal{S}|$ is larger than $v = |struc(\mathcal{S})|$, the last $u - v$ rows can be completely zeroed out at separator $\mathcal{S}$. This can be done by applying $|struc(\mathcal{S})|$ Householder transforms instead of $|\mathcal{S}|$ transforms. We adopt this *general row merge* approach when computing R.

4. Empirical Results. Our experiments were designed to ascertain the feasibility of our approach with respect to two key issues. The first is the efficacy of our method in preserving sparsity in the factor; low fill results in low arithmetic costs. A second issue is that of performance on a parallel machine. As a consequence, we present two sets of experimental results. In the first set, we compare the fill incurred by our scheme with that incurred by using well established serial algorithms. In the second set, we present times and execution rates achieved on the Intel iPSC/860 for symbolic factorization, Gaussian elimination and orthogonal factorization. Our results indicate that our method performs well with respect to both issues.

Sparsity Of the Factor. The serial methods we use for comparison are those in Sparspak [1, 6]. In both Sparspak and our approach, a fill-reducing column ordering of a nonsymmetric A is induced by a heuristic applied to the graph of $A^T A$. In our approach, the fill reducing ordering is that obtained by implicit Cartesian nested dissection [17] while for Sparspak we use Multiple Minimum Degree or Automatic Nested Dissection [1, 6]. Sparspak structures computations based on the elimination

Label	n	$\|A\|$	Comments
airfoil 2	4,720	32,164	airfoil by Barth and Jesperson
airfoil 3	15,606	107,362	4-element airfoil mesh by Barth
barth	6,691	46,187	1-element airfoil
graded box	7,861	53,725	small elements at bottom right corner
graded L	6,142	42,448	small elements at a middle corner
graded +	6,043	39,775	small elements at a middle corner
hollow square	5,512	37,960	small elements around hollow
parc	1,240	7,950	PARC cut from a rectangle, small elements around letters
pinched hole	8,848	61,264	pentagon with hole, small elements around hole
regular grid	10,000	49,600	
six hole	9,971	68,451	small elements around each hole
venkat 1	10,089	69,529	concentric layers with elements of increasing size
venkat 2	460	3,066	same as above
ac	2,851	33,035	mostly tetrahedral, some beam and plate elments
hscts	2,028	42,710	same as above
kall0	3,000	34,900	same as above
kall1	4,363	57,503	same as above
shuttle	10,429	103,599	mostly 2-D elements, some 3-D elements
sphere 3	258	1,794	surface triangulation of a sphere
sphere 4	1,026	7,170	same as above
sphere 5	4,098	28,674	same as above
sphere 6	16,386	114,690	same as above

Table 1
Description of Test Problems, Square A.

tree of A^TA; the structure of rows of the factor are determined using symbolic factorization on A^TA, and rows of A are are transformed by *diagonal pivoting* against rows of the factor; the reader is referred to [7] for more details. In our approach, the structure is estimated using the separator tree; unless the separators are minimal this could lead to an overestimate of the nonzero structure and arithmetic work. Furthermore, for overdetermined systems, the processing of rows greatly affects the amount of arithmetic work performed. Our approach uses the *general row merge* method [14] whereas the Sparspak implementation uses *diagonal row pivoting*. To allow for objective comparison, we use the number of nonzeroes in the upper triangular factor (as opposed to the actual number of floating point operations) as a measure of efficacy of each approach. This measure applies to both Gaussian elimination and orthogonal factorization since it does not depend on the actual numeric transforms. It is also independent of row-ordering and reflects the effect of the fill-reducing and symbolic factorization algorithms. We report on this measure for our approach and

Label	n	m	$\lvert A \rvert$
geo10	5540	41,640	98,940
geo11	7425	55,704	131,736
geo12	9696	72,624	171,744

Table 2
Description of Geodetic Network Problems, Overdetermined A.

Label	Multiple MD	Automatic ND	Cartesian ND
airfoil 2	252	**394**	341
airfoil 3	1,075	**2,057**	1,405
barth	392	**701**	544
graded box	296	499	**511**
graded L	412	541	**702**
graded +	184	269	**442**
hollow square	396	460	**569**
parc	22	43	**46**
pinched hole	673	**870**	846
regular grid	596	615	**824**
six hole	765	**844**	811
venkat 1	755	953	**1007**
venkat 2	15	**18**	17
ac	185	256	**292**
hscts	220	**376**	366
kall0	765	**901**	870
kall1	1,506	**1,765**	1,721
shuttle	1,061	**1,271**	1,209
sphere 3	10	**12**	**12**
sphere 4	70	76	**77**
sphere 5	**421**	420	416
sphere 6	**2,523**	2,288	2,246
geo10	27	**50**	45
geo11	36	**64**	63
geo12	47	**90**	82

Table 3
Number of Off-diagonal Nonzeroes (in thousands) in the Factor.

for Sparspak (with two different ordering schemes) for a set of twenty five problems.

Most of the problems in our test suite arise from highly graded finite-element discretizations. We generated some of the problems associated with 2 dimensional meshes using Patran, a commercially available finite-element package. Several others in both two and three dimensions were obtained from practical applications such as structural analysis of airfoils. The problems range in size from 460 to $16,286$ equations; these sizes are not suitable for observing speed-ups on current parallel architectures. However, they do quite well for the task at hand, namely, comparing the performance of our approach with that of well known serial algorithms and implementations. These problems were used to to construct square systems with symmetric nonzero structure that are numerically nonsymmetric. For orthogonal factorization, we needed overdetermined systems whose unknowns are associated with coordinates in space. We generated problems associated with geodetic networks as described in [4]; these are labeled geo10, geo11 and geo12 in Table 2. A problem labeled "geox" corresponds to the geodetic problem with "x" junction boxes and chain-lengths. There are two variables per vertex and two observations each for any set of four variables joined by an edge and any set of six vertices joined in a triangle. A description of the test suite is given in Tables 1 and 2.

In Table 3, we report on the number of off-diagonal nonzeroes in the factor. The worst (largest) value of the measure over all methods is typeset in boldface for each problem in the test suite. The column labeled "Cartesian ND" contains the best result (smallest number of off-diagonal nonzeroes in the factor) for our approach when our Cartesian nested dissection ordering is applied to an implicit representation of the graph of A^TA with a balance factor α in the range .3 to .4. The balance factor $\alpha < (1/2)$ is a value used in Cartesian dissection to compute small separators subject to the constraint that a subgraph be split into pieces that contain at least α times the number of vertices. The ordering is followed by symbolic factorization on the separator tree. The column labeled "Multiple MD" ("Automatic ND") contains the best result when the Multiple Minimum Degree (Automatic Nested Dissection) heuristic of Sparspak is applied to the graph of A^TA over six different initial numberings of the columns of A; the six initial numberings correspond to five random numberings and a natural numbering, i.e., the numbering in which the problem was given to us. The symbolic factorization is *exact*, i.e., is based on the structure and elimination tree of A^TA.

The numbers in Table 3 demonstrate that our parallel method compares well with the best sequential methods. Not surprisingly, the Multiple Minimum Degree ordering leads to the lowest fill in all cases except two. The Automatic Nested Dissection ordering results in the largest fill for fourteen of the problems while our method leads to largest fill for ten of the problems. Interestingly enough, four of these ten problems are those we constructed with Patran as difficult test cases for our Cartesian nested dissection algorithm. These problems have very highly graded elements and dense subregions located such that our heuristic would find it hard to construct small separators for a balance constraint in the range .3 to .4. A fifth problem for that our method has the largest fill is the regular grid; however, our method computes wide separators which are known to be asymptotically optimal.

Performance of an iPSC/860 Implementation. Our algorithms were implemented on the Intel iPSC/860 in C with message passing extensions. The rows in the dense matrix in question were wrap mapped onto the relevant set of processors before invoking a distributed dense kernel. The computation was done in single precision with assembler coded "saxpy" routines. We report on symbolic and numeric factor

Label	n	Serial Cost (in millions)	Load Ratio
g200	40,000	888	1.2, 1.4
g300	90,000	3,179	1.2, 1.4
g400	160,000	7,767	1.3, 1.3
g500	250,000	15,486	1.3, 1.3
g600	360,000	27,164	1.3, 1,3
g700	490,000	43,624	1.3, 1.3
gph0	15,680	277	1.5, 2.7
gph1	35,112	978	1.4, 3.5
gph2	62,272	2,466	2.0, 3.9
g6h0	15,475	152	1.3, 1.6
g6h1	30,095	440	1.3, 1.6
g6h2	61,055	1,376	1.3, 1.6
ghs0	14,976	287	1.2, 1.6
ghs1	39,312	1,055	1.2, 1.4
ghs2	74,464	2,319	1.1, 1.6
geo20 ($m = 342,480$)	46,080	26	1.7, 2.0
geo25 ($m = 672,600$)	90,725	62	1.6, 2.0

TABLE 4
Description of Large Test Problems.

Label	$P = 8$		$P = 16$		$P = 32$		$P = 64$		$P = 128$	
	Time	Cost	Time	Cost	Time	Cost	Time	Cost	Time	Cost
g200	0.65	133	0.30	74	0.15	38	0.12	19	0.10	10
g300	0.90	474	0.60	262	0.26	133	0.16	67	0.15	34
g400					0.56	321	0.24	161	0.20	81
g500					0.59	639	0.34	321	0.28	162
g600							0.44	558	0.34	282
g700							0.58	900	0.44	452
gph0	0.35	55	0.19	32	0.12	19	0.08	11	0.07	6
gph1	1.07	179	0.53	118	0.27	76	0.16	45	0.11	27
gph2			0.40	322	0.53	213	0.29	132	0.16	77
g6h0	0.25	25	0.13	13	0.09	7	0.06	3.8	0.06	1.9
g6h1	0.55	73	0.28	38	0.16	21	0.10	11	0.08	5.7
g6h2			0.59	111	0.29	61	0.15	33.7	0.11	17.3
ghs0	0.35	44	0.20	23	0.14	13	0.10	7.1	0.08	3.7
ghs1	1.06	158	0.54	83	0.30	45	0.17	23	0.11	11.6
ghs2			1.20	166	0.60	91	0.31	53	0.18	29
geo20			3.32	2.8	2.13	1.6	0.92	.82	0.48	.42
geo25					2.90	3.1	1.48	1.8	1.10	1.0

TABLE 5
Time (in seconds) for Symbolic Factorization and Critical Cost (in millions).

times for a set of large problems. We do not report on triangular solution times since they are similar to those in [12] for a symmetric, positive definite A.

Label	$P = 8$		$P = 16$		$P = 32$		$P = 64$		$P = 128$	
	Time	Rate	Time	Rate	Time	Rate	Time	Rate	Time	Rate
g200	25.8	34	16.3	54	**12.5**	**71**	13.1	68	15.2	58
g300	78.6	40	44.8	71	29.8	106	**28.1**	**113**	31.4	102
g400					57.6	135	**50.3**	**154**	54.4	143
g500					98.5	157	**81.0**	**191**	83.6	184
g600							121.1	223	**120.0**	**226**
g700							169.2	257	**163.7**	**264**
gph0	12.7	22	8.1	35	6.5	43	**6.1**	**45**	6.7	41
gph1	39.1	25	26.7	37	18.9	52	14.5	67	**12.6**	**77**
gph2			64.6	38	45.1	55	31.9	77	**25.1**	**99**
g6h0	6.8	22	4.2	37	**3.4**	**45**	3.6	42	4.3	35
g6h1	17.2	26	9.6	46	7.1	62	**6.7**	**65**	7.4	59
g6h2			23.1	60	15.1	92)	12.6	100	**12.3**	**112**
ghs0	10.3	28	6.6	44	**5.9**	**48**	6.4	45	7.2	38
ghs1	31.5	34	17.8	59	13.5	78	**12.5**	**84**	12.6	74
ghs2			30.2	77	24.5	95	22.8	102	**22.1**	**106**

TABLE 6
Execution Time (in seconds) and Rate (in Mflops) for Distributed Gaussian Elimination.

Our test suite was limited by the fact that we required large problems associated with an embedding (for Cartesian Nesed dissection). The set of square problems correspond to the regular grid in six sizes (gxxx) and the graded pinched hole (gphx), the graded six-hole (g6hx) and the graded hollow square (ghsx), in three sizes each. The graded problems were generated using Patran and are highly irregular; they are larger versions of those used in the test suite for comparison with serial codes. We also generated overdetermined systems associated with geodetic networks; again these are bigger versions of those used in the earlier test set. Finally we have a set of simulated least squares problems on $k \times l$ grids, typical of those arising in the natural factor formulation of finite element methods [4]. Each grid consists of $(k-1) \times (l-1)$ small squares and associated with each small square are four observations in the four variables at the corners. Cartesian nested dissection produces ideal separators for problems associated with the regular grid; for all other problems we used a balance constraint of .4.

The problems are described in Table 4 where the column labeled "Serial Cost" contains the total number of floating point operations required for Gaussian elimination; this value is half the number required for computing R using Householder transforms. Table 5 contains the time in seconds for symbolic factorization using the separator tree. It also contains the largest number of floating point operations performed at any one processor (once again, twice this number is required for Householder transforms) for 8 through 128 processors. The symbolic factor times are very small owing to our formulation in terms of the separator tree and even these small time requirements are decreased with a larger number of processors. For P processors, we compute the ratio of the actual load to the "ideal" load where we define the

Label	$P = 8$		$P = 16$		$P = 32$		$P = 64$		$P = 128$	
	Time	Rate	Time	Rate	Time	Rate	Time	Rate	Time	Rate
g200	39.8	45	23.3	78	16.1	114	**16.0**	**117**	18.4	106
g300	126.4	51	67.3	96	39.2	165	**33.8**	**193**	36.8	183
g400					77.3	204	**60.8**	**261**	62.7	258
g500					136.4	230	98.8	320	**97.3**	**329**
g600							148.2	372	**138.8**	**400**
g700							192	400	**165.7**	**464**
gph0	19.2	30	11.9	48	9.4	61	**8.7**	**67**	9.3	65
gph1	61.2	33	41.0	49	29.0	69	21.4	94	**18.2**	**114**
gph2			102.2	50	70.6	71	48.5	104	**36.1**	**143**
g6h0	10.2	31	6.1	51	**4.8**	**66**	5.3	61	6.1	55
g6h1	26.4	34	14.3	63	10.0	90	**9.4**	**98**	10.0	94
g6h2			35.2	80	21.9	128	17.2	163	**16.3**	**175**
ghs0	15.3	38	9.2	64	**7.8**	**76**	8.3	73	9.6	67
ghs1	48.5	44	25.8	83	18.1	119	**16.1**	**136**	17.6	128
ghs2			44.2	106	33.6	140	29.4	162	**28.4**	**172**
geo20					2.9	17	2.3	23	**1.6**	**32**
geo25					4.8	24	4.1	28	**2.8**	**44**

TABLE 7
Execution Time (in seconds) and Rate (in Mflops) for Distributed Orthogonal Factorization.

Label	P	Time (Predicted)	Time (Observed)	Rate	Speedup
64×32	1		1.96	1.1	
64×64	2	2.9	2.20	1.2	2.64
128×64	4	3.9	2.54	5.9	6.14
128×128	8	5.8	3.41	19.5	13.6
256×128	16	7.8	5.23	46.3	23.8
256×256	32	11.6	9.07	108.2	40.8
512×256	64	15.6	17.11	179.6	58.3
512×512	128	23.1	34.73	308.8	85.1

TABLE 8
Distributed Orthogonal Factorization: Model Problem

ideal load as the serial cost divided by the value of P; we present the range of this ratio over different values of P in the column labeled "Load Ratio" in Table 4. This range of ratios indicates that our ordering with a balance constraint of 0.4, results in at worst a load that is 3.9 times the ideal; in most cases it is within a factor of two of the ideal load. Furthermore, from Table 5, the largest load at any processor is approximately halved upon doubling the number of processors. These results suggest that for the test problems our approach succeeds in balancing loads within a small constant of the ideal value for varying numbers of processors.

Our algorithms exploit task parallelism by allowing computations on disjoint subtrees to occur on disjoint sets of processors. However, given the tree structure, such functional or task parallelism decreases towards the root while the tasks become of larger size making data-parallelism within each task more viable. This data-parallelism is to be utilized by means of dense distributed kernels. For the architecture at hand, the communication to computation ratio is very high and this makes the distributed dense factorization kernels achieve rather low execution rates for dense matrices of the size that occur during sparse computations. Over all our test problems, the largest dense matrices vary in size from 100 to about 1400. Furthermore, we use full row-pivoting for Gaussian-elimination with the matrix wrapped by rows and of the possible dense distributed Gaussian elimination kernels, this tends to perform worse than those with full row-pivoting and column wrap-mapping [3]. A similar statement is true of distributed Householder transforms [13]. However, row-oriented schemes fit naturally with sparse factorization and are also suitable for exploiting parallelism in the triangular solution to follow. In spite of these limitations, we do observe speedups on increasing the problem size while increasing the number of processors. We also see higher execution rates on moving to larger problems for the same number of processors.

Tables 6 and 7 contain times in seconds for the two numeric schemes. The times include all overheads such as allocating new dense matrices, freeing up old matrices and copying rows. The numbers in each column correspond to execution times for a fixed number of processors over all problems; the numbers in parentheses in each column represent the corresponding execution rate in millions of operations per second. This execution rate is arrived at by allowing a count of one per 'useful' floating point multiply-add pair; symbolic operations are not counted and neither are overhead operations such as those used to initialize a vector to contain zeroes. The problems are listed in increasing order of size and are grouped by problem type. Numbers in boldface represent the best execution time (rate) for a given problem. Comparing the two tables, it can be seen that higher execution rates are achieved for orthogonal factorization than for Gaussian elimination for the same problem. This occurs because although both schemes have the same communication costs, these costs are amortized over twice the number of floating point operations in orthogonal factorization. For each problem type, execution rates increase down a column for either method indicating that processor utilization increases as the problem size is increased. For a given problem type, the best execution rates migrate to columns with higher processors upon increasing the problem size. The execution rate for Gaussian elimination reaches 264 Mflops with 128 processors while that for orthogonal factorization reaches 464 Mflops. In terms of the recent practice of counting a floating point multiply-add pair as two operations, these results amount to an execution rate of over one-half Gflops for sparse Gaussian elimination and about one Gflops for orthogonal factorization. These results indicate that that our approach is indeed feasible. Our implementation uses simple row-wrap mapped distributed dense matrix kernels that tend to perform poorly for both Gaussian elimination with row pivoting and orthogonal factorization with Householder transforms. We are hopeful that performance can be further enhanced by incorporating more recent distributed dense kernels [2] which map blocks of a matrix to a processor (as opposed to rows) to reduce communication requirements.

In Table 8 we present results for the model overdetermined problem based on a $k \times l$ grid. We can estimate theoretically the higher order cost of solving a $2k \times k$ problem as $54k^3$ and that of solving a $k \times k$ problem as $19.7k^3$. We estimate speed-ups using these costs and the actual times observed on one processor for the 64×32 problem. The computed speed-ups are pessimistic since they use the best one processor time and higher order costs. Despite this, we achieve a speed-up of about 85 with 128 processors. It can also be seen that increasing the number of processors and the problem size does indeed increase the execution rates substantially.

5. Conclusions. We have developed a scheme for solving nonsymmetric sparse systems on parallel machines by using our Cartesian nested dissection ordering of the graph of $A^T A$ and the resulting separator tree for both task and data assignment. The data assignment based on the separator tree allows tasks in disjoint subtrees to be computed on disjoint processor subsets. With respect to fill incurred, our approach compares favorably with traditional sequential methods that use either Multiple Minimum Degree or Automatic Nested Dissection as ordering heuristics. The parallel performance observed on the Intel iPSC/860 is encouraging; despite the large communication to computation ratio of the machine, we were able to obtain an execution rate of more than 2 Mflops per processor with 128 processors for Gaussian elimination with row pivoting. Twice that execution rate was observed for orthogonal factorization. This work completes the development of a suite of parallel algorithms for sparse systems reported in [11, 12, 17].

As a next step, we plan to study the effect of using more recent blocked distributed dense kernels. We also plan to investigate the impact of ordering and data assignment strategies on the performance of the overall distributed sparse factorization process.

REFERENCES

[1] E. Chu, A. George, J. Liu, and E. Ng, *Sparspak: Waterloo sparse matrix package user's guide for Sparspak-A*, Tech. Rep. CS-84-36, Department of Computer Science, Univ. of Waterloo, Waterloo, Ontario, Canada, 1984.

[2] J. Demmel, J. Dongarra, R. van de Geijn, and D. Walker, *Lapack for distributed memory architectures: The next generation*, in Sixth Siam Conference on Parallel Processing for Scientific Computing, R. F. Sincovec, D. E. Keyes, M. R. Leuze, L. R. Petzold, and D. A. Reed, eds., Philadephia, PA, 1993, SIAM Publications.

[3] G. Geist and C. Romine, *LU factorization algorithms on distributed-memory multiprocessor architectures*, SIAM J. Sci. Stat. Comput., 9 (1988), pp. 639–649.

[4] A. George, M. T. Heath, and E. Ng, *A comparison of some methods for solving sparse linear least squares pro blems*, SIAM J. Sci. Stat. Comput., 4 (1983).

[5] A. George and J. W. H. Liu, *Householder reflections versus Givens rotations in sparse orthogonal decomposition*, Linear Algebra Appl., 88/89 (1989), pp. 223–238.

[6] A. George and E. Ng, *Sparspak: Waterloo sparse matrix package user's guide for Sparspak-B*, Tech. Rep. CS-84-37, Department of Computer Science, Univ. of Waterloo, Waterloo, Ontario , Canada, 1984.

[7] J. A. George and M. T. Heath, *Solution of sparse linear least squares problems using Givens rotations*, Linear Alg. Appl., 88 (1980), pp. 223–238.

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

[9] J. A. George and E. G.-Y. Ng, *Symbolic factorization for sparse gaussian elimination with partial pivoting*, SIAM J. Sci. Stat. Comput., 8 (1987), pp. 877–898.

[10] J. R. Gilbert and E. Ng, *Predicting structure in nonsymmetric sparse matrix factorizations*, Tech. Rep. ORNL/TM-12204, Oak Ridge National Laboratory, Oak Ridge, Tennessee 37831-8083, 1992.

[11] M. T. Heath and P. Raghavan, *A Cartesian nested dissection algorithm*, Tech. Rep. UIUCDCS-R-92-1772, Department of Computer Science, University of Illinois, Urbana, IL 61801, October 1992.

[12] ———, *Distributed solution of sparse linear systems*, Tech. Rep. UIUCDCS-R-93-1793, Department of Computer Science, University of Illinois, Urbana, IL 61801, February 1993.

[13] B. Hendrickson and D. Womble, *The torus-wrap mapping for dense matrix calculations on massively parallel computers*, Tech. Rep. SAND92-0792, Sandia National Laboratories, Albuquerque, NM 87185, 1992.

[14] J. W.-H. Liu, *On general row mergng schemes for sparse Givens transformations*, SIAM J. Sci. Stat. Comput., 7 (1986), pp. 1190–1211.

[15] K. Mehlhorn, *Graph Algorithms and NP-Completeness*, Springer-Verlag, Berlin Heidelberg, 1984.

[16] P. Raghavan, *Distributed sparse matrix factorization: QR and Cholesky decompositions*, PhD thesis, Department of Computer Science, Pennsylvania State University, University Park, PA, 1991.

[17] ———, *Line and plane separators*, Tech. Rep. UIUCDCS-R-93-1794, Department of Computer Science, University of Illinois, Urbana, IL 61801, February 1993.

[18] G. W. Stewart, *Introduction to Matrix Computations*, Academic Press, New York, 1973.