SIAM J. Matrix Anal. Appl.
Vol. 31, No. 3, pp. 1382–1411

SUPERFAST MULTIFRONTAL METHOD FOR LARGE STRUCTURED LINEAR SYSTEMS OF EQUATIONS*

JIANLIN XIA†, SHIVKUMAR CHANDRASEKARAN‡, MING GU§, AND XIAOYE S. LI¶

Abstract. In this paper we develop a fast direct solver for large discretized linear systems using the supernodal multifrontal method together with low-rank approximations. For linear systems arising from certain partial differential equations such as elliptic equations, during the Gaussian elimination of the matrices with proper ordering, the fill-in has a low-rank property: all off-diagonal blocks have small numerical ranks with proper definition of off-diagonal blocks. Matrices with this low-rank property can be efficiently approximated with semiseparable structures called hierarchically semiseparable (HSS) representations. We reveal the above low-rank property by ordering the variables with nested dissection and eliminating them with the multifrontal method. All matrix operations in the multifrontal method are performed in HSS forms. We present efficient ways to organize the HSS structured operations along the elimination. Some fast HSS matrix operations using tree structures are proposed. This new structured multifrontal method has nearly linear complexity and a linear storage requirement. Thus, we call it a superfast multifrontal method. It is especially suitable for large sparse problems and also has natural adaptability to parallel computations and great potential to provide effective preconditioners. Numerical results demonstrate the efficiency.

Key words. structured direct solver, hierarchically semiseparable matrix, low-rank property, superfast multifrontal method, nested dissection

AMS subject classifications. 15A23, 65F05, 65F30, 65F50

DOI. 10.1137/09074543X

1. Introduction.

In many computational and engineering problems it is critical to solve large structured linear systems of equations. Different structures come from different natures of the original problems or different techniques of discretization, linearization, or simplification. It is usually important to take advantage of the special structures or to preserve the structures when necessary. Direct solvers often provide good chances to exploit the structures. Direct methods are attractive also due to their efficiency, reliability, and generality.

Here we are interested in the discretization of differential equations such as elliptic PDEs on a two-dimensional (2D) finite element mesh (grid) $\mathbb{M}$. We consider discretizations with a regular mesh or, more generally, discretizations with a well-shaped mesh [37, 38, 42], so that nested dissection or its generalizations [19, 20, 23, 32, 42] can be used. The finite element system

$$Ax = b \tag{1.1}$$

associated with the discretization on $\mathbb{M}$ is considered, where A is symmetric positive definite (SPD). Such matrices arise, say, when we apply finite difference or finite ele-

*Received by the editors January 5, 2009; accepted for publication (in revised form) by J. H. Brandts September 2, 2009; published electronically December 4, 2009.
http://www.siam.org/journals/simax/31-3/74543.html

†Department of Mathematics, Purdue University, West Lafayette, IN 47907 (xiaj@math.purdue.edu).

‡Department of Electrical and Computer Engineering, University of California, Santa Barbara, CA 93106 (shiv@ece.ucsb.edu).

§Department of Mathematics, University of California, Berkeley, CA 94720 (mgu@math.berkeley.edu).

¶Lawrence Berkeley National Laboratory, MS 50F-1650, One Cyclotron Rd., Berkeley, CA 94720 (xsli@lbl.gov).

ment techniques to solve 2D linear boundary value problems such as elliptic boundary problems on rectangular domains.

For the purpose of presentations, we focus on 2D problems and demonstrate the potential of our ideas, although it is possible to generalize to other problems. For convenience, we will refer to the following model problem at some places.

Model Problem 1.1. We use the 2D discrete Laplacian on a 5-point stencil as a model problem, where A is a 5-diagonal $n^2 \times n^2$ sparse matrix:

$$A = \begin{pmatrix} T & -I & & 0 \\ -I & \ddots & \ddots & \\ & \ddots & \ddots & -I \\ 0 & & -I & T \end{pmatrix}, \quad T = \begin{pmatrix} 4 & -1 & & 0 \\ -1 & \ddots & \ddots & \\ & \ddots & \ddots & -1 \\ 0 & & -1 & 4 \end{pmatrix}. \tag{1.2}$$

Note we can also use the 9-point or other stencils, since the actual entries or nonzero patterns are not referenced in the descriptions of our method below.

To factorize the discretized matrix A, people typically first order the mesh points. Direct factorization of such a matrix with rowwise or columnwise mesh ordering takes $O(n^4)$ flops [19]. The nested dissection ordering [19] gives an elimination scheme with $O(n^3)$ cost, which is optimal for any ordering in exact arithmetic [31] (ignoring any special techniques such as Strassen's algorithm [41]). But $O(n^3)$ is still large for a big n. Sometimes, iterative methods are cheaper if effective preconditioners are available. But for hard problems good preconditioners can be difficult to find.

1.1. Superfast multifrontal method. Direct solvers have been considered expensive because of the large amount of fill-in even if the original matrix is sparse. To effectively handle the fill-in, some people approximate full matrices in complicated problems with structured matrices such as $\mathcal{H}$-matrices [26, 28, 29], $\mathcal{H}^2$-matrices [3, 27, 30], quasiseparable matrices [17], semiseparable matrices [7, 9, 10], etc. Similarly, when solving discretized PDEs such as elliptic equations, we can develop fast direct solvers by exploiting the rank property and by approximating dense matrices in these problems with compact structured matrices without compromising the accuracy. These approximations are feasible, as we notice that the fill-in during the elimination with certain ordering is actually structured. The fill-in is closely related to the Green's functions via Schur complements. Thus the off-diagonal blocks of the fill-in have small numerical ranks, which has been observed in [1, 2, 8, 24, 25, 35, 45] and other work. Due to this *low-rank property*, we show that the structured approximations of the dense $N \times N$ subproblems in those problems can be solved with a cost of $O(p^2N)$, and this leads to a total complexity of $O(pn^2)$, where p is a constant related to the PDE and the accuracy in the matrix approximations and n is the mesh dimension. Our solver includes both the approximation stage of dense matrices and the direct solution stage using the approximations. We still call the overall procedure a direct solver.

Our work shares similar ideas as those in [1, 2, 8, 24, 25, 35, 45]. Here, we fully integrate sparse matrix techniques (nested dissection) in the context of a supernodal multifrontal method, and we use tree structures for both the overall matrix factorization and the intermediate data structure. The multifrontal method is one of the most important direct methods for sparse matrix solutions [16, 34]. In our direct solver we order the mesh nodes into separators with nested dissection [19] and organize the elimination process with a supernodal multifrontal method. The method eliminates separators and accumulates updates locally following an elimination tree [22, 33].

Moreover, the dense intermediate matrices (fill-in) in the supernodal multifrontal method for many discretized PDEs have the above low-rank property. Those dense matrices are then approximated by tree-structured semiseparable matrices, called *hierarchically semiseparable* (HSS) matrices [11, 13, 14, 46]. HSS matrices have close relation to $\mathcal{H}^2$-matrices in [3, 27, 30]. Many HSS operations such as matrix multiplications and system solutions can be done in nearly linear complexity. HSS matrices feature hierarchical low-rank properties in the off-diagonal blocks. More specifically, we say that a matrix has the low-rank property if all its off-diagonal blocks have small ranks or numerical ranks (Definition 2.6). In our supernodal multifrontal method, all matrix operations are conducted efficiently via HSS approximations. Some basic HSS operations can be found in [11, 13, 14]. In this paper, we provide some new HSS algorithms necessary to convert the multifrontal method into a structured one. These new HSS algorithms provide an innovative way of handling HSS matrix operations by tree techniques.

Both the multifrontal method and the HSS structure have nice hierarchical tree structures. They both take good advantage of dense matrix computations and have efficient storage, good data locality, and natural adaptability to parallel computations. The usage of HSS matrices in the multifrontal method leads to an efficient structured multifrontal method. For problems such as the 5-point or 9-point finite difference Laplacian, this structured multifrontal method has nearly linear complexity. We thus call this method a *superfast multifrontal method*. The method is also memory efficient. By setting a relatively large tolerance in the matrix approximations, the method can also work as an efficient and effective preconditioner. We have developed a software package for the solver and various HSS operations.

1.2. Outline of the paper. This paper is organized as follows. In section 2, we give an introduction to the multifrontal method, nested dissection, and HSS structures. Section 3 demonstrates the low-rank property and presents an overview of our new superfast multifrontal method. The detailed superfast structured multifrontal algorithm is discussed in section 4. Two major steps are covered: structured elimination of separators and structured matrix assembly (called extend-add). Some new HSS operations are proposed. Section 5 demonstrates the efficiency of the superfast multifrontal method with numerical experiments in terms of the model problem and a linear elasticity problem. Section 6 gives some general remarks.

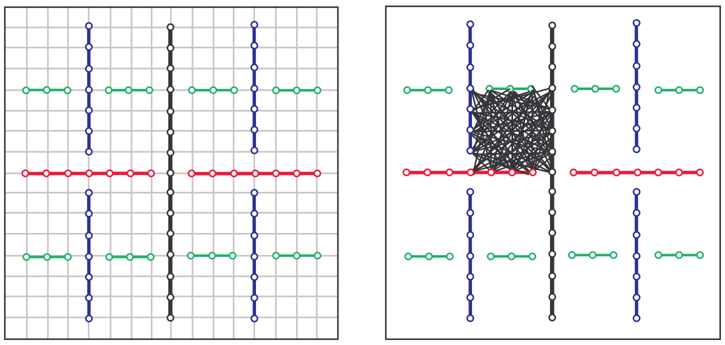
2. Review of the multifrontal method and HSS representations.

In this section, we briefly review the multifrontal method and HSS representations which build our superfast structured multifrontal method.

2.1. Multifrontal method with nested dissection ordering.

In the direct factorization of a sparse matrix, usually, the rows and columns of the matrix are first ordered to reduce fill-in. Nested dissection [19] is an important method for finding an elimination ordering. Consider a discretized matrix and its associated mesh. Nested dissection orders mesh points with the aid of *separators*. A separator is a small set of mesh points whose removal divides the rest of the mesh or submesh into two disjoint pieces. The mesh is recursively partitioned with multiple levels of separators. Lower level separators are ordered before upper level ones; see Figure 2.1(i). Here, by a "level" we mean a set of separators at the same level of partition.

After the nodes are ordered, we compute the Cholesky factorization $A = LL^T$ by Gaussian elimination, which corresponds to the elimination of the mesh points



(i) Partition with separators (ii) Connections of points during elimination

FIG. 2.1. *Separators in nested dissection and the connections of mesh points.*

(unknowns) from lower levels to upper levels. Mesh points may get connected during the elimination (Figure 2.1(ii)). The elimination of a mesh point pairwise connects all its neighbor points [19, 39].

The factorization process can be conducted following the multifrontal method [16, 34]. The central idea of the multifrontal method is to reorganize the overall factorization of A into partial updates and factorizations of small dense matrices. It has been widely used in numerical methods for PDEs, optimization, fluid dynamics, and other areas.

Suppose $j-1$ steps of factorizations of A are finished. Some portions of the first $j-1$ columns of L contribute to later computations in the form of *outer-product updates* [34]. The nonzero portion of column j of A and some early outer-product contributions are assembled together by an operation called *extend-add*. The result matrix is called a *frontal matrix* $\mathcal{F}_j$. One step of factorization of $\mathcal{F}_j$ gives the jth column of L. The Schur complement is called the *update matrix*. See [34] for a formal discussion of the procedure. To conveniently consider how the update contributions are passed, a powerful tool called *elimination tree* is used. The following definitions can be found in [33, 34, 40, etc.].

DEFINITION 2.1. *The elimination tree $T(A)$ of an $N \times N$ matrix A is the tree structure with N nodes $\{1, \ldots, N\}$ such that node p is the parent of j if and only if $p = \min\{i > j | l_{ij} \neq 0\}$, where $A = LL^T$ and $L = (l_{ij})_{N \times N}$ is lower triangular.*

In addition, the concept of an assembly tree is given in [34]. In this work, we do not distinguish between these two types of trees. We also assume the elimination follows the *postordering* of the elimination tree.

We use a *supernodal version* of the multifrontal method together with nested dissection to solve the discretized problems. Each separator in nested dissection is considered to be a node in the postordering elimination tree. The separators are put into different levels of the tree (Figure 2.2).

During elimination, the separators are eliminated following the postordering elimination tree. The elimination of a separator will connect all its neighbor separator pieces. For a separator i, let $p_1, p_2, \ldots, p_k$ be the pieces of the uneliminated neighbor separators of i which are directly or indirectly connected to i (due to matrix factorization). We say that the pieces $\{i, p_1, p_2, \ldots, p_k\}$ form an *element* and that i is the pivot separator of this element. Figure 2.3 shows two examples.

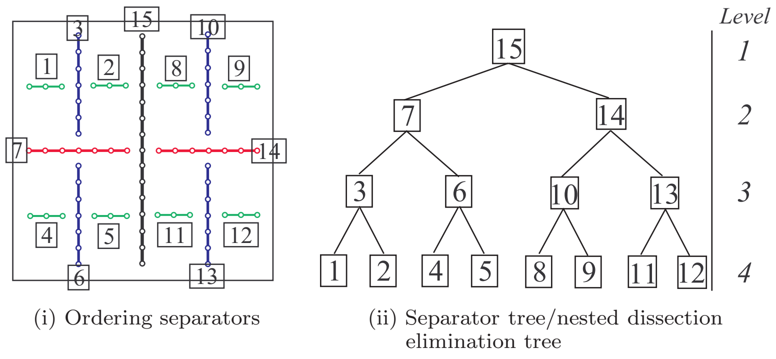


FIG. 2.2. *Ordering separators.*

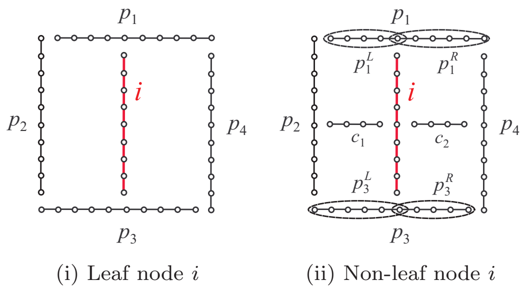


FIG. 2.3. *Examples of elements. The ordering of* p_1, p_2, p_3, p_4 *may not necessarily follow their order in the elimination tree.*

If separator i is a bottom level separator in nested dissection (or a leaf node in the elimination tree) (Figure 2.3(i)), the frontal matrix $\mathcal{F}_i$ is directly formed from A:

$$\mathcal{F}_i \equiv \mathcal{F}_i^0 = \left(\begin{array}{c|c} A_{ii} & A_{ip_1} \cdots A_{ip_k} \\ \hline A_{p_1 i} & \\ \vdots & 0 \\ A_{p_k i} & \end{array}\right), \quad \mathcal{U}_i = -\begin{pmatrix} A_{p_1 i} \\ \vdots \\ A_{p_k i} \end{pmatrix} A_{ii}^{-1} \left(A_{ip_1} \cdots A_{ip_k} \right). \tag{2.1}$$

The elimination of separator i provides the block column in L corresponding to i, and the Schur complement is the ith update matrix $\mathcal{U}_i$.

If separator i is not a leaf node (Figure 2.3(ii)), we assume it has two children c_1 and c_2. The update matrices $\mathcal{U}_{c_1}$ and $\mathcal{U}_{c_2}$ represent contributions from the subtrees rooted at c_1 and c_2, respectively. Then $\mathcal{F}_i$ is obtained by assembling $\mathcal{F}_i^0$, $\mathcal{U}_{c_1}$, and $\mathcal{U}_{c_2}$ with the extend-add process, and the elimination of separator i yields $\mathcal{U}_i$:

$$\mathcal{F}_i = \mathcal{F}_i^0 \updownarrow\!\!\leftrightarrow \mathcal{U}_{c_1} \updownarrow\!\!\leftrightarrow \mathcal{U}_{c_2} = \begin{pmatrix} L_{ii} & 0 \\ L_{Bi} & I \end{pmatrix} \begin{pmatrix} L_{ii}^T & L_{Bi}^T \\ 0 & \mathcal{U}_i \end{pmatrix}, \tag{2.2}$$

where B denotes the element boundary $\{p_1, p_2, \ldots, p_k\}$.

For convenience, when presenting the ideas of handling the connections of separators, we use the situation $k = 4$ as in the model problem. Situations with a general k can be similarly discussed (subsection 4.6). Here, as shown in Figure 2.4, we say that the separator pieces $\{c_1,\ p_1^L,\ p_2,\ p_3^L,\ i\}$ form the left child element and $\{c_2,\ p_1^R,\ p_2,\ p_3^R,\ i\}$ form the right child element.

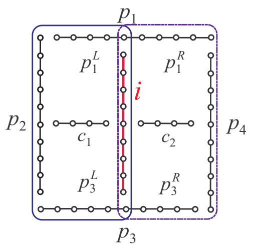
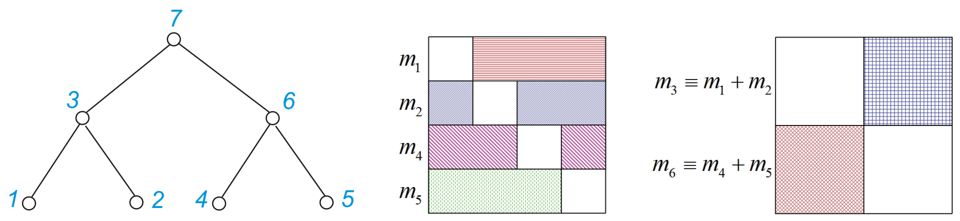


FIG. 2.4. *In extend-add, separator pieces in the left and right child elements, marked by the solid-line oval box and the dashed-line oval box, respectively.*



(i) Postordering of a binary tree (ii) 2nd level HSS blocks (iii) 1st level HSS blocks

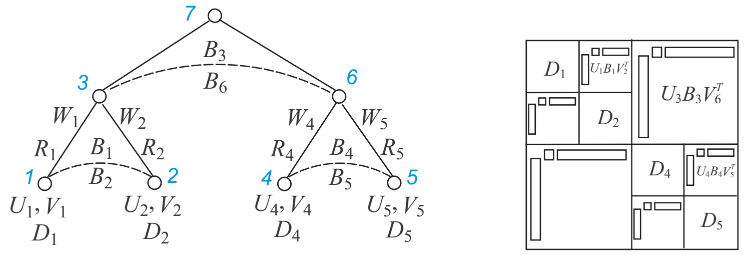
FIG. 2.5. *Example: A binary tree with postordering and two levels of HSS off-diagonal blocks of a matrix, where the indices follow the postordering of the tree.*

Recursive application of the above procedure to all nodes of the elimination tree leads to a supernodal multifrontal method. The supernodal multifrontal method with nested dissection for factorizing A in (1.2) costs $O(n^3)$ flops. The number of nonzeros in the Cholesky factor is $O(n^2 \log_2 n)$. A stack of size $O(n^2)$ is used to store the update matrices.

2.2. Hierarchically semiseparable structures. Semiseparable or quasi-separable structures have attracted a lot of interests in the recent years. In our superfast multifrontal method, frontal matrices and update matrices are approximated by semiseparable matrices. Semiseparable forms of upper level frontal matrices are obtained from lower level ones recursively. In this subsection, we review a tree-structured semiseparable representation. Note this tree structure is used for each frontal matrix and is not associated with the outer assembly tree. Thus, this subsection can be understood independent of the multifrontal method.

There are different definitions for semiseparable matrices [17, 18, 43, 44]. One definition often used is based on the low-rankness of appropriate off-diagonal blocks. Here we use the HSS off-diagonal blocks in [11, 12, 13, 14], as shown in the example in Figure 2.5. We first define HSS blocks with the aid of a full binary tree (a binary tree where each node except the root has exactly one sibling) and its postordering.

DEFINITION 2.2 (HSS blocks). *HSS blocks are block rows or columns excluding the diagonal parts defined at different levels of splittings of a matrix as follows. Given a full binary tree with its postordering, an $N \times N$ matrix H, and a partition sequence $\{m_{i_j}\}_{j=1}^k$, where $i_j, j = 1, 2, \ldots, k$, are the leaf nodes of the tree and $\sum_{j=1}^k m_{i_j} = N$, partition H into k block rows (columns) following $\{m_{i_j}\}_{j=1}^k$ so that block row (column) j has m_{i_j} rows (columns) of H. Any block row (column) i excluding the $m_{i_j} \times m_{i_j}$ diagonal block is called a bottom level HSS (off-diagonal) block. Associate*



(i) HSS tree (ii) Hierarchical form of the matrix

FIG. 2.6. *An HSS tree corresponding to Figure* 2.5 *and the structured form of the matrix.*

with each leaf a bottom level HSS block. An HSS block for an upper level node is defined recursively from child HSS blocks but with the appropriate diagonal block removed.

We emphasize that postordered trees are used in this paper so that the HSS blocks in Figure 2.5(ii) are indexed following the ordering of the tree nodes. This significantly simplifies the HSS notation below and the coding, as discussed in [11], since for each node only one index is needed instead of two or three as in [12, 13, 14]. A full binary tree with k leaves has totally $2k-1$ nodes, where node 1 is the first leaf and $2k-1$ is the root. The binary tree used in the above definition is called an *HSS tree* (Figure 2.6(i), also called a merge tree in [12, 13, 14]), which helps define the HSS structure.

DEFINITION 2.3 (HSS tree and HSS representation). *An HSS tree* $\mathbf{T} = (\mathbf{V}, \mathbf{E})$ *that defines an HSS representation for a matrix H is the binary tree in Definition* 2.2 *and is further defined as follows. Let node* 1 *be the first leaf,* $2k-1$ *be the root, and* i_j, $j = 1, 2, \ldots, k$, *be the leaves of* $\mathbf{T}$. *Each node* $i \in \mathbf{V}$ $(i < 2k-1)$ *is associated with matrices* $D_i, U_i, V_i, R_i, W_i, B_i$, *which are called generators of H. The HSS representation of H is given by the generators* $\{R_i, W_i, B_i\}_{i=1}^{2k-2}$ *and* $\{D_{i_j}, U_{i_j}, V_{i_j}\}_{j=1}^{k}$, *which satisfy the recursive definition of upper level generators* D_i, U_i, *and* V_i*:*

$$\begin{aligned} D_i &= \begin{pmatrix} D_{c_1} & U_{c_1}B_{c_1}V_{c_2}^T \\ U_{c_2}B_{c_2}V_{c_1}^T & D_{c_2} \end{pmatrix}, \quad i \in \mathbf{V} \textit{ is a nonleaf node,} \\ U_i &= \begin{pmatrix} U_{c_1}R_{c_1} \\ U_{c_2}R_{c_2} \end{pmatrix}, \quad V_i = \begin{pmatrix} V_{c_1}W_{c_1} \\ V_{c_2}W_{c_2} \end{pmatrix}, \quad i \in \mathbf{V}\backslash\{2k-1\} \textit{ is a nonleaf node,} \end{aligned} \tag{2.3}$$

so that at the top level, $D_{2k-1} \equiv H$, *where* c_1 *and* c_2 *represent the left and right children of i, respectively.*

Remark 2.4. Generators are indexed following the postordering of the tree nodes. The generators D_i, U_i, V_i for a nonleaf node i are not explicitly stored. R_i and W_i are empty matrices if i is a direct child of the root.

The following is a block 4×4 HSS example corresponding to Figure 2.5(ii) and Figure 2.6:

$$H = \left(\begin{array}{cc|cc} D_1 & U_1B_1V_2^T & U_1R_1B_3W_4^TV_4^T & U_1R_1B_3W_5^TV_5^T \\ U_2B_2V_1^T & D_2 & U_2R_2B_3W_4^TV_4^T & U_2R_2B_3W_5^TV_5^T \\ \hline U_4R_4B_6W_1^TV_1^T & U_4R_4B_6W_2^TV_2^T & D_4 & U_4B_4V_5^T \\ U_5R_5B_6W_1^TV_1^T & U_5R_5B_6W_2^TV_2^T & U_5B_5V_5^T & D_4 \end{array}\right) \begin{array}{c} m_1 \\ m_2 \\ m_4 \\ m_5 \end{array}. \tag{2.4}$$

To see the hierarchical structure of (2.4), we can write H as

$$H = \left(\begin{array}{c|c} D_3 & U_3B_3V_6^T \\ \hline U_6B_6V_3^T & D_6 \end{array}\right) \begin{array}{c} m_1+m_2 \\ m_4+m_5 \end{array},$$

corresponding to Figure 2.5(iii), where the generators are obtained by setting $i = 3,\ c_1 = 1,\ c_2 = 2$ in (2.3). However, only the generators in (2.4) are explicitly stored.

Remark 2.5. In the HSS representation,

- each U_i is an appropriate column basis matrix for an HSS block row. For example, the second level HSS block row associated with node $i = 1$ is

$$U_1 \begin{pmatrix} B_1 V_2^T & R_1 B_3 \begin{pmatrix} W_4^T V_4^T & W_5^T V_5^T \end{pmatrix} \end{pmatrix}.$$

- we can verify the following [11]: to identify a block of H, say, the $(2, 3)$ block in (2.4), we can use the directed path connecting the 2nd and 3rd nodes at the bottom level (nodes 2 and 4 as marked) of the HSS tree:

$$\overset{U_2}{2} \xrightarrow{R_2} 3 \xrightarrow{B_3} 6 \xrightarrow{W_4^T} \overset{V_4^T}{4}.$$

- for a symmetric HSS matrix, we can set $U_i = V_i, R_i = W_i$, and $B_i = B_j^T$, $i = 1, 2, \ldots, 2k-2$, where j represents the sibling of each i.

Theoretically, an HSS form representation can be constructed for any matrix H [11, 14] and an appropriate HSS tree. However, such a representation is generally more useful when the HSS blocks have small (numerical) ranks.

Definition 2.6 (numerical rank and HSS rank). *In this work, the numerical rank of any matrix block with a relative or absolute tolerance τ is the rank obtained by applying rank revealing QR factorizations* [5, 6] *or τ-accurate SVD (SVD with a tolerance τ for singular values) to the block. For a matrix and an HSS tree, the maximum of the numerical ranks of the HSS blocks at all tree levels is called the HSS rank (with a given τ) of the matrix. Later, we say a matrix is hierarchically separable if its HSS rank is small with a given τ.*

For an HSS matrix with a small HSS rank p, if all B_i generators in its HSS representation have sizes close to p, we say that the HSS form is *compact*. It is shown in [11, 14] that for a matrix in compact HSS form, nearly linear complexity system solvers exist. Many other HSS matrix operations such as structure generation, compression, etc., are also very efficient. The reader is referred to [11, 12, 13, 14] for more details on HSS representations.

3. Superfast multifrontal method: Low-rank property and overview.

Notice that in the multifrontal method for discretized matrices the frontal and update matrices are generally dense because of the mutual connections among mesh nodes (Figure 2.1(ii)). The elimination together with the extend-add operation on such a dense $N \times N$ matrix typically take $O(N^3)$ flops in exact arithmetic. Here we consider approximations of these dense matrices. Approximations of dense matrices are feasible in solving linear systems derived from discretizations of certain PDEs such as elliptic equations, as we discover that low-rank properties exist in these problems. Similar results can also be found in [1, 2, 8, 24, 25, 35].

In this work, we take advantage of the hierarchical tree structures of both HSS matrices and the multifrontal method. We have developed a series of efficient HSS operations [11, 14]. Additional HSS operations necessary for our superfast multifrontal method will be presented here. With these techniques, we are able to produce a structured multifrontal method, and we reduce the total complexity for solving discretized problems such as (1.2) from $O(n^3)$ to $O(pn^2)$ and storage from $O(n^2 \log_2 n)$ to $O(n^2 \log_2 p)$, where p is a parameter related to the problem and the tolerance for matrix approximations.

TABLE 3.1
Numerical ranks with different relative tolerances τ of four F_{iB} blocks from mesh dimensions $n = 127{,}255{,}511$ and 1023, respectively. Note the size of F_{iB} can be larger than n.

τ		10^{-2}	10^{-4}	10^{-6}	10^{-8}	10^{-10}
size(F_{iB})	31×161	8	13	17	20	23
	63×321	9	15	21	25	29
	127×641	9	18	24	30	36
	255×1281	10	20	28	35	42

TABLE 3.2
HSS ranks of an order 1023 block F_{ii} with different τ, where the bottom level HSS block row size is about 16 and a perfect binary HSS tree with 64 nodes is used.

τ	10^{-2}	10^{-4}	10^{-6}	10^{-8}	10^{-10}
HSS rank	6	12	17	21	26

3.1. Off-diagonal numerical ranks. It has been shown in [1, 2, 8, 24, 25] that the low-rank property exists in the LU factorizations of finite-element matrices from elliptic operators and some other problems. Here in the context of the supernodal multifrontal method, we show some rank results of the frontal and update matrices.

For a separator i and all its neighbors (denoted B) as shown in Figure 2.3, we order them and their interior nodes properly (Definition 3.1 below). The corresponding frontal matrix $\mathcal{F}_i$ has the following form:

$$\mathcal{F}_i = \begin{pmatrix} F_{ii} & F_{iB} \\ F_{Bi} & F_{BB} \end{pmatrix} = \begin{pmatrix} L_{ii} & 0 \\ L_{Bi} & I \end{pmatrix} \begin{pmatrix} L_{ii}^T & L_{Bi}^T \\ 0 & \mathcal{U}_i \end{pmatrix}, \tag{3.1}$$

where the elimination of separator i gives the update matrix $\mathcal{U}_i$. For the frontal and update matrices in the supernodal multifrontal method for solving Model Problem 1.1, we have the following critical rank observations:

- The off-diagonal block F_{iB} has a small numerical rank.
- The HSS blocks of F_{ii} have small numerical ranks.
- The HSS blocks of $\mathcal{U}_i$ have small numerical ranks.

Some rank results are reported as follows.

(1) *Numerical rank of* F_{iB}. We choose some frontal matrices $\mathcal{F}_i$ and compute the numerical rank of F_{iB} in each $\mathcal{F}_i$. Table 3.1 shows the rank results.

(2) *Off-diagonal numerical ranks of* F_{ii}. We then test the HSS ranks of F_{ii} with different relative tolerances. As an example, we use the frontal matrix corresponding to the top level separator of a 1023×1023 mesh. In such a situation, $\mathcal{F}_i (\equiv F_{ii})$ has order 1023. We choose a fixed block row size and make all bottom level HSS off-diagonal blocks to have approximately the same row dimension, so that the HSS tree is a perfect binary tree. (As an example, when there are four block rows, a binary tree as in Figure 2.6 is used. Again, this binary tree is not related to the elimination tree.) Table 3.2 shows the HSS ranks, which are relatively small as compared with the size of F_{ii}.

(3) *Off-diagonal numerical ranks of* $\mathcal{U}_i$. Similarly, Table 3.3 shows HSS ranks of an update matrix $\mathcal{U}_i$ with different tolerances. The mesh dimension is $n = 1023$. Similar situations hold for the frontal matrices. When n is larger, the low-rank property is more significant.

The low-rank property has certain physical background. For example, the paper [4] considers a 2D physical model consisting of a set of particles with pairwise interactions satisfying Coulomb's law. The authors define *well-separated* sets of particles to

TABLE 3.3
HSS ranks of a 1023×1023 *update matrix* $\mathcal{U}_i$ *with different* τ*, where the bottom level HSS block row size is about* 16 *and a perfect binary HSS tree with* 64 *nodes is used.*

τ	10^{-2}	10^{-4}	10^{-6}	10^{-8}	10^{-10}
HSS rank	6	12	17	22	26

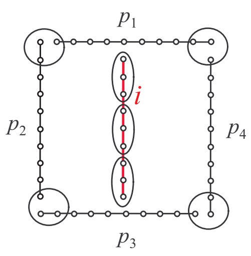


FIG. 3.1. *Examples of well-separated sets in an element.*

be the sets that have strong interactions within sets but weak ones between different sets. Here when we consider 2D meshes for discretized problems, we have a similar situation. In Figure 2.1(ii) the mesh points in the current-level element are mutually connected. But the connections differ for different points. We can think of the points closer to each other or not well separated to have stronger connections; see Figure 3.1. For theoretical analysis of the low-rank property, see, e.g., [1, 2, 8, 24, 25].

3.2. Overview of the structured extend-add process and the structured multifrontal method. We can take advantage of the previous low-rank property in the multifrontal process to get a new structured method. There are two major tasks:

- to replace the traditional dense Cholesky factorization by a structured one;
- to develop a structured extend-add process.

We use HSS matrices to approximate frontal and update matrices. HSS matrices can be quickly factorized due to the low-rank property. HSS forms are accumulated bottom-up along the assembly tree.

The structured extend-add process is relatively complicated, since the mesh nodes and separators are generally not consistent with the HSS block partitions. We consider a separator i, its four neighbor pieces p_1, p_2, p_3, p_4 at upper levels of the assembly tree, and its children c_1 and c_2 as in Figure 2.3(ii). In the traditional multifrontal method, the frontal matrix $\mathcal{F}_i$ is obtained from the extend-add operation (2.2)

$$\mathcal{F}_i = \mathcal{F}_i^0 \Updownarrow\!\!\Leftrightarrow \mathcal{U}_{c_1} \Updownarrow\!\!\Leftrightarrow \mathcal{U}_{c_2} \equiv \mathcal{F}_i^0 + \hat{\mathcal{U}}_{c_1} + \hat{\mathcal{U}}_{c_2}, \tag{3.2}$$

where each $\hat{\mathcal{U}}_{c_i}$ is a *subtree update matrix* obtained from $\mathcal{U}_{c_i}$ by matching indices to $\mathcal{F}_i^0$ and inserting zero entries [34]. The matrices in (3.2) take the nonzero patterns as illustrated in Figure 3.2.

There are three key issues in developing the structured extend-add process.

(1) *Uniform ordering.* Firstly, in order to effectively conduct (3.2) and to handle the interactions of elements, we need to match the ordering of separators and mesh points at different levels. The ordering can be predetermined in a symbolic factorization stage. We define the following uniform ordering whose effectiveness in revealing the low-rank property is shown by our numerical experiments.

DEFINITION 3.1 (uniform ordering). *The separator pieces and mesh points within an element are uniformly ordered if the neighbors are ordered counterclockwise as*

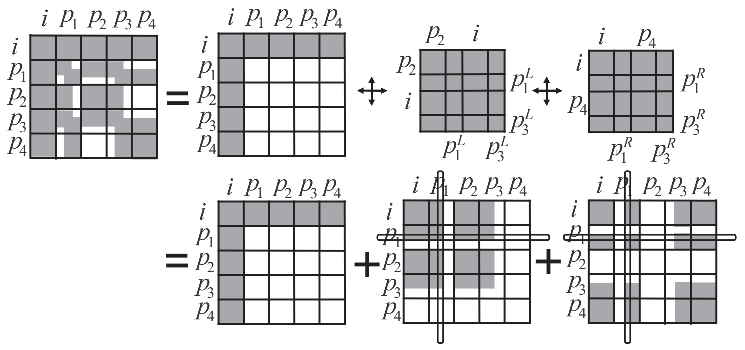


FIG. 3.2. *Matrices in extend-add* (3.2), *where the* + *shaped bars correspond to the overlap* $p_1^L \cap p_1^R$ *in separator* p_1.

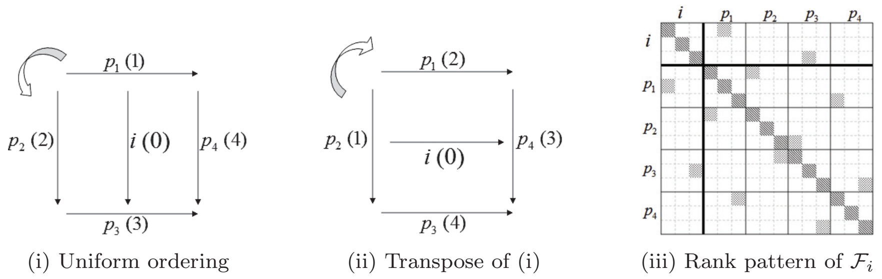


FIG. 3.3. *Uniform ordering of neighbors and mesh points and the resulting rank pattern of* $\mathcal{F}_i$.

TABLE 3.4
Making child element separator pieces consistent with the current level pieces i, p_1, p_2, p_3, p_4 *following the uniform ordering.*

Matrix	Uniform ordering	Permutation	Padding zero blocks
$\mathcal{F}_i$	i, p_1, p_2, p_3, p_4	/	/
$\mathcal{U}_{c_1}$	p_2, p_1^L, i, p_3^L	i, p_1^L, p_2, p_3^L	$i, \{p_1^L, 0\}, p_2, \{p_3^L, 0\}, 0$
$\mathcal{U}_{c_2}$	i, p_1^R, p_4, p_3^R	i, p_1^R, p_3^R, p_4	$i, \{0, p_1^R\}, 0, \{0, p_3^R\}, p_4$

shown in Figure 3.3(i) *(or clockwise in Figure* 3.3(ii)*, which can be considered as a transpose of Figure* 3.3(i)*), and the mesh points inside each separator piece are ordered following the natural ordering of mesh points (left-right and top-down).*

According to the uniform ordering for Figure 2.3(ii), we have the ordering of the separator pieces and their corresponding matrices as shown in the first two columns of Table 3.4. Clearly, the neighbor orderings associated with $\mathcal{U}_{c_1}$ and $\mathcal{U}_{c_2}$ do not match with $\mathcal{F}_i$. Thus, permutations of the separator pieces in the two child elements are needed for (3.2); see the third column of Table 3.4.

(2) *Incompatible separator pieces.* Secondly, the separator pieces for $\mathcal{U}_{c_1}$ are not fully compatible with those for $\mathcal{U}_{c_2}$. That is, separator pieces in one child element may not appear in the other. For example, p_2 appears in the left child element but not in the right one. Thus, we need to insert some zero blocks into $\mathcal{U}_{c_1}$ and $\mathcal{U}_{c_2}$. In terms of the separators, we attach a zero piece to p_1^L so that the length of $\{p_1^L, 0\}$ is consistent with p_1. Other separators are processed similarly; see the fourth column of Table 3.4.

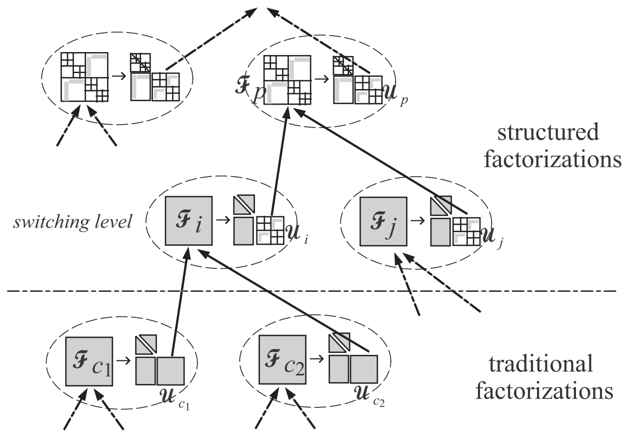


FIG. 3.4. *Illustration of the superfast multifrontal method, where unstructured matrices are in dark color and others are structured. In each oval box, a frontal matrix is partially factorized and, the update matrix is computed.*

(3) *Overlaps.* Lastly, the child elements share (parts of) the neighbors. The left and right child elements share the entire separator i, which becomes the pivot separator in the upper element. The piece p_1^L in the left child element and p_1^R in the right one satisfy $p_1^L \cup p_1^R = p_1$. In general, $p_1^L \cap p_1^R$ is nonempty (Figures 2.3(ii), 2.4, and 3.2) and is shared by both the left and right child elements. Furthermore, $p_1^L \cap p_1^R$ may not always correspond to an entire HSS block row/column. Then certain blocks may need to be split and merged with others. The techniques in subsection 4.2.2 can be used. A similar situation holds for p_3^L and p_3^R.

All the matrix operations are done in HSS forms to provide a structured extend-add process. After $\mathcal{F}_i$ is formed, we eliminate the pivot separator i and compute the Schur complement with the fast HSS algorithms in [11]. The structured extend-add is used again, and the process repeats.

Before going into the details of the HSS operations, we give an overview of the structured multifrontal algorithm. A pictorial illustration is shown in Figure 3.4.

ALGORITHM 3.2 (STRUCTURED SUPERNODAL MULTIFRONTAL METHOD).

1. Use nested dissection to order the nodes. Build postordering elimination tree of separators.
2. Do traditional factorization and extend-add at certain bottom levels.
3. At a switching level, construct HSS approximations of update matrices and do structured extend-add.
4. Following the nested dissection ordering, do structured factorization and extend-add at each upper level.
   (a) Eliminate a separator by factorizing the pivot blocks of the structured frontal matrix to obtain the structured update matrix.
   (b) Do structured extend-add and repeat.

Two layers of trees are used: the outer layer elimination tree and an inner layer HSS tree for each separator. Steps 4(a) and 4(b) are the two major structured operations.

4. Superfast multifrontal method: Detailed algorithm.

According to the previous section, in the supernodal multifrontal method, the update matrices and frontal matrices are approximated by compact HSS matrices. There are two major

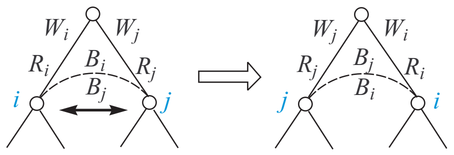


FIG. 4.1. *Permuting two subtrees.*

tasks: the structured elimination step 4(a) of Algorithm 3.2 and the HSS structured extend-add in step 4(b) of Algorithm 3.2. In this section we briefly review a fast generalized HSS factorization in [11] and then discuss in detail some new HSS algorithms which build the HSS structured extend-add.

4.1. Fast generalized HSS Cholesky factorization. The elimination of a separator with order N can be done by the fast generalized HSS Cholesky factorization in [11] in $O(p^2N)$ flops, where p is an appropriate HSS rank.

The fast generalized HSS Cholesky factorization computes explicit factorizations of HSS matrices where the factors (called *generalized HSS factors*) consist of triangular matrices, permutations, and other orthogonal matrices. For a given SPD HSS matrix with generators $\{D_j\}, \{U_j(\equiv V_j)\}, \{W_j(\equiv R_j)\}, \{B_j\}$, the major steps include the following:

1. Introduce zeros into off-diagonal blocks by compressing U_j generators. The compression is done by rank revealing QR factorizations or τ-accurate SVD.
2. Partially factorize D_j. The subblock of D_j corresponding to zero off-diagonal entries are eliminated.
3. Merge the uneliminated subblock of D_j with that of the sibling of j in the HSS tree. Pass the block to the parent.
4. The HSS matrix is reduced to a new one with a smaller size and fewer blocks. Repeat the process.

This process is applied to F_{ii} of an HSS form frontal matrix $\mathcal{F}_i$ as in (3.1). This elimination corresponds to the removal of the subtree for F_{ii} from the HSS tree for $\mathcal{F}_i$. Or, more specifically, after this elimination the HSS subtree for F_{ii} shrinks to one single node. The generators associated with this single node are used to update the rest nodes. This leads to the Schur complement, or the update matrix $\mathcal{U}_i$ in HSS form. The details are given in [11].

Note that L_{ii} in (3.1) is now the generalized HSS factor (see [11] for an example), and $\mathcal{U}_i = F_{BB} - L_{ii}L_{ii}^T$ is essentially computed by a structured low-rank update. This update is fast because F_{BB} and L_{ii} share some common generators.

4.2. Some basic HSS operations needed in structured extend-add. In order to convert the standard extend-add process into a structured one, we need some basic HSS operations, which are used to address the issues discussed in subsection 3.2. These operations include permuting, merging/splitting, and inserting/deleting HSS blocks in an HSS matrix.

4.2.1. Permuting HSS blocks. It is convenient to permute an HSS matrix by permuting its HSS tree. We can generally get the new HSS form of the permuted matrix by updating just a few generators. For example, consider permuting two neighbor block rows/columns at a certain level of the HSS matrix. This corresponds to the permutation of two neighbor HSS subtrees with roots being siblings; see Figure 4.1.

Thus in this simple situation, we can directly exchange generators associated with i and j, and all their children, without updating the matrices. The new matrix

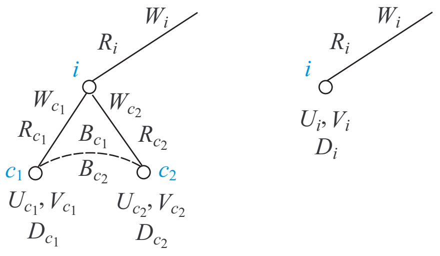


FIG. 4.2. *Merging and splitting nodes of an HSS tree.*

is still in HSS form. For more complicated situations, say, if the two subtrees are not neighbors, we can identify the path connecting the two subtrees and update the matrices associated with the nodes in the path and those connected to the path. As this varies for different situations, we come back to it in subsection 4.3, where the permutations are specifically designed for our superfast multifrontal method.

4.2.2. Merging and splitting HSS blocks. We first look at a simple situation.

(1) *Basic merging and splitting.* The hierarchical structure of HSS matrices makes it very convenient to merge and to split HSS blocks.

For a node i with two children c_1 and c_2 in an HSS tree (Figure 4.2), we can merge c_1 and c_2 and update node i as in (2.3) in Definition 2.3. On the other hand, if we want to split i into c_1 and c_2, then we need to find D_{c_k}, U_{c_k}, V_{c_k}, R_{c_k}, W_{c_k}, B_{c_k}, $k = 1, 2$ such that (2.3) is satisfied. First, partition U_i, V_i, D_i conformally as

$$(4.1) \qquad U_i = \begin{pmatrix} U_{i;1} \\ U_{i;2} \end{pmatrix}, \quad V_i = \begin{pmatrix} V_{i;1} \\ V_{i;2} \end{pmatrix}, \quad D_i = \begin{pmatrix} D_{c_1} & D_{i;1,2} \\ D_{i;2,1} & D_{c_2} \end{pmatrix}.$$

Then compute QR factorizations

$$(4.2) \qquad \begin{pmatrix} U_{i;1} & D_{i;1,2} \end{pmatrix} = U_{c_1} \begin{pmatrix} R_{c_1} & T_1 \end{pmatrix}, \quad \begin{pmatrix} T_1^T \\ V_{i;2} \end{pmatrix} = V_{c_2} \begin{pmatrix} B_{c_1}^T \\ W_{c_2} \end{pmatrix},$$

$$(4.3) \qquad \begin{pmatrix} U_{i;2} & D_{i;2,1} \end{pmatrix} = U_{c_2} \begin{pmatrix} R_{c_2} & T_2 \end{pmatrix}, \quad \begin{pmatrix} T_2^T \\ V_{i;1} \end{pmatrix} = V_{c_2} \begin{pmatrix} B_{c_2}^T \\ W_{c_1} \end{pmatrix}.$$

Equations (4.1)–(4.3) provides all the necessary new generators.

(2) *Advanced merging and splitting.* There are more complicated situations that are very useful. Sometimes, we need to maintain the tree structure during merging or splitting. For an HSS matrix H, we consider splitting a piece from a leaf node i of the HSS tree of H and then to merging that piece with a neighbor j. We look at a general situation where i and j are not siblings. Without loss of generality, we make two simplifications. One is that the matrix is symmetric; another is that block j is an empty block (or zero block whose size is to be set by the splitting). It suffices to look at an example in Figure 4.3, where $i = 5$ and $j = 10$.

We first split node i into two child nodes c_1 and c_2 by the method above: (4.1)–(4.3). Then move c_2 to the position of j. After that we merge c_1 into i.

The details are as follows. Identify the path connecting nodes c_2 and j:

$$\mathbf{p}(c_2, i) : c_2 - 5 - 6 - 7 - 8 - 9 - 14 - 13 - 12 - 11 - 10.$$

We observe that in order to get a new HSS representation for H (denoted $\tilde{H}$), all the generators associated with the nodes in this path and those directly connected to

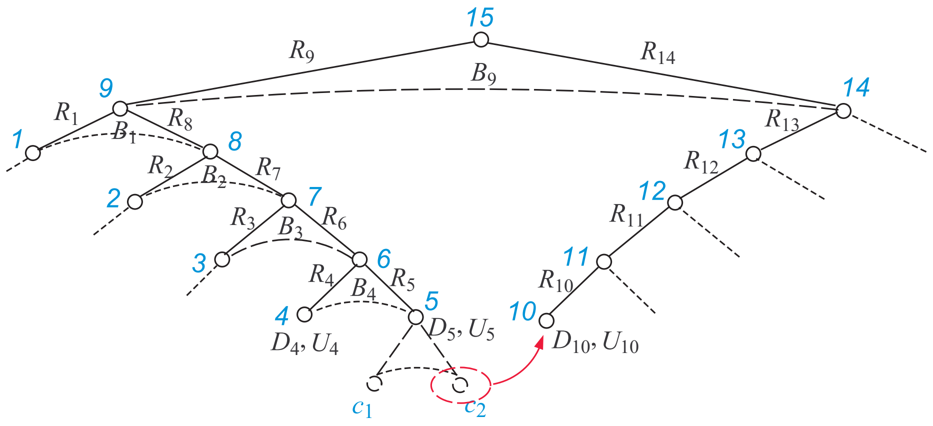


FIG. 4.3. *Splitting a block (node 5) of an HSS tree, where c_1 and c_2 are virtual nodes.*

it should be updated. We use tilded notation for the generators of $\tilde{H}$, with c_1 not merged to i yet. Call the subtrees rooted at nodes 9 and 14 left and right subtrees, respectively. To get the HSS tree for $\tilde{H}$, we consider the following connections, where by a *connection* we mean the product of the generators associated with the path connecting two nodes.

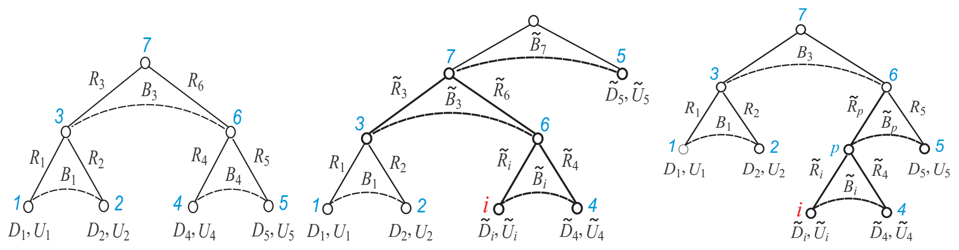
- The connection between nodes 15 and c_2 should be transferred to the connection between 15 and 10.
- The connection between node c_2 and each node $k \in \{1,2,3,4,c_1\}$ that is directly connected to the path $\mathbf{p}(c_2,i)$ should be transferred to the connection between nodes 10 and k.
- The connection between any two nodes $k_1, k_2 \in \{1,2,3,4,c_1\}$ that are directly connected to the path $\mathbf{p}(c_2,i)$ should remain the same.
- The connections between node 15 and each node $k \in \{1,2,3,4,c_1\}$ that is directly connected to the path $\mathbf{p}(c_2,i)$ should remain the same.

All these relations can be reflected by appropriate products of generators associated with the nodes; see [46] for the details. We can assemble all the matrix products into one single equation

$$\left(\begin{array}{cccc|c|c}
0 & & & & & \tilde{R}_{14}^T\tilde{U}_{14}^T \\
\hline
 & & & & \tilde{R}_1(\tilde{R}_9 & \tilde{B}_9\tilde{U}_{14}^T) \\
 & & & \tilde{R}_2(\tilde{B}_1^T & \tilde{R}_8(\tilde{R}_9 & \tilde{B}_9\tilde{U}_{14}^T)) \\
 & & \tilde{R}_3(\tilde{B}_2^T & \tilde{R}_7(\tilde{B}_1^T & \tilde{R}_8(\tilde{R}_9 & \tilde{B}_9\tilde{U}_{14}^T))) \\
 & \tilde{R}_4(\tilde{B}_3^T & \tilde{R}_6(\tilde{B}_2^T & \tilde{R}_7(\tilde{B}_1^T & \tilde{R}_8(\tilde{R}_9 & \tilde{B}_9\tilde{U}_{14}^T)))) \\
\tilde{R}_{c_1}(\tilde{B}_4^T & \tilde{R}_5(\tilde{B}_3^T & \tilde{R}_6(\tilde{B}_2^T & \tilde{R}_7(\tilde{B}_1^T & \tilde{R}_8(\tilde{R}_9 & \tilde{B}_9\tilde{U}_{14}^T)))))
\end{array}\right)$$

$$(4.4)\quad = \left(\begin{array}{rr|l|r}
0 & & & R_9^TR_8^TR_7^TR_6^TR_5^TR_{c_2}^TU_{c_2}^T \\
\hline
 & & R_1R_9 & B_1R_7^TR_6^TR_5^TR_{c_2}^TU_{c_2}^T \\
 & R_2(B_1^T & R_8R_9) & B_2R_6^TR_5^TR_{c_2}^TU_{c_2}^T \\
R_3(B_2^T & R_7(B_1^T & R_8R_9)) & B_3R_5^TR_{c_2}^TU_{c_2}^T \\
R_4(B_3^TR_6(B_2^T & R_7(B_1^T & R_8R_9))) & B_4R_{c_2}^TU_{c_2}^T \\
R_{c_1}(B_4^TR_5(B_3^TR_6(B_2^T & R_7(B_1^T & R_8R_9)))) & B_{c_1}U_{c_2}^T
\end{array}\right),$$

where $\tilde{U}_{14} \equiv \tilde{U}_{10}\tilde{R}_{10}\tilde{R}_{11}\tilde{R}_{12}\tilde{R}_{13}$. Equation (4.4) is partitioned into four nonzero blocks corresponding to the above four types of connection changes. It turns out that we


7
3
6
1
2
4
5
i
p
R_1 R_2 R_3 R_4 R_5 R_6
B_1 B_3 B_4
D_1, U_1 D_2, U_2 D_4, U_4 D_5, U_5
$\tilde R_3$ $\tilde R_6$ $\tilde R_i$ $\tilde R_4$ $\tilde R_p$
$\tilde B_7$ $\tilde B_3$ $\tilde B_i$ $\tilde B_p$
$\tilde D_5, \tilde U_5$ $\tilde D_i, \tilde U_i$ $\tilde D_4, \tilde U_4$

(i) HSS tree example (ii) Inserting a node i (iii) Another way to insert i

FIG. 4.4. *Inserting a node into an HSS tree in two different ways, where dark nodes and edges should be updated or created.*

can construct the generators on the left-hand side of (4.4) in the following sequential way.

First, consider the last row in (4.4). Compute a QR factorization of the right-hand side such that

$$\left(\tilde R_{c_1}(\tilde B_4^T \quad \tilde R_5(\tilde B_3^T \quad \tilde R_6(\tilde B_2^T \quad \tilde R_7(\tilde B_1^T \mid \tilde R_8(\tilde R_9 \mid \tilde B_9 \tilde U_{14}^T))))) \right) = Q_1 T_1.$$

Then partition $T_1 = (T_{1;1} \quad T_{1;2})$ such that $T_{1;1}$ has the same column dimension as B_4^T. Thus, we can let $\tilde R_{c_1} = Q_1,\ \tilde B_4^T = T_{1;1}$, and

$$\left(\tilde R_5(\tilde B_3^T \quad \tilde R_6(\tilde B_2^T \quad \tilde R_7(\tilde B_1^T \mid \tilde R_8(\tilde R_9 \mid \tilde B_9 \tilde U_{14}^T))))\right) = T_{1;2}. \tag{4.5}$$

In this way, one layer (the last row) is removed from (4.4).

Next, combine (4.5) with the fourth row of (4.4):

$$\begin{pmatrix} \tilde R_4 \\ \tilde R_5 \end{pmatrix} \left(\tilde B_3^T \quad \tilde R_6(\tilde B_2^T \quad \tilde R_7(\tilde B_1^T \mid \tilde R_8(\tilde R_9 \mid \tilde B_9 \tilde U_{14}^T)))\right) = \begin{pmatrix} B_4 R_{c_2}^T U_{c_2}^T \\ T_{1;2} \end{pmatrix}.$$

Again, compute a QR factorization $Q_2 T_2$ of the right-hand side. Then partition $Q_2^T = (Q_{2;1}^T \quad Q_{2;2}^T)$ such that $Q_{2;1}$ has the same row dimension as B_4, and partition $T_2 = (T_{2;1} \quad T_{2;2})$ such that $T_{2;1}$ has the same column dimension as B_3^T. We can set $\tilde R_4 = Q_{2;1}$, $\tilde R_5 = Q_{2;2}$, $\tilde B_3^T = T_{2;1}$, and

$$\left(\tilde R_6(\tilde B_2^T \quad \tilde R_7(\tilde B_1^T \mid \tilde R_8(\tilde R_9 \mid \tilde B_9 \tilde U_{14}^T)))\right) = T_{2;2}. \tag{4.6}$$

Now, we can combine (4.6) with the third row of (4.4), and the above procedure repeats.

Finally, it is trivial to merge node c_2 into i. The overall process costs no more than $O(N)$ flops for an order-N matrix H.

4.2.3. Inserting and deleting HSS blocks. Sometimes, we need to insert a block row/column to an HSS matrix or to remove one from it. To remove a block is usually straightforward. For simplicity, in this subsection we consider symmetric HSS matrices. To remove a node i from an HSS tree, we remove any generators associated with the subtree rooted at i and merge the sibling node j of i into its parent p by setting $U_p = U_j R_j$ and $D_p = D_j$ if j is a leaf node.

To insert a block row/column into an HSS matrix, the result depends on the desired HSS structure. For example, suppose Figure 4.4(i) is the original HSS tree for an HSS matrix, and we insert a new node i between node 2 and 4 to get a new matrix with a tree structure as in Figure 4.4(ii) or Figure 4.4(iii). We need only to update a few generators (shown in dark in Figure 4.4). Again, we can consider the connection changes between nodes and use QR factorizations to find the new generators. The details are similar to those in the previous subsection. Note that in Figure 4.4, the HSS tree can be more general and that the node i to be inserted can also represent another HSS tree. In all cases, we need only to update few generators to get the new matrix. Thus the overall process is fast.

4.3. HSS structured extend-add. In this subsection, we use the previous basic HSS operations to build the structured extend-add process, as outlined in subsection 3.2. We consider a general element and its two child elements, as shown in Figure 2.3(ii) or 2.4, where i is the pivot separator in the assembly tree. The frontal and update matrices in the extend-add $\mathcal{F}_i = \mathcal{F}_i^0 \updownarrow\!\!\!\leftrightarrow \mathcal{U}_{c_1} \updownarrow\!\!\!\leftrightarrow \mathcal{U}_{c_2} \equiv \mathcal{F}_i^0 + \hat{\mathcal{U}}_{c_1} + \hat{\mathcal{U}}_{c_2}$ have the relationship as shown in Figure 3.2, where all matrices should now be in HSS forms.

The general HSS extend-add procedure is as follows. Assume that, before the extend-add, the frontal matrices $\mathcal{F}_{c_1}$ and $\mathcal{F}_{c_2}$ for the left and right child elements, respectively, are already in HSS forms. (These HSS forms come recursively from lower level separators or from simple constructions at the starting level of structured factorizations.) For simplicity, assume each separator is represented by one leaf of the HSS tree, although each leaf can be potentially a subtree. There are then five leaf nodes in each HSS tree. As we use full binary HSS trees, it is natural to use trees as shown in the first row of Figure 4.5. A tree like that has the minimum depth among all full binary trees. Note that the separators are ordered with the uniform ordering.

Following the uniform ordering of the separator pieces i, p_1, p_2, p_3, p_4, the HSS tree of $\mathcal{F}_i$ has the form in Figure 4.6. Therefore, we should transform the tree structures in the first row of Figure 4.5 to the structure in Figure 4.6 (also see Table 3.4).

Figure 4.5 shows the process of generating $\hat{\mathcal{U}}_{c_1}$ and $\hat{\mathcal{U}}_{c_2}$ from $\mathcal{F}_{c_1}$ and $\mathcal{F}_{c_2}$, respectively. The HSS trees of $\mathcal{F}_{c_1}$ and $\mathcal{F}_{c_2}$ are shown in the first row of Figure 4.5, with their leaf nodes marked by the separators in Figure 2.4. Typically, there are five steps as follows for an extend-add operation to advance from the level of c_1 and c_2 to the level of i (for convenience, we also include the partial factorization of frontal matrices at the beginning):

0. Eliminate c_1 and c_2 and get update matrices $\mathcal{U}_{c_1}$ and $\mathcal{U}_{c_2}$ in HSS forms.
1. Permute the trees for $\mathcal{U}_{c_1}$ and $\mathcal{U}_{c_2}$, as in the third column of Table 3.4.
2. Insert appropriate zero nodes to the HSS trees of $\mathcal{U}_{c_1}$ and $\mathcal{U}_{c_2}$ to get $\hat{\mathcal{U}}_{c_1}$ and $\hat{\mathcal{U}}_{c_2}$, respectively; see the fourth column of Table 3.4.
3. Split HSS blocks in $\hat{\mathcal{U}}_{c_1}$ and $\hat{\mathcal{U}}_{c_2}$ to handle overlaps in separators.
4. Write the initial frontal matrix $\mathcal{F}_i^0$ in HSS form based on the tree structure of $\mathcal{U}_{c_1} \updownarrow\!\!\!\leftrightarrow \mathcal{U}_{c_2} \equiv \hat{\mathcal{U}}_{c_1} + \hat{\mathcal{U}}_{c_2}$.
5. Get $\mathcal{F}_i$ by adding HSS matrices $\mathcal{F}_i^0, \hat{\mathcal{U}}_{c_1}, \hat{\mathcal{U}}_{c_2}$. Compress $\mathcal{F}_i$ when necessary.

Step 0 can be done by applying the generalized HSS Cholesky factorization in subsection 4.1 to the leading principal blocks of $\mathcal{F}_{c_j}$ corresponding to separator c_j for $j = 1, 2$ (first row in Figure 4.5). When separator c_j is removed, the Schur complement/update matrix $\mathcal{U}_{c_j}$ is obtained by updating the rest tree nodes.

Step 1 is to permute some branches of the HSS tree of $\mathcal{U}_{c_j}$. Note that even if the HSS tree has more levels, we still just need to update few top level nodes because we need only to permute the four separator pieces. This means the cost of permutations in the superfast multifrontal method is $O(1)$, even if the update matrix has dimension N. Specifically, to permute $\mathcal{U}_{c_1}$ (Figure 4.5, left column, from row 1 to row 2), we exchange p_2 and i, as shown in the third column of Table 3.4. We can set new

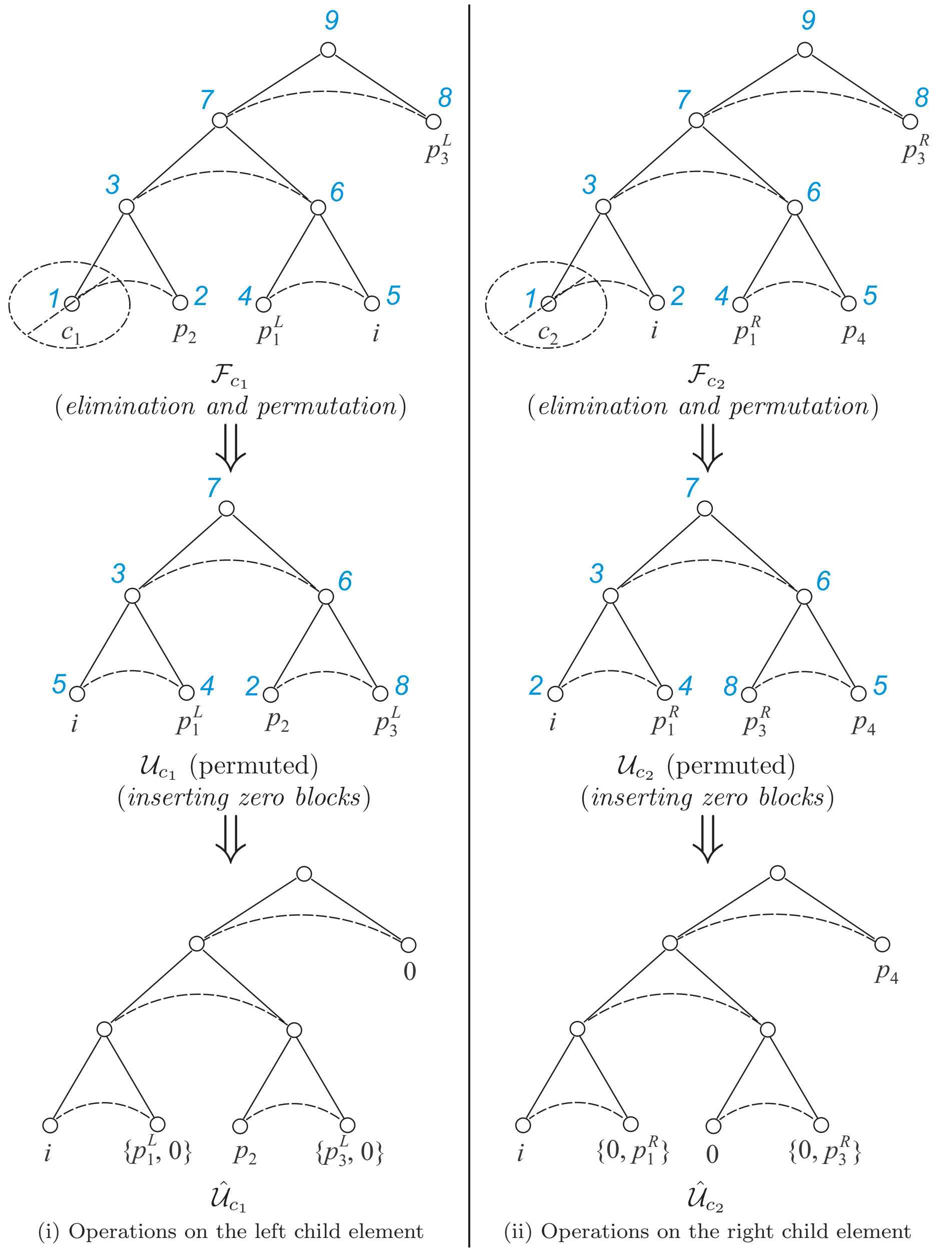

9
7
8
p_3^L
3
6
1
2
4
5
c_1
p_2
p_1^L
i
$\mathcal{F}_{c_1}$
(*elimination and permutation*)
7
3
6
5
4
2
8
i
p_1^L
p_2
p_3^L
$\mathcal{U}_{c_1}$ (permuted)
(*inserting zero blocks*)
0
i
$\{p_1^L, 0\}$
p_2
$\{p_3^L, 0\}$
$\hat{\mathcal{U}}_{c_1}$
(i) Operations on the left child element
9
7
8
p_3^R
3
6
1
2
4
5
c_2
i
p_1^R
p_4
$\mathcal{F}_{c_2}$
(*elimination and permutation*)
7
3
6
2
4
8
5
i
p_1^R
p_3^R
p_4
$\mathcal{U}_{c_2}$ (permuted)
(*inserting zero blocks*)
p_4
i
$\{0, p_1^R\}$
0
$\{0, p_3^R\}$
$\hat{\mathcal{U}}_{c_2}$
(ii) Operations on the right child element

FIG. 4.5. *Operations on child elements in the structured extend-add process* (3.2).

generators (in tilded notation) of the permuted tree to be

$$\tilde{B}_5 = B_4^T, \quad \tilde{R}_8 = B_7^T R_6^T, \quad \tilde{B}_2 = R_2 R_3 B_7, \quad \tilde{R}_2 = R_2 B_3, \quad \tilde{B}_3 = I.$$

Similarly, to permute $\mathcal{U}_{c_2}$ (Figure 4.5, right column, from row 1 to row 2), we exchange p_4 and p_3^R. The new generators of the permuted tree should satisfy

$$\tilde{B}_2 = R_2 R_3 R_4^T, \qquad \tilde{B}_8 = B_7^T R_6^T R_5^T,$$
$$\begin{pmatrix} \tilde{R}_2 \\ \tilde{R}_4 \end{pmatrix} \tilde{B}_3 \begin{pmatrix} \tilde{R}_8^T & \tilde{R}_5^T \end{pmatrix} = \begin{pmatrix} R_2 R_3 B_7 & R_2 R_3 R_5^T \\ R_4 R_6 B_7 & B_4 \end{pmatrix}.$$

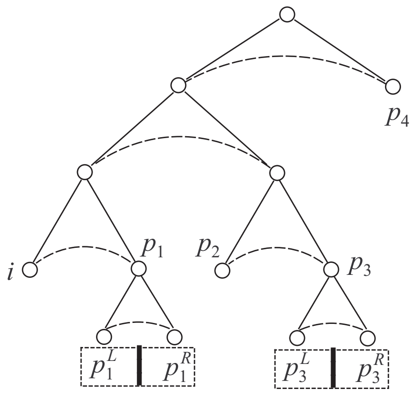


FIG. 4.6. *HSS tree structure of $\mathcal{F}_i$, where the dark bars represent overlaps $p_1^L \cap p_1^R$ and $p_3^L \cap p_3^R$, respectively.*

An SVD of the right-hand side of the last equation can provide all the generators on the left-hand side. Although these formulas look specific, they are sufficient for general extend-add in the superfast multifrontal method.

At step 2, we insert zero blocks into the permuted update matrices to get $\hat{\mathcal{U}}_{c_1}$ and $\hat{\mathcal{U}}_{c_2}$. For the left child element, a zero block (node) is attached to the matrix (tree); see Figure 4.5, left column, row 3. This operation is trivial in that only certain zero generators should be added. For the right child element, a zero node is inserted between p_1^R and p_3^R. This has already been discussed in subsection 4.2.3.

After we get the HSS trees as shown in the last row of Figure 4.5, we use step 3 to handle the overlaps so that $\hat{\mathcal{U}}_{c_1}$ and $\hat{\mathcal{U}}_{c_2}$ will have the same HSS tree structure and row/column indices. As discussed in section 3.2, overlaps occur in separators p_1 and p_3; see Figures 2.3(ii) and 2.4. As an example, the overlap $p_1^L \cap p_1^R$ may correspond to HSS blocks in both $\hat{\mathcal{U}}_{c_1}$ and $\hat{\mathcal{U}}_{c_2}$ or, more likely, parts of their HSS blocks. For the latter case, in order to match the HSS structures of $p_1^L \cap p_1^R$ in $\hat{\mathcal{U}}_{c_1}$ and $\hat{\mathcal{U}}_{c_2}$, we need to cut $p_1^L \cap p_1^R$ from either p_1^L, on the right end, or from p_1^R, on the left end, and to merge it with a nearby zero block. We use the splitting procedure in subsection 4.2.2. Now, $\hat{\mathcal{U}}_{c_1}$ and $\hat{\mathcal{U}}_{c_2}$ have the same HSS tree structure, which becomes the structure of $\hat{\mathcal{U}}_{c_1} + \hat{\mathcal{U}}_{c_2}$ and also $\mathcal{F}_i$ (Figure 4.6).

Then at step 4, we convert the initial frontal matrix $\mathcal{F}_i^0$ into an HSS form following the HSS structure of $\hat{\mathcal{U}}_{c_1} + \hat{\mathcal{U}}_{c_2}$. Because the original matrix A is sparse, $\mathcal{F}_i^0$ is generally sparse also. We are often able to write the HSS form of $\mathcal{F}_i^0$ in advance. For example, for Model Problem 1.1 with nested dissection ordering, $\mathcal{F}_i^0$ in (2.1) has the sparsity pattern where A_{ii} is tridiagonal, $A_{ip_2} = 0, A_{ip_4} = 0$, and each of A_{ip_1} and A_{ip_3} has only one nonzero entry. Such a matrix has HSS rank 2.

Now at step 5, we are ready to get $\mathcal{F}_i$ by computing the HSS sum of $\mathcal{F}_i^0$, $\hat{\mathcal{U}}_{c_1}$, and $\hat{\mathcal{U}}_{c_2}$ with formulas in [11]. The sizes of the generators of $\mathcal{F}_i$ increase after this addition, although the actual HSS rank of $\mathcal{F}_i$ will not. Thus usually the HSS addition is followed by a compression step [11]. Now $\mathcal{F}_i$ is in compact HSS form, and we can continue the factorization along the elimination tree.

4.4. Algorithm and performance. Based on the previous discussions, we present the main superfast multifrontal algorithm and its analysis. Before that, we first clarify a few implementation issues for the HSS operations.

4.4.1. Implementation issues. One issue is related to the mesh boundary. For convenience, we can assume the mesh boundary corresponds to empty separators. We may then have empty nodes in HSS trees. Empty nodes do not accumulate or change and are not associated with any actual operation.

Another issue is to predetermine the HSS structures before the actual factorizations. Similar to other sparse direct solvers, we can have a symbolic factorization stage which is used after nested dissection to approximately predict the HSS structures of the frontal/update matrices in the elimination.

Finally, for the purpose of computational performance, we usually avoid too large or too small HSS block sizes.

4.4.2. Factorization algorithm. We provide the superfast multifrontal method and analyze its performance.

ALGORITHM 4.1 (SUPERFAST MULTIFRONTAL METHOD WITH HSS STRUCTURES).

```
1. Use nested dissection to order the nodes in the $n \times n$ mesh. Build an elimination tree with separator ordering. Assume the total number of separators to be $k$ and the total number of levels to be $l = \lfloor \log_2 n \rfloor$.
2. Decide $l_0$, the number of bottom levels of traditional factorizations (see Theorem 4.2 below).
3. For separators $i = 1, \ldots, k$
   (a) If separator $i$ is at level $l_i > l - l_0$, do traditional Cholesky factorization and extend-add.
       i. If $i$ is a leaf node in the elimination tree, obtain the frontal matrix $\mathcal{F}_i$ from $A$ and compute $\mathcal{U}_i$ as in (2.1). Push $\mathcal{U}_i$ onto an update matrix stack.
       ii. Otherwise, pop two update matrices $\mathcal{U}_{c_1}$ and $\mathcal{U}_{c_2}$ from the update matrix stack, where $c_1$ and $c_2$ are the children of $i$. Use extend-add to form the frontal matrix $\mathcal{F}_i$ as in (2.2). Factorize $\mathcal{F}_i$ and get $\mathcal{U}_i$ as in (2.2). Push $\mathcal{U}_i$ onto the update matrix stack.
   (b) If separator $i$ is at the switching level $l_i = l - l_0$,
       i. Following step 3(a), build $\mathcal{F}_i$, factorize its pivot block, and get $\mathcal{U}_i$ .
       ii. Construct a simple HSS form for $\mathcal{U}_i$ with few blocks (1, 2, or 4, etc.). Push the HSS form of $\mathcal{U}_i$ onto the update matrix stack.
   (c) Otherwise (separator $i$ is at level $l_i < l - l_0$), do structured factorization and extend-add at upper levels.
       i. Pop two HSS matrices $\mathcal{U}_{c_1}$ and $\mathcal{U}_{c_2}$ from the stack. Use HSS extend-add to form the frontal matrix $\mathcal{F}_i$, as in Figure 3.2.
       ii. Compute the generalized HSS Cholesky factorization of the leading principal blocks of $\mathcal{F}_i$ and compute the Schur complement which is $\mathcal{U}_i$ (in HSS form). Push the HSS form of $\mathcal{U}_i$ onto the stack.
```

About the complexity and storage requirement of the algorithm, we have the following theorem.

THEOREM 4.2. *Assume p is the maximum of all HSS ranks of the frontal and update matrices throughout the multifrontal method. Then the optimal complexity of Algorithm* 4.1 *is $O(pn^2)$. In this situation, the number of bottom levels of traditional Cholesky factorizations is $l_0 = O(\log_2 p)$, the bottom level traditional Cholesky factorizations and upper level structured factorizations take the same amount of work, the storage required for the factors is $O(n^2 \log_2 p)$, and the update matrix stack size is $O(pn)$.*

TABLE 4.1

	Traditional factorization	Structured factorization
$l = O(\log(n))$ levels	l_0 bottom levels	$l - l_0$ upper levels
Each level ($i = 1, \ldots, l$)	4^{i-1} subproblems, each of dim $O(2^{l-i})$	
Cost (subtotal)	$\sum_{i=l-l_0+1}^{l} 4^{i-1}(O(2^{l-i}))^3$	$\sum_{i=1}^{l-l_0} 4^{i-1} O(p^2 2^{l-i})$
Cost (total)	$O(4^l 2^{l_0}) + O\left(4^l \frac{p^2}{2^{l_0}}\right)$	

Proof. Consider the costs before and after the switching level (see Table 4.1).

Choose l_0 such that $2^{l_0} \approx p$. Then the total cost is $O(4^l p) = O(pn^2)$. In this case, the costs before and after the switching level are approximately the same. Similarly, the total storage required for the HSS factors is

$$\sum_{i=l-l_0+1}^{l} 4^{k-1} \left(O\left(2^{l-i}\right)\right)^2 + \sum_{i=1}^{l-l_0} 4^{i-1} O\left(p2^{l-i}\right) = O\left(l_0 4^l\right) + O\left(\frac{p}{2^{l_0}} 4^l\right) = O(n^2 \log_2 p).$$

The maximum update matrix stack size is

$$\sum_{i=l-l_0+1}^{l} \left(O\left(2^{l-i}\right)\right)^2 + \sum_{i=1}^{l-l_0} O\left(p2^{l-i}\right) = O\left(4^{l_0-1}\right) + O\left(p2^l\right) = O(pn).$$

Similar to the traditional multifrontal method, a frontal matrix $\mathcal{F}_i$ can use the stack space of $\mathcal{U}_{c_1}$ and $\mathcal{U}_{c_2}$, and $O(pn)$ extra space may be needed. ☐

4.5. System solution. The system solution using generalized HSS Cholesky factors have overall structures similar to the standard multifrontal method, except in two major places. One is that for each node of the elimination tree, the standard triangular solver is replaced by an HSS solver [11], and another is that the forward and backward substitutions are now done via generators and solution vectors associated with the HSS tree nodes and elimination tree nodes (separators). For example, we want to solve the following system:

$$\begin{pmatrix} L_{ii} & 0 \\ L_{iB} & L_{BB} \end{pmatrix} \begin{pmatrix} x_i \\ x_B \end{pmatrix} = \begin{pmatrix} b_i \\ b_B \end{pmatrix},$$

where the coefficient matrix is a lower triangular HSS matrix and x_i and x_B are associated with separator i and its neighbors, respectively. First, we solve an HSS system $L_{ii}x_i = b_i$ to get x_i. Then we compute $\hat{b}_B = b_B - L_{iB}x_i$. Here $L_{iB}x_i$ is usually computed via the products of certain generators (associated with the neighbors of i) and x_i. This solution process is highly efficient using HSS representations.

THEOREM 4.3. *With the same conditions as in Theorem* 4.2, *the cost to solve the linear system with the HSS factors is* $O(n^2 \log p)$.

Notice that the standard multifrontal method solution process costs $O(n^2 \log n)$ with the Cholesky factor. Stability results for each individual component of the

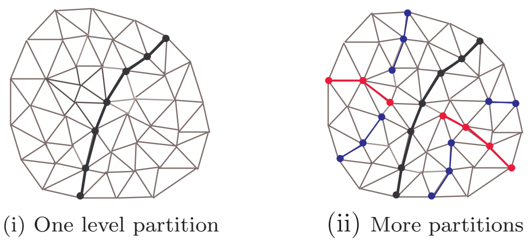


FIG. 4.7. *Nested dissection ordering of a general mesh.*

superfast multifrontal method can be possibly obtained similar to [14]. However, it can be quite complicated to analyze the entire algorithm. Thus instead, we use numerical experiments to show that the method is stable in practice.

4.6. Generalizations. We have used Model Problem 1.1 in some discussions of the algorithm. In fact, the method can be used to solve various other problems discretized on general meshes. The previous discussions do not specifically depend on the matrix pattern or entries in (1.2). Once the low-rank property is verified for the problem, the new method can be applied similarly. In general, the mesh just needs to be well shaped [37, 38, 42]. The low-rank property usually arises in some PDEs with Green's functions smooth away from the diagonal singularity and even more general problems; see [1, 2, 8, 24, 25, 45].

In nested dissection, graph partitioning algorithms or tools such as `Metis` [36] applied to mesh pieces can be used to generate the separators. The graph partition can also be done based on the sparsity pattern of A without a specific grid. Although it is possible to find a uniform ordering similar to Definition 3.1, more straightforward orderings can be used for simplicity. (It is an open problem to find nearly optimal ordering of the separator pieces and mesh points within an element.) Once the separators are obtained, the multifrontal method apply similar to before.

During the levels of structured eliminations of separators, the structured factorization algorithm is the same as before, with the frontal and update matrices in HSS forms. The structured extend-add operation can be done similar to before, although the connectivity between separator pieces can be more complicated, and the number of neighbors of a separator can be arbitrary; see Figure 4.7 for an example.

The major tasks in the extend-add process are still to handle overlaps between separator pieces and to align indices. The HSS permuting/splitting/merging/deleting/inserting operations summarized in subsections 4.2 and 4.3 can be similarly used.

Although our current preliminary implementation of the algorithm is based on 2D rectangular regular meshes, we expect to develop codes for more general meshes and more general sparsity patterns of A. Also, we expect to find more practical problems with the low-rank property.

5. Numerical experiments.

We show some numerical results for the model problem and a linear elasticity problem.

5.1. Example 1: The model problem. Here, we factorize A in (1.2) of Model Problem 1.1 with our superfast multifrontal method implemented in Fortran 90. We run the code on a 2.33 GHz Intel E5410 server. The superfast multifrontal method is compared with other methods in various aspects (computation time, flop counts, memory usage, errors, etc.) and with different parameters (mesh dimension, tolerance, number of elimination levels, switching level, etc.). The following notation is used below.

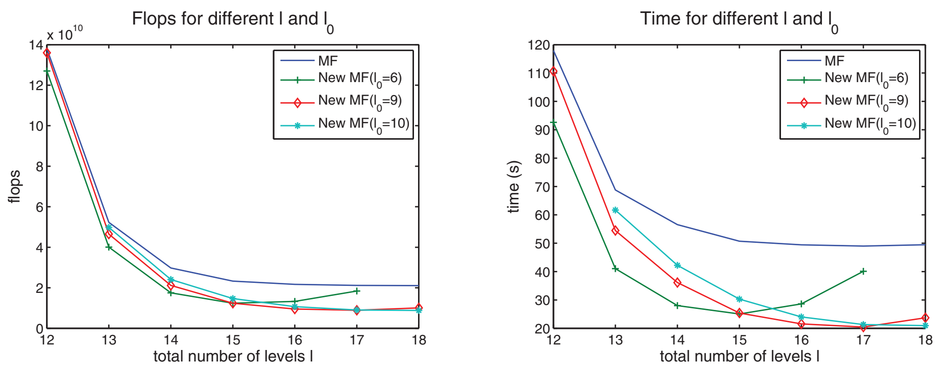


FIG. 5.1. *Numerical results of* MF *and* New MF *for solving Model Problem* 1.1 *with different number of elimination levels l, where $n = 1023$ and $\tau = 10^{-6}$.*

Notation	**Meaning**
`MF`	traditional multifrontal method
`New MF`	superfast multifrontal method with HSS structures
τ	relative tolerance
l	total number of elimination levels in nested dissection
l_0	number of bottom levels ($l - l_0$: switch level)
time (s)	time in seconds
x	numerical solution
x^*	exact solution

5.1.1. Choices of parameters. For simplicity, we consider Model Problem 1.1 on square regular meshes with mesh dimensions $n = 2^k - 1$. Before giving comprehensive comparisons, we first run the tests in various aspects in terms of the parameters.

(1) *Total number of elimination levels l.* The maximum possible number of levels in the elimination tree is $l = 2k - 1$. Theoretically, we should use as many levels as possible. But practically, a large l leads to small bottom level matrices, and the computations get slower. We test different choices of l in `MF` and `New MF`; see Figure 5.1. The optimal computation time generally occurs for certain l smaller than $2k - 1$.

(2) *Optimal number of bottom levels l_0.* For `New MF` with a fixed l, different numbers of bottom levels l_0 lead to highly different performance. According to Theorem 4.2 and its proof, the optimal complexity occurs when the costs before the switching level (bottom level standard Cholesky factorization cost) and after the switching level (HSS structured factorization cost) are approximately the same. For the model problem, we fix l and record the flop counts and computation time for different l_0 in Figure 5.2. The left plot in Figure 5.2 is also marked with the percentage of the cost before the switching level over the total cost. We can see that the approximately optimal flop count is achieved when this percentage is closest to 50%.

(3) *Tolerance τ.* Different tolerances τ can be used, depending on the accuracy requirements. The cost decreases when τ increases. The decreasing rate varies for different l_0; see Figure 5.3 for an example. These results indicate that with different purposes, we may use different tolerances, and the corresponding l_0 should also vary. This gives the flexibility to control the accuracy of direct solutions of linear systems. When only modest accuracy is desired, the method is very attractive. High accuracy and good efficiency can also be achieved when the method is combined with iterative refinement (subsection 5.1.4) or other iterative schemes.

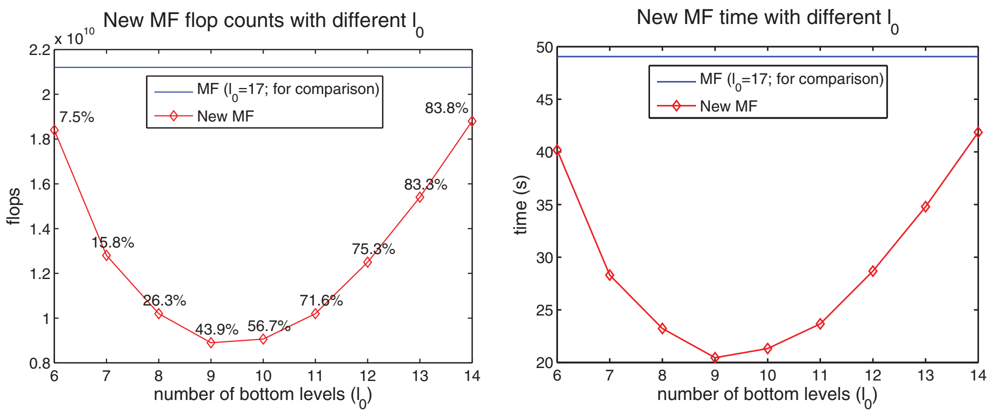


FIG. 5.2. *Numerical results of* `New MF` *for Model Problem* 1.1 *with different* l_0*, where* $n = 1023$*,* $l = 17$*,* $\tau = 10^{-6}$*, and in the left plot, each percentage means the fraction of the bottom level traditional factorization cost over the total cost.*

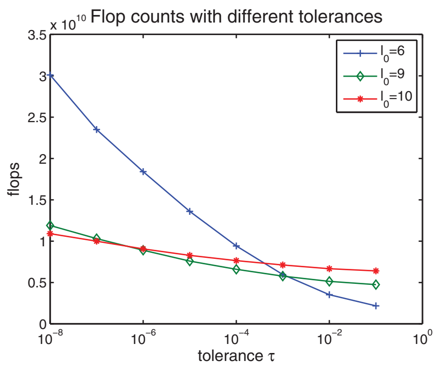


FIG. 5.3. *Tolerance vs. flop count for* $n = 1023$ *with different number of bottom levels* l_0*.*

5.1.2. Factorization complexity and timing comparison. Now we give a comparison of `New MF`, `MF`, and `SuperLU` [15]. For mesh dimensions n ranging from 127 to 4095, Table 5.1 lists the factorization flop counts with different l_0, and Table 5.2 lists the timing. Note that `MF` has complexity $O(n^3)$, which is the current optimal count in exact arithmetic (ignoring special techniques such as Strassen's algorithm). `New MF` has complexity $O(pn^2)$ and starts to outperform `MF` and `SuperLU` for $n = 255$.

Remark 5.1. The break-even size n can be smaller if a larger l is used. However, as discussed above, a too large l is not preferred in practice. Here, `New MF` demonstrates the potential of structured supernodal multifrontal factorizations, although it can be further improved.

To check the complexity, we calculate the *flop scaling factor* $\text{flops}_{2n}/\text{flops}_n$, that is, the ratio of the flop count with mesh dimension $2n$ over the flop count with mesh dimension n, and also the *time scaling factor* $\text{time}_{2n}/\text{time}_n$ defined similarly. Figure 5.4 shows the numerical results with the scaling factors marked. The experimental scaling factors are consistent with the theoretical complexity. When n doubles, the matrix size quadruples. The scaling factors of `New MF` approach 4 when n is large.

TABLE 5.1
Flop counts of `MF`, `SuperLU`, and `New MF` with $\tau = 10^{-6}$, where the count $\frac{829}{42}n^3$ is the leading term of the theoretical count of `MF` [20].

Mesh dimension n			127	255	511	1023	2047	4095
l			11	13	15	17	19	21
`New MF`	l_0	6	$1.20E8$	$7.69E8$	$3.98E9$	$1.84E10$	$7.95E10$	$3.32E11$
		7	$7.12E7$	$4.96E8$	$2.70E9$	$1.28E10$	$5.64E10$	$2.38E11$
		8	$5.31E7$	$3.77E8$	$2.11E9$	$1.02E10$	$4.53E10$	$1.92E11$
		9	**4.56E7**	**3.25E8**	**1.83E9**	**8.90E9**	**3.97E10**	**1.69E11**
		10	/	$3.29E8$	$1.87E9$	$9.06E9$	$4.03E10$	$1.71E11$
		11	/	$3.42E8$	$2.08E9$	$1.02E10$	$4.57E10$	$1.93E11$
		12	/	/	$2.42E9$	$1.25E10$	$5.68E10$	$2.32E11$
		13	/	/	$2.67E9$	$1.54E10$	$7.27E10$	$3.14E11$
`MF`			$4.11E7$	$3.30E8$	$2.64E9$	$2.12E10$	$1.67E11$	$1.33E12$
$\frac{829}{42}n^3$			$4.04E7$	$3.27E8$	$2.63E9$	$2.11E10$	$1.69E11$	$1.36E12$
`SuperLU`			$4.05E7$	$3.95E8$	$3.89E9$	$3.19E10$	$2.77E11$	$2.39E12$

TABLE 5.2
Computation time of `MF`, `SuperLU`, and `New MF` with $\tau = 10^{-6}$.

Mesh dimension n			127	255	511	1023	2047	4095
l			11	13	15	17	19	21
`New MF`	l_0	6	0.273	1.703	8.707	40.18	174.0	726.7
		7	0.168	1.113	6.011	28.30	124.1	524.0
		8	0.133	0.875	4.828	23.23	103.4	438.0
		9	**0.113**	**0.758**	**4.227**	**20.46**	**91.54**	**388.9**
		10	/	0.781	4.395	21.32	94.95	403.7
		11	/	0.813	4.805	23.67	106.2	452.0
		12	/	/	5.543	28.68	130.5	559.7
		13	/	/	6.125	34.80	163.9	712.6
`MF`			0.105	0.789	6.188	49.05	405.3	3321.3
`SuperLU`			0.110	0.940	7.950	60.37	508.7	4364.2

This demonstrates that `New MF` has nearly linear complexity. On the other hand, the scaling factors of `MF` and `SuperLU` are about 8 or larger. In addition, `New MF` is also significantly faster than iterative methods such as the straightforward CG.

5.1.3. Storage usage. According to Theorem 4.2, the superfast multifrontal method is also memory efficient. Figure 5.5 shows the storage requirements for the stacks and the factors. Notice that the traditional multifrontal method needs an $O(n^2)$ size stack, while the HSS multifrontal method requires only $O(pn)$. When we use a larger τ in `NEW MF`, the storage is reduced accordingly.

5.1.4. System solution, accuracy, and iterative refinement. It is also efficient to solve the linear systems with the structured Cholesky factors obtained from `New MF`. The cost of system solution is usually insignificant as compared with the factorization cost and is not reported here. The errors in the `New MF` solutions are listed in Table 5.3. The results indicate that the method is stable in practice.

Since the solution process is very efficient, we can use relatively large tolerances in the factorization and then use iterative refinements to improve the solution. Here for different tolerances we did a simple test; see Figure 5.6 for the error results after selected numbers of iterative refinement steps. For a modest tolerance, high accuracy can be achieved with very few steps of iterative refinement.

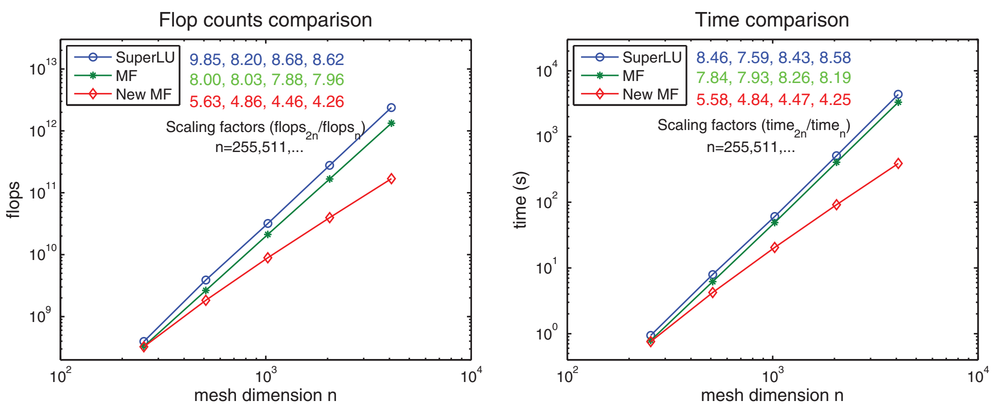


FIG. 5.4. *Comparison of* MF *and* New MF *with* $\tau = 10^{-6}$ *and* $l_0 = 9$*, where the scaling factors* $flops_{2n}/flops_n$ *and* $time_{2n}/time_n$ *are shown. Note that when* n *doubles, the matrix size* n^2 *quadruples.*

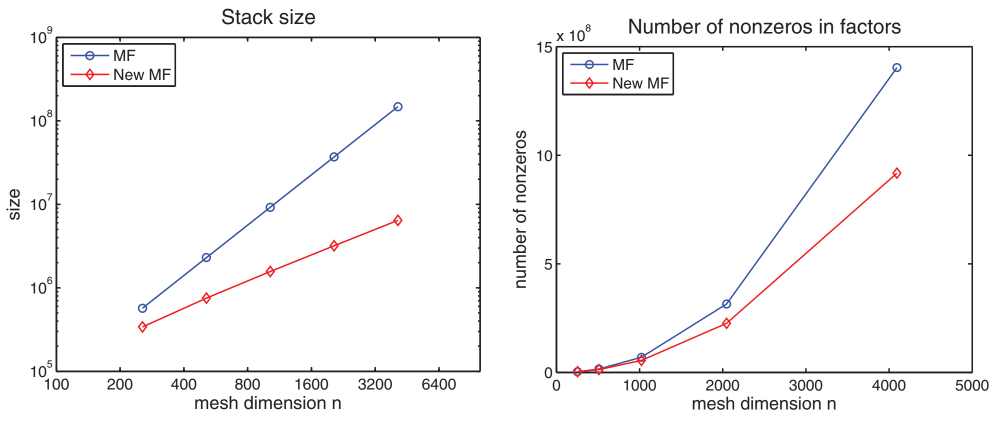


FIG. 5.5. *Storage comparison of* MF *and* New MF*.*

TABLE 5.3
Errors in system solution with the structured factors from New MF*, where* $\tau = 10^{-6}$*,* $l_0 = 9$*, and each* b *is obtained from* Ax^* *with a random* x^**.*

n	255	511	1023	2047	4095
$\frac{\|\|x-x^*\|\|_\infty}{\|\|x^*\|\|_\infty}$	$4.58E-6$	$2.01E-5$	$3.85E-5$	$8.45E-5$	$3.60E-4$
$\frac{\|\|Ax-b\|\|_\infty}{\|\|\|A\|\|x\|+\|b\|\|\|_\infty}$	$7.58E-8$	$9.84E-8$	$1.21E-7$	$1.59E-7$	$2.41E-7$
$\frac{\|\|x-x^*\|\|_2}{\|\|x^*\|\|_2}$	$2.31E-6$	$1.25E-5$	$2.21E-5$	$5.47E-5$	$2.58E-4$
$\frac{\|\|Ax-b\|\|_2}{\|\|\|A\|\|x\|+\|b\|\|\|_2}$	$4.73E-9$	$5.83E-9$	$6.31E-9$	$6.88E-9$	$6.95E-9$

Also note that when large tolerances are used, our superfast multifrontal method has great potential to work as an effective preconditioner. As a preliminary test, for a 1023×1023 exact frontal matrix $\mathcal{F}$ with 2-norm condition number 845.5, we approximate it by an HSS matrix $\tilde{\mathcal{F}}$ with bottom level HSS block sizes about 32 and

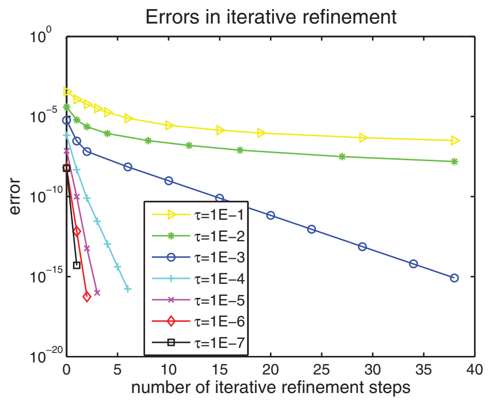


FIG. 5.6. *Errors after some steps of iterative refinement with different tolerances τ for $n = 1023$ and $l_0 = 9$.*

TABLE 5.4
Flops counts for solving (5.1) *with $\lambda/\mu = 10^5$, where each flop ratio is calculated when the matrix order nearly quadruples and* `New MF` *uses $\tau = 10^{-6}$ and $l_0 = 6$.*

Matrix order	31,250	124,002	498,002	1,996,002	7,992,002	31,984,002
l	10	12	14	16	18	20
`MF` flops	$2.44E9$	$2.06E10$	$1.70E11$	$1.38E12$	$1.12E13$	$8.99E13$
`New MF` flops	$2.23E9$	$1.13E10$	$5.68E10$	$2.75E11$	$1.23E12$	$5.24E12$
`MF` flop ratio	8.44	8.25	8.12	8.12	8.03	/
`New MF` flop ratio	5.07	5.03	4.84	4.45	4.26	/

$\tau = 0.2$. $\tilde{\mathcal{F}}$ is used to precondition $\mathcal{F}$. The preconditioned matrix has condition number 17.6.

5.2. Example 2: A linear elasticity problem. As another example, we consider solving a linear elasticity equation

$$-(\mu\overrightarrow{\Delta u} + (\lambda+\mu)\nabla\overrightarrow{\nabla\cdot u}) = \overrightarrow{f} \text{ in } \Omega, \qquad (5.1)$$
$$\overrightarrow{u} = \overrightarrow{0} \text{ on } \partial\Omega,$$

where Ω is a rectangular domain, $\overrightarrow{u}$ is the displacement vector field, and λ and μ are the Lamé constants. The problem is discretized on a rectangular grid. This PDE is very ill conditioned when λ/μ is large. For situations near the incompressible limit, classical iterative methods including multigrid may diverge or converge very slowly when no effective preconditioner is used.

Here, we use our superfast multifrontal method to factorize the discretized matrix A. Performance and accuracy results similar to the model problem are observed. For example, the dense intermediate frontal and update matrices also have the off-diagonal low-rank property. With modest accuracy in the factorization, the flop counts are consistent with the nearly linear complexity, as shown in Table 5.4.

In system solutions, accuracy similar to Table 5.3 is observed. To get higher accuracy, we can use iterative refinements or the precoditioned conjugate CG (PCG) with `NEW MF` as the preconditioner. For example, for a system where the A matrix has order 124,002, a relative residual of $2.2E-14$ is reached after 44 steps of PCG iterations. As a simple comparison, CG without preconditioning needs over $2.3E5$ iterations to reach the same accuracy. The total cost (flop count) of CG without preconditioning is about 120 times of the PCG cost. Even with the initial factorization cost included for generating the preconditioner, PCG is still about 30 times faster than CG. The matrix A has 2-norm condition number $6.8E9$.

In future work, more comprehensive comparisons with other methods such as multigrid and other preconditioning techniques are expected to be done.

6. General remarks. In this paper, we have presented our superfast multifrontal method with HSS structures. Sparse matrix techniques (nested dissection, etc.) are integrated into the multifrontal method. Low-rank properties in the multifrontal method for solving discretized problems are exploited. Some fast HSS matrix operations are developed based on tree techniques. Both the multifrontal method and HSS structures have some nice features. They take good advantage of dense matrix computations (frontal/update matrices in the multifrontal method vs. generators in HSS forms). They both have efficient storage, nice data locality, and natural adaptability to parallel computations. The multifrontal method reorganizes the factorization of a large sparse matrix into a series of small dense matrix factorizations, and then the HSS representations approximate the dense matrices by a sequence of compact generators. There is a good potential for the method to apply to different problems. It remains open to discover the low-rank property in more applications. Some potential examples include seismic-imaging problems, flow problems in porous and fractured media, integral equations of scattering theory, time harmonic Maxwell equations, etc.

A major difference between our method and existing ones is in that we fully integrate sparse matrix techniques with dense structured matrix operations by using two layers of trees: an elimination tree for the supernodal multifrontal factorization and an HSS tree for each node (separator) of the elimination tree corresponding to dense intermediate frontal and update matrices. The multifrontal scheme brings good data locality, since separators only directly communicate with their parents, instead of all connected neighbors. The use of HSS matrices further makes dense intermediate matrices data sparse and enables convenient conversion of complicated algorithms such as extend-add into structured ones. Our new solver can also work as an effective preconditioner by using large tolerances in the semiseparable approximations.

A thorough analysis can be done on how the rank p, the accuracy, the cost, and the storage requirement are related. Detailed flop and storage counts can be conducted. We refer the reader to the details of counting the cost of nested dissection in [20] and the costs of some HSS operations in [11, 13, 14].

Finally, we point out that the current algorithm is based on 2D problems with known low-rank property. The present implementation is also preliminary. Improvement of the algorithm, parallelization, and extension to complicated meshes and three-dimensional (3D) problems are expected to appear in future work. For 3D problems, nested dissection can be possibly performed by algorithms such as the geometric separator algorithm [21, 37, 38]. We also expect to develop a black-box package which can use A as the input and does not necessarily depend on the mesh.

Acknowledgments. Many thanks to the anonymous referees for their valuable comments. The authors are also very grateful to James Demmel and Panayot Vassilevski for their help in the development of the solver.

REFERENCES

[1] M. Bebendorf, *Efficient inversion of Galerkin matrices of general second-order elliptic differential operators with nonsmooth coefficients*, Math. Comp., 74 (2005), pp. 1179–1199.

[2] M. Bebendorf and W. Hackbusch, *Existence of $\mathcal{H}$-matrix approximants to the inverse FE-matrix of elliptic operators with L^∞-coefficients*, Numer. Math., 95 (2003), pp. 1–28.

[3] S. Börm, L. Grasedyck, and W. Hackbusch, *Introduction to hierarchical matrices with applications*, Eng. Anal. Bound. Elem., 27 (2003), pp. 405–422.

[4] J. Carrier, L. Greengard, and V. Rokhlin, *A fast adaptive multipole algorithm for particle simulations*, SIAM J. Sci. Statist. Comput., 9 (1988), pp. 669–686.

[5] T. F. Chan, *Rank revealing QR factorizations*, Linear Algebra Appl., 88/89 (1987), pp. 67–82.

[6] T. F. Chan and P. C. Hansen, *Computing truncated singular value decomposition least squares solutions by rank revealing QR-factorizations*, SIAM J. Sci. Statist. Comput., 11 (1990), pp. 519–530.

[7] S. Chandrasekaran, P. Dewilde, M. Gu, T. Pals, and A.-J. van der Veen, *Fast stable solver for sequentially semiseparable linear systems of equations*, in High Performance Computing—HiPC 2002: 9th International Conference, Lecture Notes in Comput. Sci. 2552, Springer-Verlag, Heidelberg, 2002, pp. 545–554.

[8] S. Chandrasekaran, P. Dewilde, and M. Gu, *On the Numerical Rank of the Off-Diagonal Blocks of Schur Complements of Discretized Elliptic PDEs*, preprint, University of California, Santa Barbara, CA, 2007.

[9] S. Chandrasekaran, P. Dewilde, M. Gu, T. Pals, X. Sun, A.-J. van der Veen, and D. White, *Some fast algorithms for sequentially semiseparable representations*, SIAM J. Matrix Anal. Appl., 27 (2005), pp. 341–364.

[10] S. Chandrasekaran and M. Gu, *A fast and stable solver for recursively semiseparable systems of equations*, in Structured Matrices in Mathematics, Computer Science and Engineering, II, Contemp. Math. 281, V. Olshevsky, ed., AMS, Providence, RI, 2001, pp. 39–53.

[11] S. Chandrasekaran, M. Gu, X. S. Li, and J. Xia, *Some Fast Algorithms for Hierarchically Semiseparable Matrices*, Technical report CAM08-24, UCLA, Los Angeles, CA, 2008, http://www.math.purdue.edu/~xiaj/work/fasthss.pdf.

[12] S. Chandrasekaran, M. Gu, and W. Lyons, *A fast adaptive solver for hierarchically semiseparable representations*, Calcolo, 42 (2005), pp. 171–185.

[13] S. Chandrasekaran, M. Gu, and T. Pals, *Fast and Stable Algorithms for Hierarchically Semi-Separable Representations*, Technical report, Department of Mathematics, University of California, Berkeley, 2004.

[14] S. Chandrasekaran, M. Gu, and T. Pals, *A fast ULV decomposition solver for hierarchically semiseparable representations*, SIAM J. Matrix Anal. Appl., 28 (2006), pp. 603–622.

[15] J. W. Demmel, J. R. Gilbert, and X. S. Li, *SuperLU Users' Guide*, http://crd.lbl.gov/~xiaoye/SuperLU/superlu_ug.pdf.

[16] I. S. Duff and J. K. Reid, *The multifrontal solution of indefinite sparse symmetric linear equations*, ACM Trans. Math. Software, 9 (1983), pp. 302–325.

[17] Y. Eidelman and I. Gohberg, *On a new class of structured matrices*, Integral Equations Operator Theory, 34 (1999), pp. 293–324.

[18] Y. Eidelman, I. Gohberg, and V. Olshevsky, *The QR iteration method for Hermitian quasiseparable matrices of an arbitrary order*, Linear Algebra Appl., 404 (2005), pp. 305–324.

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

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

[21] J. R. Gilbert, G. L. Miller, and S.-H. Teng, *Geometric mesh partitioning and nested dissection*, in 12th Householder Symposium on Numerical Algebra, Plenary talk, Los Angeles, CA, 1993.

[22] J. R. Gilbert and J. W. H. Liu, *Elimination structures for unsymmetric sparse LU factors*, SIAM J. Matrix Anal. Appl., 14 (1993), pp. 334–352.

[23] J. R. Gilbert and R. E. Tarjan, *The analysis of a nested dissection algorithm*, Numer. Math., 50 (1987), pp. 377–404.

[24] L. Grasedyck, R. Kriemann, and S. Le Borne, *Parallel black box $\mathcal{H}$-LU preconditioning for elliptic boundary value problems*, Comput. Visual. Sci., 11 (2008), pp. 273–291.

[25] L. Grasedyck, R. Kriemann, and S. Le Borne, *Domain-decomposition based $\mathcal{H}$-LU preconditioners*, in Domain Decomposition Methods in Science and Engineering XVI, Springer

Lecture Notes Comput. Sci. Eng., O. B. Widlund and D. E. Keyes, eds., 55, 2006, pp. 661–668.

[26] W. Hackbusch, *A Sparse matrix arithmetic based on $\mathcal{H}$-matrices. Part* I: *Introduction to $\mathcal{H}$-matrices*, Computing, 62 (1999), pp. 89–108.

[27] W. Hackbusch and S. Börm, *Data-sparse approximation by adaptive $\mathcal{H}^2$-matrices*, Computing, 69 (2002), pp. 1–35.

[28] W. Hackbusch, L. Grasedyck, and S. Börm, *An introduction to hierarchical matrices*, Math. Bohem., 127 (2002), pp. 229–241.

[29] W. Hackbusch and B. N. Khoromskij, *A sparse $\mathcal{H}$-matrix arithmetic. Part*-II: *Application to multi-dimensional problems*, Computing, 64 (2000), pp. 21–47.

[30] W. Hackbusch, B. Khoromskij, and S. Sauter, *On $\mathcal{H}^2$-matrices*, in Lectures on Applied Mathematics, H. Bungartz, R. H. W. Hoppe, C. Zenger, eds., Springer, Berlin, 2000, pp. 9–29.

[31] A. J. Hoffman, M. S. Martin, and D. J. Rose, *Complexity bounds for regular finite difference and finite element grids*, SIAM J. Numer. Anal., 10 (1973), pp. 364–369.

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

[33] J. W. H. Liu, *The role of elimination trees in sparse factorization*, SIAM J. Matrix Anal. Appl., 18 (1990), pp. 134–172.

[34] J. W. H. Liu, *The multifrontal method for sparse matrix solution: Theory and practice*, SIAM Rev., 34 (1992), pp. 82–109.

[35] P. G. Martinsson, *A fast direct solver for a class of elliptic partial differential equations*, J. Sci. Comput., 38 (2009), pp. 316–330.

[36] METIS*, Family of Multilevel Partitioning Algorithms*, http://glaros.dtc.umn.edu/gkhome/views/metis.

[37] G. L. Miller, S.-H. Teng, W. Thurston, and S. A. Vavasis, *Automatic mesh partitioning, in Sparse Matrix Computations: Graph Theory Issues and Algorithms*, IMA Vol. Math. Appl. 56, A. George, J. Gilbert, and J. Liu, eds., Springer-Verlag, New York, 1993, pp. 57–84.

[38] G. L. Miller, S.-H. Teng, W. Thurston, and S. Vavasis, *Geometric separators for finite-element meshes*, SIAM J. Sci. Comput., 19 (1998), pp. 364–386.

[39] S. V. Parter, *The use of linear graphs in Gauss elimination*, SIAM Rev., 3 (1961), pp. 119–130.

[40] R. Schreiber, *A new implementation of sparse Gaussian elimination*, ACM Trans. Math. Software, 8 (1982), pp. 256–276.

[41] V. Strassen, *Gaussian elimination is not optimal*, Numer. Math., 13 (1969), pp. 354–356.

[42] S. Teng, *Fast nested dissection for finite element meshes*, SIAM J. Matrix Anal. Appl., 18 (1997), pp. 552–565.

[43] R. Vandebril, M. Van Barel, G. Golub, and N. Mastronardi, *A bibliography on semiseparable matrices*, Calcolo, 42 (2005), pp. 249–270.

[44] R. Vandebril, M. Van Barel, and N. Mastronardi, *A note on the representation and definition of semiseparable matrices*, Numer. Linear Algebra Appl., 12 (2005), pp. 839–858.

[45] C. H. Wolters, H. Köstler, C. Möller, J. Härdtlein, L. Grasedyck, and W. Hackbusch, *Numerical mathematics of the subtraction method for the modeling of a current dipole in EEG source reconstruction using finite element head models*, SIAM J. Sci. Comput., 30 (2007), pp. 24–45.

[46] J. Xia, *Fast Direct Solvers for Structured Linear Systems of Equations*, Ph.D. thesis, University of California, Berkeley, 2006.