SIAM J. SCI. COMPUT.
Vol. 35, No. 2, pp. A832–A860
© 2013 Society for Industrial and Applied Mathematics

EFFICIENT STRUCTURED MULTIFRONTAL FACTORIZATION FOR GENERAL LARGE SPARSE MATRICES*

JIANLIN XIA†

Abstract. Rank structures provide an opportunity to develop new efficient numerical methods for practical problems, when the off-diagonal blocks of certain dense intermediate matrices have small (numerical) ranks. In this work, we present a framework of structured direct factorizations for general sparse matrices, including discretized PDEs on general meshes, based on the multifrontal method and hierarchically semiseparable (HSS) matrices. We prove the idea of replacing certain complex structured operations by fast simple ones performed on compact reduced matrix forms. Such forms result from the hierarchical factorization of a tree-structured HSS matrix in a ULV-type scheme, so that the tree structure is reduced into a single node, the root of the original tree. This idea is shown to be very useful in the partial ULV factorization of an HSS matrix (for quickly computing Schur complements) as well as the solution stage. These techniques are then built into the multifrontal method for sparse factorizations after nested dissection, so as to convert the intermediate dense factorizations into fast structured ones. This method keeps certain Schur complements dense so as to avoid complicated data assembly, and is much simpler and more general than some existing methods. In particular, if the matrix arises from the discretization of certain PDEs, the factorization costs roughly $O(n)$ flops in two dimensions, and roughly $O(n^{4/3})$ flops or less in three dimensions. The solution cost and memory are nearly $O(n)$ in both cases. These counts are obtained with an idea of rank relaxation, so that this method is more generally applicable to problems where the intermediate off-diagonal ranks are not small. We demonstrate the performance of the method with two- and three-dimensional discretized equations, as well as various examples from a sparse matrix collection. The ideas here are also useful in future developments of fast structured solvers.

Key words. structured multifrontal method, general sparse matrix, HSS matrix, ULV factorization, reduced matrix, sparse rank relaxation

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

DOI. 10.1137/120867032

1. Introduction.

In scientific computations and engineering simulations, the major computational work is often to solve large sparse linear systems. Consider a linear system

$$Ax = b,\ A : n \times n. \tag{1.1}$$

Classical direct solvers for (1.1) are robust and are efficient for multiple right-hand sides. However, they are often expensive in the costs and memory, due to the creation of *fill-in* in the factorization. In fact, if A arises from the discretization of an $N \times N$ two-dimensional (2D) mesh, it needs at least $O(n^{3/2})$ flops to factor A [24]. For three dimensions, this cost is $O(n^2)$, and even the solution costs $O(n^{4/3})$. Iterative methods can take good advantage of the sparsity using matrix-vector multiplications. But without good preconditioners, iterative methods may converge very slowly or may not even converge.

A lot of recent developments focus on structured approximate factorizations. It has been observed that some discretized PDEs and integral equations have a *low-rank property*. That is, the fill-in in their direct factorization has low-rank (or low-

*Submitted to the journal's Methods and Algorithms for Scientific Computing section February 22, 2012; accepted for publication (in revised form) December 17, 2012; published electronically March 21, 2013.

http://www.siam.org/journals/sisc/35-2/86703.html

†Department of Mathematics, Purdue University, West Lafayette, IN 47907 (xiaj@math.purdue.edu). This author's work was supported in part by NSF grants DMS-1115572 and CHE-0957024.

numerical-rank) off-diagonal blocks [1, 2, 8, 18, 19, 26, 27, 28, 39]. Based on this property, the fill-in can be approximated by rank-structured matrices such as quasiseparable, semiseparable, or hierarchical ($\mathcal{H}$-, $\mathcal{H}^2$-) matrices (see, e.g., [3, 4, 5, 7, 13, 22, 34] for some introductions and surveys). Examples of such methods include $\mathcal{H}$-LU methods [18, 19] and structured multifrontal methods [39]. These methods often significantly reduce classical lower complexity bounds for direct factorizations.

The structured multifrontal method in [39] employs the multifrontal method [12, 25] and hierarchically semiseparable (HSS) matrix representations [6, 9, 40], after the nested dissection ordering of A [16]. For 2D elliptic equations, the method computes structured approximate factorizations with nearly linear complexity and linear storage. The major intermediate operations are performed in HSS forms, including dense factorization, block permutation, splitting, merging, etc. This thus makes the algorithm difficult to implement. The existing implementation in [39] concentrates on symmetric positive definite (SPD) problems on 2D regular meshes. A variation of the method is given in [32]. However, this variation still focuses mainly on 2D meshes where the orientation of the mesh points roughly follow regular meshes. Also, it uses inversions for the intermediate structured matrices, followed by matrix-vector multiplications. These are often slower than ULV-type factorizations in [39].

This paper proposes a new structured multifrontal method which is much simpler and more general. It can be applied to discretized matrices in both two and three dimensions, as well as general sparse matrices. The work includes general sparse matrix reordering with graph partitioning, new improved HSS factorizations, a flexible factorization framework, and relaxed rank requirements. This is explained as follows.

(1) General sparse matrix. Our nested dissection reordering is performed with the aid of graph partitioning tools. Separators [16] in nested dissection are allowed to have general shapes and orientations. We also accommodate general connectivity of the separators. These make the method more widely applicable than those in [32, 39]. In addition, the new formulas and concepts in this paper are given for both symmetric and nonsymmetric situations.

(2) Improved HSS algorithms and a concept of reduced matrices. HSS matrices are used to approximate dense intermediate fill-in, and then factored in our modified factorization scheme with improved efficiency.

Furthermore, we introduce a very useful concept of *reduced (HSS) matrices*, which is not involved in [39]. That is, the ULV factorization of an HSS matrix F generally results in smaller intermediate matrices (called reduced matrices) after orthogonal transformations or triangular factorizations. The original HSS matrix corresponds to a tree structure called an HSS tree, which is also reduced accordingly. The overall ULV factorization leads to a final reduced matrix which is generally much smaller and corresponds to one single node of the HSS tree, or its root. We prove that various complex operations involving F can be replaced by simple ones performed on this final reduced matrix. The related HSS operation costs can then be reduced from, say, $O\left(r^2N\right)$ to $O\left(r^3\right)$, where N is the order of F and r is its maximum off-diagonal rank. Such an idea is used for the fast computation of the intermediate Schur complements in our structured sparse factorization and also the fast solution. It can also benefit the method in [39] as well as our future developments.

(3) Simplified structured sparse factorization. We organize the overall factorization into a simplified scheme. The multifrontal method converts the sparse factorization into that of a sequence of local dense ones, called *frontal matrices*. Partial factorizations of the frontal matrices provide certain columns of the factors, and the intermediate Schur complements are called *update matrices*. Here, unlike the method

in [39], we approximate the frontal matrices by HSS forms, but keep the update matrices dense. This enables us to avoid complicated data assembly (called *extend-add* operation [25]) for structured update matrices. The HSS extend-add operation in [39] requires special separator connectivity as in regular meshes. Our new scheme uses the classical extend-add operation so as to accommodate more general problems, and it produces factors with the same structure as that by a fully structured version.

(4) Sparse rank relaxation. we systematically relax the classical rank requirement in structured sparse algorithms. Traditional HSS operations often require the related off-diagonal (numerical) ranks to be bounded in order to achieve high efficiency. In [42] it is shown that the same or similar complexity can be obtained for HSS operations without this requirement. That is, the ranks are actually allowed to increase along the block sizes. Such rank phenomenon is indeed observed for the intermediate matrices in the factorization of certain sparse discretized PDEs. Thus, we generalize the dense rank relaxation idea in [42] to sparse factorizations. This enhances both the flexibility and the applicability of structured factorizations.

(5) Two and three dimensions. With certain flexible rank conditions, we can achieve satisfactory factorization costs for both 2D and three-dimensional (3D) problems, which are roughly $O(n)$ and $O(n^{4/3})$ flops, respectively (see Theorem 4.3). The storage requirement in both two and three dimensions is nearly $O(n)$. The complexity of solutions with the structured factors is also nearly $O(n)$. Different criteria are used to choose a switching level (for allowing certain levels of dense local factorizations) so as to optimize either the factorization or the solution complexity.

These types of structured approximate factorizations are very attractive for real applications such as Helmholtz equations in seismic imaging, where linear systems with a large number of right-hand sides are solved, but only modest accuracy is desired.

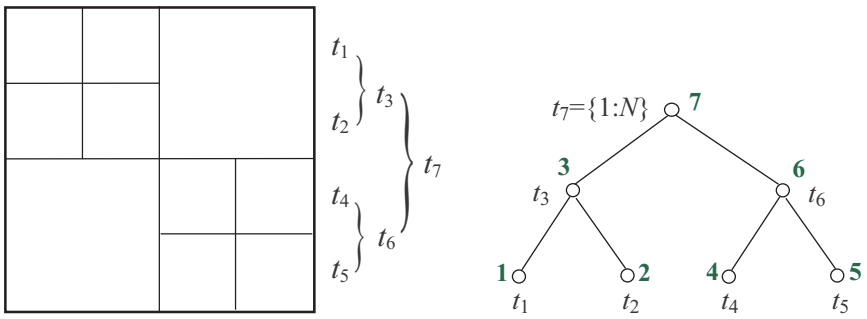
The remaining sections are organized as follows. HSS structures are reviewed in section 2, followed by some improved HSS algorithms. Section 3 presents our new structured factorization and solution schemes. The algorithms are summarized and analyzed in section 4, which shows the idea of sparse rank relaxation. Section 5 provides some numerical examples for 2D and 3D Helmholtz equations and more general examples from a classical sparse matrix collection. We draw some concluding remarks in section 6. The following notation is used:

- $F|_{t_i \times t_j}$ is the submatrix of F specified by the row index set t_i and column index set t_j, and $F|_{:\times t_j}$ is the submatrix formed by the columns of F specified by the index set t_j.
- $b|_{t_i}$ is a vector formed by the entries of the vector b from the row index set t_i.
- $\mathrm{diag}\,(A_1, \ldots, A_k)$ is a block diagonal matrix with the diagonal blocks $A_1, \ldots, A_k$.
- $\mathrm{root}\,(\mathbf{T})$ is the root of a binary tree $\mathbf{T}$, and $\mathrm{par}(j)$ and $\mathrm{sib}\,(j)$ denote the parent and sibling nodes of node j in $\mathbf{T}$, respectively, with the nodes of $\mathbf{T}$ labeled by $j = 1, 2, \ldots$.
- $F_1 \bigoplus F_2$ denotes an extend-add operation [12, 25], which permutes and expands the matrices F_1 and F_2 following a certain global index set and then adds the resulting matrices (section 3.2).

2. HSS structures and algorithms.

2.1. Review of HSS structures.

HSS structures are very useful in handling dense matrices with the low-rank property. An HSS representation is generally defined recursively [6, 9]. The definition for a postordered HSS matrix is as follows [40].



(i) Hierarchical partition (ii) Binary tree for the index sets

FIG. 2.1. *Two levels of block partition of an $N \times N$ matrix and the organization of the index sets with a binary tree.*

Assume F is an $N \times N$ dense matrix, and $\mathcal{I} = \{1 : N\} \equiv \{1, 2, \ldots, N\}$. Let $\mathbf{T}$ be a binary tree with k nodes, denoted by $j = 1, 2, \ldots, k \equiv \text{root}\,(\mathbf{T})$, and $t_j \subset \mathcal{I}$ be a subset of contiguous indices in $\mathcal{I}$ associated with each node j of $\mathbf{T}$. (Initially, $\mathbf{T}$ is used to manage the recursive splitting of $\mathcal{I}$ and the recursive partition of F.) We say F is in an *HSS form* with the corresponding postordered *HSS tree* $\mathbf{T}$ if the following hold:

(1) $\mathbf{T}$ is a full binary tree in its postordering, or each node j is either a leaf or a nonleaf node with two children c_1 and c_2 which satisfy $c_1 < c_2 < j$.
(2) The index sets satisfy $t_{c_1} \cup t_{c_2} = t_j$ and $t_{c_1} \cap t_{c_2} = \emptyset$ for each nonleaf node j, with $t_k \equiv \mathcal{I}$.
(3) For each node j, there exist matrices $D_j, U_j, V_j, R_j, W_j, B_j$ (called *HSS generators*), which satisfy the following recursions for each nonleaf node j:

$$D_j \equiv F|_{t_j \times t_j} = \begin{pmatrix} D_{c_1} & U_{c_1} B_{c_1} V_{c_2}^T \\ U_{c_2} B_{c_2} V_{c_2}^T & D_{c_2} \end{pmatrix}, \; U_j = \begin{pmatrix} U_{c_1} R_{c_1} \\ U_{c_2} R_{c_2} \end{pmatrix}, \; V_j = \begin{pmatrix} V_{c_1} W_{c_1} \\ V_{c_2} W_{c_2} \end{pmatrix},$$

where U_k, V_k, R_k, W_k, and B_k are not needed (since $D_k \equiv F$ is the entire diagonal block without a corresponding off-diagonal one).

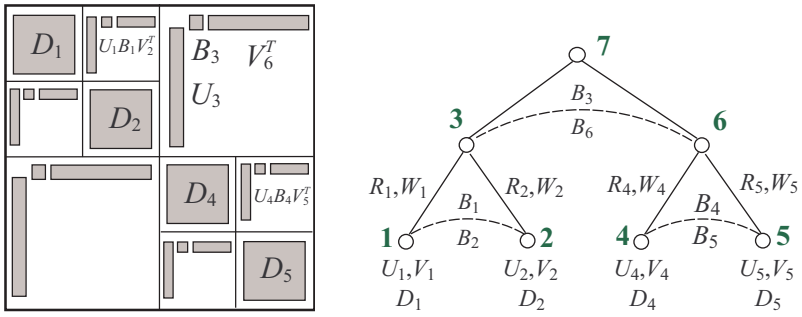
Only the D_j, U_j, V_j generators associated with a leaf node j of $\mathbf{T}$ are stored. Here, U_j and V_j are called *basis matrices* since the columns of U_j and the rows of V_j^T form bases of the *HSS (off-diagonal) blocks*

$$F_j^{-} \equiv F|_{t_j \times (\mathcal{I} \setminus t_j)} \quad \text{and} \quad F_j^{|} \equiv F|_{(\mathcal{I} \setminus t_j) \times t_j}, \tag{2.1}$$

respectively. Also, we call the block row of F formed by D_j and F_j^{-} the jth *block row* of F. Similarly, define the jth *block column*. Clearly, a low-rank off-diagonal block used in HSS representations is an entire block row or column without the diagonal block. On the other hand, $\mathcal{H}$-matrices involve low-rank blocks more general than (2.1) [18, 19, 22], and HSS matrices can be considered as a special case of $\mathcal{H}$-matrices.

For example, for an $N \times N$ matrix F as shown in Figure 2.1(i), we partition it following a hierarchical splitting of the set $t_7 = \{1 : N\}$ as illustrated in the binary tree form in Figure 2.1(ii). Then we can define an HSS form and an HSS tree $\mathbf{T}$ as in Figure 2.2.

Moreover, the HSS tree can help quickly identify any off-diagonal block of the matrix [9, 40]. For example, the block corresponding to nodes 1 and 4 in Figure 2.2(ii)



(i) HSS matrix (ii) HSS tree

FIG. 2.2. *A block* 4×4 *HSS matrix and the corresponding HSS tree.*

can be decided by visiting the path connecting these two nodes: $1 \to 3 \to 6 \to 4$. That is

$$F|_{t_1 \times t_4} = U_1 R_1 B_3 W_4^T V_4^T.$$

An HSS form for F can be constructed with recursive *compression* of the HSS blocks F_j^- and $F_j^|$ [9, 40]. For example, in a construction scheme in [36, 38], F_j^- is compressed as follows to get the U, R generators. If j is a leaf of the HSS tree $\mathbf{T}$, compute a QR factorization

$$F_j^- = U_j G_j.$$

(Note: If rank-revealing factorizations are used for the compression, then the HSS form approximates F.) If j is a nonleaf node with children c_1 and c_2, stack the columns of G_{c_1} and G_{c_2} whose column indices match $\mathcal{I} \backslash t_j$ (the column indices of F_j^-) and compute a QR factorization

$$\begin{pmatrix} G_{c_1}|_{:\times(\mathcal{I}\backslash t_j)} \\ G_{c_2}|_{:\times(\mathcal{I}\backslash t_j)} \end{pmatrix} = \begin{pmatrix} R_{c_1} \\ R_{c_2} \end{pmatrix} G_j.$$

Then by recursion, we can verify that $F_j^- = U_j F|_{\hat{t}_j \times (\mathcal{I}\backslash t_j)}$. Similarly, we compress $F_j^|$ to get the V, W generators, and then extract the B generators [38].

If F is symmetric, only F_j^- needs to be compressed. Then $D_j = D_j^T$, and we can set [40]

$$V_j = U_j, \ B_i = B_j^T,$$

where $i = \text{sib}\,(j)$.

2.2. An improved ULV factorization and reduced HSS matrices. An HSS linear system can be quickly solved with ULV-type factorizations and solutions [9, 40, 38]. We briefly review the original version in [9] and then show a modified one. For convenience, assume the D_j blocks corresponding to all leaves j of the HSS tree $\mathbf{T}$ have sizes m, and all HSS blocks have ranks r. The scheme in [9] includes the following major steps.

(1) For a leaf j, introduce zeros into F_j^- by multiplying Q_j^T to F_j^-, where Q_j is obtained from a full QL factorization

$$U_j = Q_j \begin{pmatrix} 0 \\ \hat{U}_j \end{pmatrix} \begin{matrix} m-r \\ r \end{matrix}.$$

Also update the diagonal block D_j to

$$\bar{D}_j = Q_j^T D_j \equiv \begin{array}{c} \\ m-r \\ r \end{array} \begin{array}{c} \begin{array}{cc} m-r & r \end{array} \\ \begin{pmatrix} \bar{D}_{j;1,1} & \bar{D}_{j;1,2} \\ \bar{D}_{j;2,1} & \bar{D}_{j;2,2} \end{pmatrix} \end{array} \begin{array}{c} \\ m-r \\ r \end{array}. \tag{2.2}$$

(2) Compute an LQ factorization

$$\begin{pmatrix} \bar{D}_{j;1,1} & \bar{D}_{j;1,2} \end{pmatrix} = \begin{array}{c} \begin{array}{cc} m-r & r \end{array} \\ \begin{pmatrix} L_j & 0 \end{pmatrix} \end{array} P_j.$$

Multiply P_j^T to $\bar{D}_j$ and $F_j^|$ on the right by updating $\bar{D}_j$ and V_j, respectively:

$$\bar{D}_j P_j^T \equiv \begin{array}{c} \\ m-r \\ r \end{array} \begin{array}{c} \begin{array}{cc} m-r & r \end{array} \\ \begin{pmatrix} L_j & \\ \tilde{D}_{j;2,1} & \tilde{D}_j \end{pmatrix} \end{array}, \quad P_j V_j \equiv \begin{array}{c} \\ m-r \\ r \end{array} \begin{array}{c} \\ \begin{pmatrix} \tilde{V}_j \\ \hat{V}_j \end{pmatrix} \end{array}.$$

(3) Eliminate L_j and remove node j from the tree $\mathbf{T}$.

(4) If j is a nonleaf node with children c_1 and c_2 which has been eliminated in the previous steps, merge blocks to obtain new generators:

(2.3)
$$\tilde{D}_j = \begin{pmatrix} \hat{D}_{c_1} & \hat{U}_{c_1} B_{c_1} \hat{V}_{c_2}^T \\ \hat{U}_{c_2} B_{c_2} \hat{V}_{c_1}^T & \hat{D}_{c_2} \end{pmatrix}, \ \tilde{U}_j = \begin{pmatrix} \hat{U}_{c_1} R_{c_1} \\ \hat{U}_{c_2} R_{c_2} \end{pmatrix}, \ \tilde{V}_j = \begin{pmatrix} \hat{V}_{c_1} W_{c_1} \\ \hat{V}_{c_2} W_{c_2} \end{pmatrix}.$$

Then j becomes a leaf corresponding to generators $\tilde{D}_j, \tilde{U}_j, \tilde{V}_j, B_j$, and F is reduced to a smaller HSS matrix, called a *reduced matrix*.

DEFINITION 2.1. *In the ULV factorization of an HSS matrix F, a new HSS matrix with generators in* (2.3) *resulting from the elimination of the children of a node j of the HSS tree is called a reduced (HSS) matrix.*

The above process then repeats for the reduced matrix.

This procedure can be modified to improve the efficiency, especially to reduce the costs for dense block multiplications such as the one in (2.2). We convert each diagonal block to an identity matrix and then preserve it. A similar method has been proposed for SPD matrices [38]. Here, we handle nonsymmetric ones. The details are presented since they are needed in Theorems 3.1 and 3.3 later.

For a leaf j of $\mathbf{T}$, compute an LU factorization

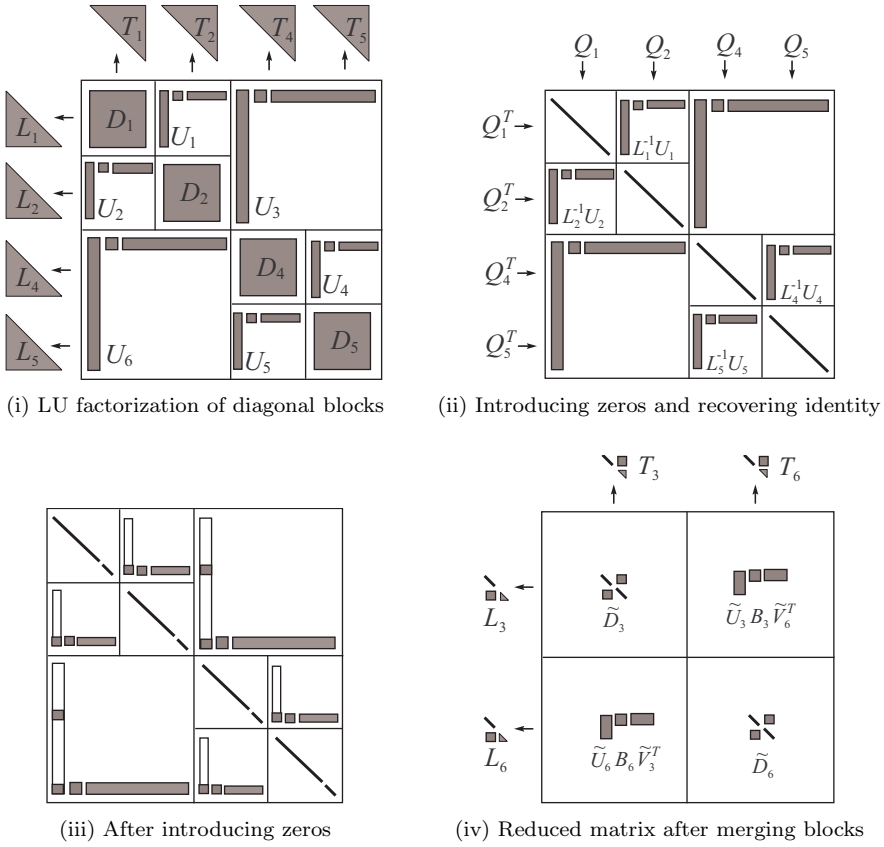
$$D_j = L_j T_j. \tag{2.4}$$

Then multiply L_j^{-1} to the jth block row of F on the left, and T_j^{-1} to the jth block column of F on the right, so as to convert the diagonal block to an identity matrix. This is done via the update of the generators:

$$\bar{D}_j, = I, \ \bar{U}_j = L_j^{-1} U_j, \ \bar{V}_j = T_j^{-T} V_j. \tag{2.5}$$

See Figure 2.3(i). Next, compute a full QL factorization

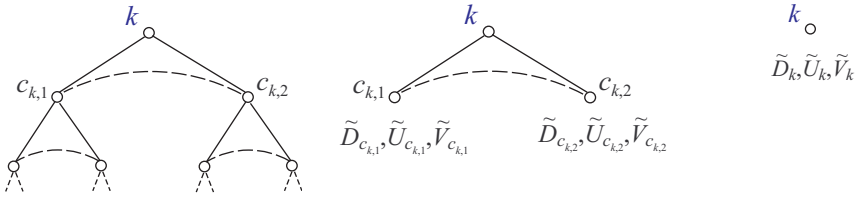
$$\bar{U}_j = Q_j \begin{pmatrix} 0 \\ \tilde{U}_j \end{pmatrix}, \tag{2.6}$$



(i) LU factorization of diagonal blocks (ii) Introducing zeros and recovering identity

(iii) After introducing zeros (iv) Reduced matrix after merging blocks

FIG. 2.3. *An improved ULV factorization scheme for a nonsymmetric HSS matrix.*



(i) An HSS tree with root k (ii) After eliminating lower levels (iii) Only the root left

FIG. 2.4. *Elimination of the nodes of the HSS tree in the ULV HSS factorization schemes.*

and multiply Q_j^T to $\bar{U}_j$ on the left, so that the first $m-r$ rows of the updated off-diagonal block row $Q_j^T L_j^{-1} F_j^-$ are zeros. Also, update $\bar{V}_j$ to

$$Q_j^T \bar{V}_j \equiv \begin{pmatrix} \tilde{V}_j \\ \hat{V}_j \end{pmatrix} \begin{matrix} m-r \\ r \end{matrix}. \tag{2.7}$$

The diagonal block becomes an identity matrix again. See Figure 2.3(ii)–(iii). After this, the diagonal identity matrix can be partially eliminated, as in Figure 2.3(iii)–(iv). Then remove node j from $\mathbf{T}$, which is reduced to a smaller HSS tree for a reduced HSS matrix. See Figure 2.4.

If j is a nonleaf node with its children c_1 and c_2 partially eliminated as above, we merge appropriate blocks just like (2.3), except $\hat{D}_{c_1} = I$ and $\hat{D}_{c_2} = I$ in (2.3) (Figure 2.3(iv)). This is useful in improving the efficiency of upper-level eliminations. For example, the upper-level LU factorization of D_j is

$$D_j = L_j T_j \equiv \begin{pmatrix} I & \\ \tilde{U}_{c_2} B_{c_2} V_{c_1}^T & \tilde{L}_j \end{pmatrix} \begin{pmatrix} I & \tilde{U}_{c_1} B_{c_1} \tilde{V}_{c_2}^T \\ & \tilde{T}_j \end{pmatrix}, \tag{2.8}$$

where $\tilde{L}_j \tilde{T}_j$ is the LU factorization of the Schur complement $I - \tilde{U}_{c_2} B_{c_2} V_{c_1}^T \tilde{U}_{c_1} B_{c_1} \tilde{V}_{c_2}^T$. Similarly, (2.5) can also be accelerated due to the special form of L_j and T_j. The elimination proceeds until the root k is reached. Then we compute a direct LU factorization as in (2.4) for $j = k$.

Similar to the idea in [40], it can be verified that the ULV factorization has a form

$$H = \mathbf{L}\mathbf{U},$$

where $\mathbf{L}$ and $\mathbf{U}$ are given by a sequence of block orthogonal and triangular matrices.

2.3. ULV HSS solution. The ULV HSS solution scheme follows the idea in [38], but is presented with more details here for later use.

In a forward-substitution stage, we traverse the HSS tree in a bottom-up order to solve the system

$$\mathbf{L}y = b. \tag{2.9}$$

In the process, there is a piece of b and a piece of y associated with each node j of $\mathbf{T}$, denoted b_j and y_j, respectively. Initially, $b_j \equiv b|_{t_j}$ for all leaves j.

For a leaf j, let

$$\tilde{b}_j = Q_j^T L_j^{-1} b_j \equiv \begin{pmatrix} \tilde{b}_{j,1} \\ \tilde{b}_{j,2} \end{pmatrix} \begin{matrix} m-r \\ r \end{matrix}, \quad y_j \equiv \tilde{b}_{j,1}, \quad z_j = \tilde{V}_j^T y_j, \tag{2.10}$$

where $\tilde{V}_j$ is given in (2.7). For a nonleaf node j with children c_1 and c_2, set

$$b_j = \begin{pmatrix} \tilde{b}_{c_1,2} \\ \tilde{b}_{c_2,2} \end{pmatrix} - \begin{pmatrix} \tilde{U}_{c_1} B_{c_1} z_{c_2} \\ \tilde{U}_{c_2} B_{c_1} z_{c_1} \end{pmatrix}.$$

Then we similarly compute y_j as in (2.10), except to set

$$z_j = W_{c_1}^T z_{c_1} + W_{c_2}^T z_{c_2} + \tilde{V}_j^T y_j.$$

These operations can be applied recursively, until the root node k is reached, where we compute

$$y_k = L_k^{-1} b_k \tag{2.11}$$

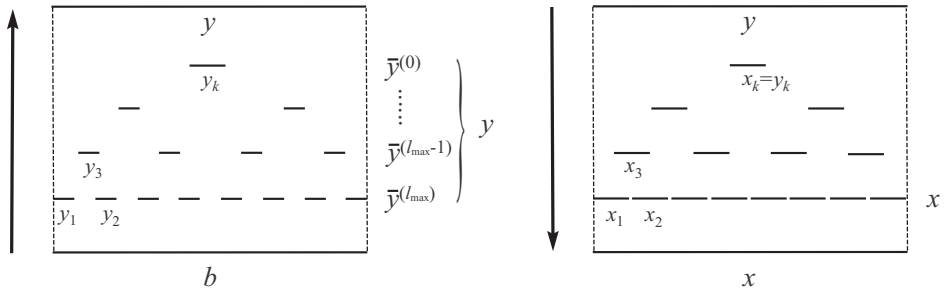
See Figure 2.5(i). Then we merge all y_j pieces with appropriate permutations to form y. The details are shown in the proof of Theorem 3.3 (as in (3.25)).

In a backward-substitution stage, we traverse the HSS tree in a top-down order to solve the system

$$\mathbf{U}x = y. \tag{2.12}$$

Initially, let

$$x_k = T_k^{-1} y_k.$$



(i) Forward substitution for solving (2.9) (ii) Backward substitution for solving (2.12)

FIG. 2.5. *ULV HSS solution process, where the solution pieces are illustrated by horizontal bars.*

For each nonleaf node j with children c_1 and c_2, partition y_j as $y_j = \begin{pmatrix} y_{j,1} \\ y_{j,2} \end{pmatrix} \begin{matrix} r \\ r \end{matrix}$ and compute

$$x_{c_1} = T_{c_1}^{-1} Q_{c_1} \begin{pmatrix} \tilde{y}_{c_1} \\ y_{j,1} \end{pmatrix}, \quad x_{c_2} = T_{c_2}^{-1} Q_{c_2} \begin{pmatrix} \tilde{y}_{c_2} \\ y_{j,2} \end{pmatrix}. \tag{2.13}$$

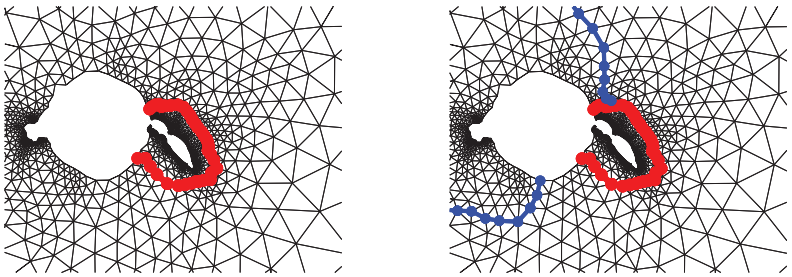
When all the nonleaf nodes are visited, stack the pieces x_j (by setting $x|_{t_j} \equiv x_j$) for all the leaves j to form x. See Figure 2.5(ii).

3. New structured multifrontal method. In this section, we present our new structured sparse factorization method. The HSS algorithms in the previous sections are applied to the dense intermediate matrices in a multifrontal factorization framework. For simplicity, we assume A is SPD in the discussions. For nonsymmetric ones, the idea of using reduced matrices is also shown (Corollaries 3.2 and 3.5), and the algorithm can be similarly derived. Note that we do not consider pivoting issues since they are not our focus. In fact, we may use static pivoting in a preprocessing step similar to that in `SuperLU` [11], followed by iterative refinements in a postprocessing step.

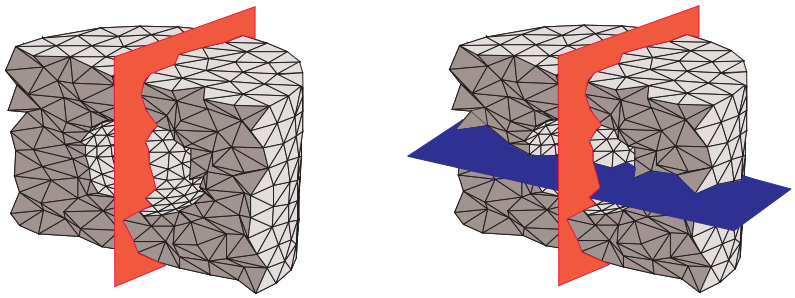
3.1. Nested dissection and separator partitioning for general adjacency graphs. For a sparse symmetric matrix A, the adjacency graph has a vertex corresponding to each row/column of A, so that there is an edge (i, j) connecting vertices i and j if $a_{ij} = a_{ji} \neq 0$. For a discretized matrix, the mesh can often serve as the adjacency graph. In nested dissection [16], the graph is recursively divided with *separators* (small sets of vertices). A top-level separator divides the graph into two subregions, which are further divided recursively. See Figure 3.1. Lower-level separators are ordered and eliminated before upper-level ones. The elimination of a separator mutually connects its (upper-level) neighbors which creates fill-in [30, 33]. (When we say the *neighbors* of a separator we mean those separators which are ordered after this separator and are connected to it due to the elimination of lower-level separators.) Nested dissection is very useful in reducing the fill-in. In fact, it can in general help the factorization of a discretized matrix in two or three dimensions achieve the theoretical lower complexity bounds tightly [24].

Here in the context of our structured solution method, we allow the flexibility of using an adjacency graph or mesh which is irregular. Since the graph partition is not the major focus of this work, we only point out the following aspects:

(1) Unlike [39], we can handle separators (and their neighbors) with general orientation and connectivity. We use graph partitioning tools such as `METIS`



(i) Two levels of partitions of a 2D mesh (image based on [17] for a matrix from [10])



(ii) Two levels of partitions of a 3D mesh (image based on [31])

FIG. 3.1. *Graph partitioning in nested dissection ordering.*

[29] to construct a nested dissection ordering. The ordering needs only the matrix A, although additional mesh information (if any) can help improve the quality of the ordering.

(2) Each separator is partitioned into multiple pieces, which correspond to the partitions of appropriate HSS matrices. The partition information can be accumulated from lower levels, so as to preserve the graph connectivity.

3.2. Review of the multifrontal method. As one of the most important factorization methods, the multifrontal method [12, 25] converts a sparse factorization into a sequence of factorizations of smaller intermediate dense matrices. A tree structure called *elimination tree* or *assembly tree* is used so that these local factorizations can be done independently at each level. For convenience, assume the sparse matrix A is SPD with the Cholesky factorization $A = \mathcal{L}\mathcal{L}^T$. The elimination tree $\mathcal{T}$ has n nodes corresponding to the n rows/columns of A, and the parent of a node i of $\mathcal{T}$ is defined as

$$\mathrm{par}(i) = \min\{j > i\colon\ L|_{j\times i} \neq 0\}.$$

Let $\mathcal{T}[i]$ be the subtree of $\mathcal{T}$ with root i, nodes $c_1, c_2, \ldots, c_q$ be the children of i, and $\mathcal{N}_i \equiv \{j_1, j_2, \ldots, j_d\}$ be the set of nonzero row indices in $\mathcal{L}|_{(1:n)\times i}$. The ith *frontal matrix* is defined to be

$$\mathcal{F}_i = \begin{pmatrix} A|_{i\times i} & \left(A|_{\mathcal{N}_i\times i}\right)^T \\ A|_{\mathcal{N}_i\times i} & 0 \end{pmatrix} - \sum_{j\in\mathcal{T}[i]\setminus i} \mathcal{L}|_{(i\cup\mathcal{N}_i)\times j}\left(\mathcal{L}|_{(i\cup\mathcal{N}_i)\times j}\right)^T \tag{3.1}$$

$$\equiv \mathcal{F}_i^0 \oplus\!\!\!\!\!- \mathcal{U}_{c_1} \oplus\!\!\!\!\!- \mathcal{U}_{c_2} \oplus\!\!\!\!\!- \cdots \oplus\!\!\!\!\!- \mathcal{U}_{c_q},$$

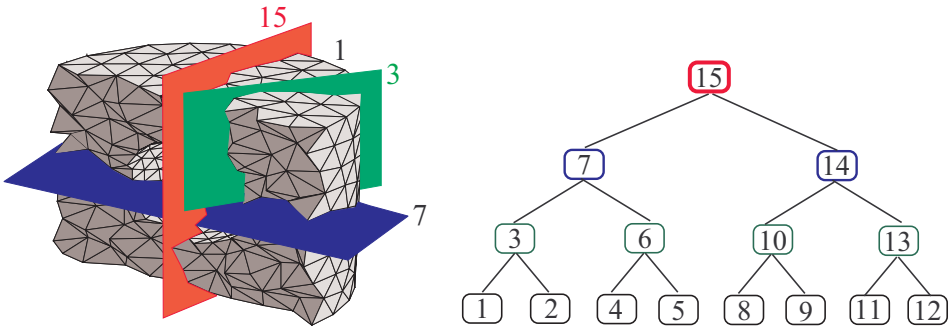


FIG. 3.2. *Nested dissection of a 3D mesh and the corresponding assembly tree $\mathcal{T}$ for the separators.*

where each $\mathcal{U}_{c_j}$ is called the c_jth *update matrix* obtained from $\mathcal{F}_{c_j}$ by recursion as in (3.2) below, and $\dotplus$ denotes an *extend-add* operation, which permutes and expands the matrices following a global index set and then adds the matrix entries. For example, assume that $\mathcal{F}_i^0$, $\mathcal{U}_{c_1}$, and $\mathcal{U}_{c_2}$ correspond to index sets $\{1,2,3\}$, $\{3,1\}$, and $\{3,2\}$, respectively, and

$$\mathcal{F}_i^0 = \begin{pmatrix} a_{11} & a_{21} & a_{31} \\ a_{21} & a_{22} & a_{32} \\ a_{31} & a_{32} & a_{33} \end{pmatrix}, \ \mathcal{U}_{c_1} = \begin{pmatrix} u_{11}^{(1)} & u_{21}^{(1)} \\ u_{21}^{(1)} & u_{22}^{(1)} \end{pmatrix}, \ \mathcal{U}_{c_2} = \begin{pmatrix} u_{11}^{(2)} & u_{21}^{(2)} \\ u_{21}^{(2)} & u_{22}^{(2)} \end{pmatrix}.$$

Then the extend-add operation is performed as

$$\mathcal{F}_i^0 \dotplus \mathcal{U}_{c_1} \dotplus \mathcal{U}_{c_2} = \begin{pmatrix} a_{11} & a_{21} & a_{31} \\ a_{21} & a_{22} & a_{32} \\ a_{31} & a_{32} & a_{33} \end{pmatrix} + \begin{pmatrix} u_{22}^{(1)} & 0 & u_{21}^{(1)} \\ 0 & 0 & 0 \\ u_{21}^{(1)} & 0 & u_{11}^{(1)} \end{pmatrix} + \begin{pmatrix} 0 & 0 & 0 \\ 0 & u_{22}^{(2)} & u_{21}^{(2)} \\ 0 & u_{21}^{(2)} & u_{11}^{(2)} \end{pmatrix}.$$

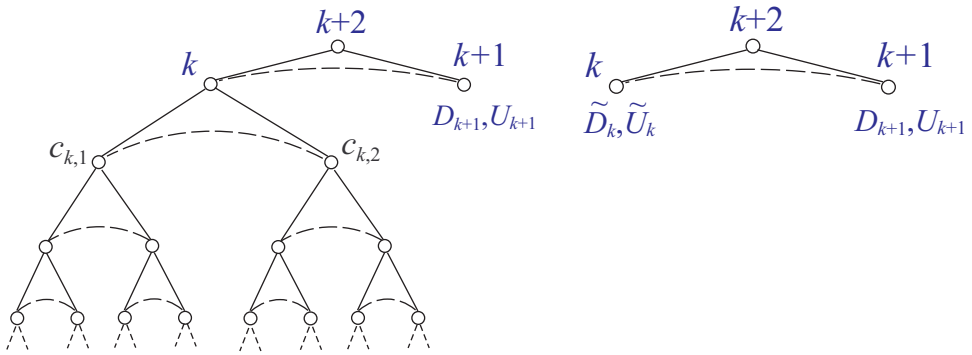
Factoring the leading entry in (3.1) yields one column of $\mathcal{L}$:

$$\mathcal{F}_i = \begin{pmatrix} \mathcal{L}|_{i\times i} & \\ \mathcal{L}|_{\mathcal{N}_i\times i} & I \end{pmatrix} \begin{pmatrix} I & \\ & \mathcal{U}_i \end{pmatrix} \begin{pmatrix} (\mathcal{L}|_{i\times i})^T & (\mathcal{L}|_{\mathcal{N}_i\times i})^T \\ & I \end{pmatrix}, \tag{3.2}$$

where $\mathcal{U}_i$ is the Schur complement (the ith update matrix). Following (3.1)–(3.2), the factorization process repeats for all the nodes of $\mathcal{T}$.

3.3. Structured multifrontal method. We derive an efficient structured sparse solver based on a supernodal version of the multifrontal method. That is, nested dissection in section 3.1 is used to reorder the adjacency graph of A so that a binary assembly tree $\mathcal{T}$ is formed, where the separators are denoted by $\mathbf{i} = 1, 2, \ldots$. Each separator $\mathbf{i}$ is treated as a node in $\mathcal{T}$ and replaces the index i in section 3.2. See Figure 3.2. This is similar to the method in [39] for 2D regular meshes, but is more general.

Let $\mathcal{N}_\mathbf{i} \equiv \{\mathbf{j}_1, \mathbf{j}_2, \ldots, \mathbf{j}_d\}$ be the set of neighbor separators of separator $\mathbf{i}$. For example, for separator $\mathbf{i} = 1$ in Figure 3.2, $\mathcal{N}_\mathbf{i} = \{3, 7, 15\}$. Assume separator $\mathbf{j}$ corresponds to the index set $t_\mathbf{j}$ for A, and $\hat{t}_\mathbf{j}$ is the subset of $t_\mathbf{j}$ that is connected to separator $\mathbf{i}$ (due to nonzeros in A or fill-in created by earlier eliminations [23]). If $\mathbf{c}_1$



(i) HSS tree for $\mathbf{F_i}$ (ii) After the ULV factorization of H

FIG. 3.3. *HSS tree* $\mathbf{T_i}$ *for* $\mathbf{F_i}$ *before and after the partial factorization (ULV factorization of* H*).*

and $\mathbf{c}_2$ are the children of $\mathbf{i}$ in $\mathcal{T}$, then the frontal matrix $\mathcal{F}_\mathbf{i}$ is formed by the block form extend-add operation

$$\mathbf{F}_i = \mathbf{F}_\mathbf{i}^0 \Diamond \mathbf{U}_{\mathbf{c}_1} \Diamond \mathbf{U}_{\mathbf{c}_2}, \quad \text{with} \quad \mathbf{F}_\mathbf{i}^0 \equiv \begin{pmatrix} A|_{t_\mathbf{i}\times t_\mathbf{i}} & (A|_{(\cup_{\mathbf{j}=1}^d \hat{t}_\mathbf{j})\times t_\mathbf{i}})^T \\ A|_{(\cup_{\mathbf{j}=1}^d \hat{t}_\mathbf{j})\times t_\mathbf{i}} & 0 \end{pmatrix}. \tag{3.3}$$

Partition $\mathbf{F_i}$ conformably as

$$\mathbf{F_i} \equiv \begin{pmatrix} F_{\mathbf{i},\mathbf{i}} & F_{\mathcal{N}_\mathbf{i},\mathbf{i}}^T \\ F_{\mathcal{N}_\mathbf{i},\mathbf{i}} & F_{\mathcal{N}_\mathbf{i},\mathcal{N}_\mathbf{i}} \end{pmatrix}. \tag{3.4}$$

Our structured multifrontal method has the following major steps:

(1) Approximate $\mathbf{F_i}$ by an HSS matrix.
(2) Partially factor $\mathbf{F_i}$ with a ULV factorization scheme.
(3) Compute the dense Schur complement or update matrix $\mathbf{U}_i$ with a low-rank update.
(4) Perform the dense extend-add operation as in the standard multifrontal method for (3.3). The details of this can be found in [12] and are skipped here.

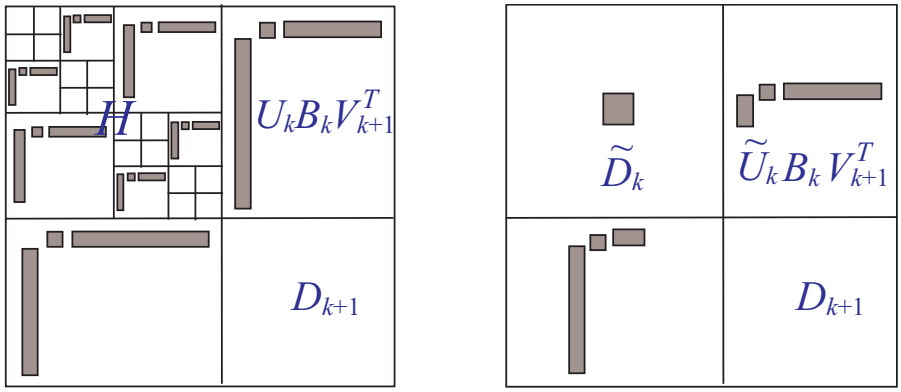
The first three steps are elaborated as follows. First, we construct an HSS approximation to (3.4) as mentioned in section 2. (For convenience, we assume the HSS forms are exact instead of approximate, so as to avoid using additional notation.) An HSS tree $\mathbf{T_i}$ with $k+2$ nodes as in Figure 3.3(i) is used, and root $(\mathbf{T_i})$ has children k and $k+1$. That is, we assume the HSS form of $\mathbf{F_i}$ is

$$\mathbf{F_i} = \begin{pmatrix} H & U_k B_k U_{k+1}^T \\ U_{k+1} B_k^T U_k^T & D_{k+1} \end{pmatrix}, \tag{3.5}$$

where H is an HSS representation for $F_{\mathbf{i},\mathbf{i}}$ that corresponds to the subtree $\mathbf{T_i}[k]$, and $D_{k+1} \equiv F_{\mathcal{N}_\mathbf{i},\mathcal{N}_\mathbf{i}}$. See Figure 3.4(i). For convenience, assume the HSS rank of each $\mathbf{F_i}$ is r.

Second, apply a symmetric ULV HSS factorization method to H (a symmetric version of the original one in [9, 40] or the improved one in section 2.2). This yields (see (3.23) below)

$$H = \mathbf{L_i} \mathbf{L_i}^T. \tag{3.6}$$



(i) $\mathbf{F_i}$ (ii) After the ULV factorization of H

FIG. 3.4. *HSS matrix patterns for* $\mathbf{F_i}$ *before and after the partial factorization (ULV factorization of* H*).*

Then

$$\mathbf{F_i} = \begin{pmatrix} \mathbf{L_i} & \\ U_{k+1}B_k^TU_k^T\mathbf{L_i}^{-T} & I \end{pmatrix} \begin{pmatrix} I & \\ & \mathbf{U_i} \end{pmatrix} \begin{pmatrix} \mathbf{L_i}^T & \mathbf{L_i}^{-1}U_kB_kU_{k+1}^T \\ & I \end{pmatrix}, \tag{3.7}$$

where $\mathbf{U_i}$ is the update matrix

$$\begin{aligned} \mathbf{U_i} &= D_{k+1} - \left(U_{k+1}B_k^TU_k^T\right) H^{-1} \left(U_kB_kU_{k+1}^T\right) \\ &= D_{k+1} - \left(\left(\mathbf{L_i}^{-1}U_k\right) B_kU_{k+1}^T\right)^T \left(\left(\mathbf{L_i}^{-1}U_k\right) B_kU_{k+1}^T\right). \end{aligned} \tag{3.8}$$

$\mathbf{U_i}$ is formed quickly in the next step, not directly with (3.8). After the ULV factorization of H, its HSS tree $\mathbf{T_i}[k]$ is reduced to a single node k with the associated generators $\tilde{D}_k$ and $\tilde{U}_k$ as in Figures 2.4(iii) and 3.3(ii). That is, the HSS form of $\mathbf{F_i}$ is converted into a reduced matrix

$$\begin{pmatrix} \tilde{D}_k & \tilde{U}_kB_kU_{k+1}^T \\ U_{k+1}B_k^T\tilde{U}_k^T & D_{k+1} \end{pmatrix}. \tag{3.9}$$

The procedure is illustrated in 3.4.

Finally, consider the fast computation of $\mathbf{U}_i$. We emphasize that $\mathbf{L_i}$ is not explicitly available. We do not need to directly apply $\mathbf{L_i}^{-1}$ to $U_kB_kU_{k+1}^T$ as in (3.7), neither do we compute $\mathbf{U_i}$ by fully forming H^{-1} as in (3.8). The first issue is actually addressed in the solution stage in section 3.4 and Theorem 3.3 below, and the second issue is elaborated as follows. That is, although the ULV factorization $H = \mathbf{L_iL_i}^T$ is not a classical triangular factorization, we can still compute its Schur complement $\mathbf{U_i}$ conveniently. Using the notation in section 2.2, we have the following theorem.

THEOREM 3.1. *Assume a symmetric ULV HSS factorization* (3.6) *for* H *is computed, and* $\tilde{D}_k$ *and* $\tilde{U}_k$ *are the generators in the reduced matrix* (3.9)*. Then*

$$U_k^TH^{-1}U_k = \tilde{U}_k^T\tilde{D}_k^{-1}\tilde{U}_k. \tag{3.10}$$

Therefore,

$$\mathbf{U_i} = D_{k+1} - \Theta_k^T\Theta_k \text{ with } \Theta_k = \left(L_k^{-1}\tilde{U}_k\right) B_kU_{k+1}^T, \tag{3.11}$$

where $\tilde{D}_k = L_kL_k^T$ *is the Cholesky factorization of* $\tilde{D}_k$*.*

Proof. We consider the improved ULV factorization in section 2.2 only. Assume $\mathbf{T_i}[k]$ (the HSS tree of H) has $l_{\max}$ levels with the root k at level 0 and the leaves at level $l_{\max}$. For $l = l_{\max}, l_{\max}-1, \ldots, 1, 0$, let $H^{(l)}$ be the reduced matrix at level l in the ULV factorization. That is, let $H^{(l_{\max})} \equiv H$, and $H^{(l)}$ be the reduced matrix obtained from $H^{(l+1)}$ after the elimination of all nodes at level $l+1$. This is formulated as follows. For convenience, we denote the children of a node j of $\mathbf{T_i}[k]$ by $c_{j,1}$ and $c_{j,2}$.

Assume the notation in section 2.2 is specifically used for H. Noting (2.4) and (2.6), we define

$$
\begin{aligned}
&U^{(l)} = \operatorname{diag}(\tilde{U}_{j_1}, \ldots, \tilde{U}_{j_\alpha}),\ l = l_{\max}, l_{\max}-1, \ldots, 1,\ U^{(0)} \equiv \tilde{U}_k,\\
&R^{(l)} = \operatorname{diag}\left(\begin{pmatrix} R_{c_{j_1,1}} \\ R_{c_{j_1,2}} \end{pmatrix} R_{j_1}, \ldots, \begin{pmatrix} R_{c_{j_\alpha,1}} \\ R_{c_{j_\alpha,2}} \end{pmatrix} R_{j_\alpha}\right),\ l = l_{\max}-1, l_{\max}-2, \ldots, 1,\ R^{(0)} \equiv I,\\
&X^{(l)} = \operatorname{diag}\left(\begin{pmatrix} Q^T_{c_{j_1,1}} L^{-1}_{c_{j_1,1}} & \\ & Q^T_{c_{j_1,2}} L^{-1}_{c_{j_1,2}} \end{pmatrix}, \ldots, \begin{pmatrix} Q^T_{c_{j_\alpha,1}} L^{-1}_{c_{j_\alpha,1}} & \\ & Q^T_{c_{j_\alpha,2}} L^{-1}_{c_{j_\alpha,2}} \end{pmatrix}\right),\\
&\quad l = l_{\max}-1, l_{\max}-2, \ldots, 0,
\end{aligned}
$$

where $j_1, j_2, \ldots, j_\alpha$ are the nodes at level l of $\mathbf{T_i}[k]$. Clearly, the hierarchical structure of the HSS form of H means

$$
U_k = U^{(l_{\max})}\left(R^{(l_{\max}-1)} R^{(l_{\max}-2)} \cdots R^{(0)}\right) \equiv U^{(l_{\max})} \prod_{l=l_{\max}-1}^{0} R^{(l)}. \tag{3.12}
$$

Also let Ω_l be a permutation matrix during the elimination at level $l+1$ which performs all the merging steps on $H^{(l+1)}$ to form $H^{(l)}$. Then the ULV factorization process can be recursively represented by

$$
\begin{pmatrix} 0 \\ U^{(l)} \end{pmatrix} = \Omega^{(l)} X^{(l)} U^{(l+1)} R^{(l)} \qquad \text{(Figure 2.3(i)–(ii))}, \tag{3.13}
$$

$$
\begin{pmatrix} I & \\ & H^{(l)} \end{pmatrix} = \Omega^{(l)} X^{(l)} H^{(l+1)} (\Omega^{(l)} X^{(l)})^T \qquad \text{(Figure 2.3(iii)–(iv))}, \tag{3.14}
$$

$$
l = l_{\max}-1, l_{\max}-2, \ldots, 1, 0.
$$

For $l = l_{\max}-1, l_{\max}-2, \ldots, 0$, we have the recursive relationship

$$
\begin{aligned}
&(R^{(l)})^T \left[\left(U^{(l+1)}\right)^T \left(H^{(l+1)}\right)^{-1} U^{(l+1)}\right] R^{(l)} \quad \text{(equation (3.14))}\\
&= (U^{(l+1)} R^{(l)})^T \left[\left(\Omega^{(l)} X^{(l)}\right)^{-1} \begin{pmatrix} I & \\ & H^{(l)} \end{pmatrix} (\Omega^{(l)} X^{(l)})^{-T}\right]^{-1} (U^{(l+1)} R^{(l)})\\
&= (\Omega^{(l)} X^{(l)} U^{(l+1)} R^{(l)})^T \begin{pmatrix} I & \\ & \left(H^{(l)}\right)^{-1} \end{pmatrix} (\Omega^{(l)} X^{(l)} U^{(l+1)} R^{(l)}) \quad \text{(equation (3.13))}\\
&= \begin{pmatrix} 0 & \left(U^{(l)}\right)^T \end{pmatrix} \begin{pmatrix} I & \\ & \left(H^{(l)}\right)^{-1} \end{pmatrix} \begin{pmatrix} 0 \\ U^{(l)} \end{pmatrix}\\
&= (U^{(l)})^T (H^{(l)})^{-1} U^{(l)}.
\end{aligned}
$$

Thus,

$$\begin{aligned}
\tilde{U}_k^T \tilde{D}_k^{-1} \tilde{U}_k &= \left(U^{(0)}\right)^T \left(H^{(0)}\right)^{-1} U^{(0)} \\
&= (R^{(0)})^T \left[\left(U^{(1)}\right)^T \left(H^{(1)}\right)^{-1} U^{(1)}\right] R^{(0)} \\
&= \cdots \\
&= \left(\prod_{l=l_{\max}-1}^{0} R^{(l)}\right)^T \left[\left(U^{(l_{\max})}\right)^T \left(H^{(l_{\max})}\right)^{-1} U^{(l_{\max})}\right] \left(\prod_{l=l_{\max}-1}^{0} R^{(l)}\right) \\
&= \left(U^{(l_{\max})} \prod_{l=l_{\max}-1}^{0} R^{(l)}\right)^T H^{-1} \left(U^{(l_{\max})} \prod_{l=l_{\max}-1}^{0} R^{(l)}\right) \text{ (equation (3.12))} \\
&= U_k^T H^{-1} U_k.
\end{aligned}$$

Then (3.10) holds. Equation (3.11) follows from (3.8) and (3.10). ∎

Remark 3.1. Note that $\tilde{D}_k$ is the final reduced matrix after the ULV factorization of H. This theorem indicates that, in the computation of $\mathbf{U_i}$, the roles of H and U_k can be replaced by those of $\tilde{D}_k$ and $\tilde{U}_k$, respectively. Thus, $\mathbf{U}_i$ can be computed quickly with a low-rank update in (3.11), since $\tilde{D}_k$ is a much smaller matrix with size equal to the HSS rank of H. In fact, if H has size N and HSS rank r, then the computation of $\mathbf{L_i}^{-1} U_k$ in (3.8) costs $O\left(r^2 N\right)$ flops, while the computation of $L_k^{-1}\tilde{U}_k$ in (3.11) costs only $O(r^3)$.

For the nonsymmetric case, we can similarly prove the following result.

COROLLARY 3.2. *Assume H and its ULV HSS factorization in Theorem* 3.1 *are nonsymmetric. Then*

$$V_k^T H^{-1} U_k = \tilde{V}_k^T \tilde{D}_k^{-1} \tilde{U}_k.$$

(See (2.3) *with j set to be k.) Therefore,*

(3.15)
$$\mathbf{U}_i = D_{k+1} - \Theta_k^T \Phi_k, \quad \textit{with} \quad \Theta_k = \left(T_k^{-T} \tilde{V}_k\right) B_{k+1} U_{k+1}^T, \ \Phi_k = \left(L_k^{-1} \tilde{U}_k\right) B_k V_{k+1}^T,$$

where $\tilde{D}_k = L_k T_k$ is the LU factorization of $\tilde{D}_k$.

After the computation of $\mathbf{U}_i$, it participates in the standard extend-add operation to form upper-level frontal matrices. This process then proceeds along the elimination tree. When the process finishes, we have a structured sparse factorization

$$A = \mathcal{L}\mathcal{L}^T. \tag{3.16}$$

If rank-revealing factorizations are used in the intermediate HSS constructions, $\mathcal{L}\mathcal{L}^T$ approximates A.

Remark 3.2. The structure of $\mathcal{L}$ from the above partially structured scheme (with dense $\mathbf{U}_i$) is the same as that produced by a fully structured version (with HSS $\mathbf{U}_i$). According to the discussions in section 4, the costs of these two schemes are also similar. However, by keeping $\mathbf{U}_i$ dense, we significantly reduce the complication of the extend-add operation. The idea of reduced matrices in Theorem 3.1 and Corollary 3.2 is also very useful in future developments of fully structured versions, where the computation of $\mathbf{L_i}^{-1} U_k$ in (3.8) dominates the cost.

3.4. Structured multifrontal solution. After the factorization (3.16), we solve two structured systems

$$\mathcal{L}\mathbf{y} = \mathbf{b}, \tag{3.17}$$

$$\mathcal{L}^T\mathbf{x} = \mathbf{y}. \tag{3.18}$$

For convenience, partition the vectors conformably into $\mathbf{b_i}$, $\mathbf{y_i}$, and $\mathbf{x_i}$ pieces according to the sizes of the separators in the nest dissection, so that $\mathbf{b_i}$, $\mathbf{y_i}$, and $\mathbf{x_i}$ correspond to the variables associated with separator $\mathbf{i}$ (node $\mathbf{i}$ in the assembly tree $\mathcal{T}$).

The solution of (3.17) with forward substitution involves a forward (or postordering) traversal of the assembly tree. We show the intermediate structured solution steps associated with a node $\mathbf{i}$. According to (3.7), we need to solve intermediate systems that look like

$$\begin{pmatrix} \mathbf{L_i} & \\ U_{k+1}B_k^T U_k^T \mathbf{L_i}^{-T} & I \end{pmatrix} \begin{pmatrix} \mathbf{y_i} \\ \tilde{\mathbf{b}}_{\mathcal{N}_\mathbf{i}} \end{pmatrix} = \begin{pmatrix} \mathbf{b_i} \\ \mathbf{b}_{\mathcal{N}_\mathbf{i}} \end{pmatrix},$$

where $\mathbf{b}_{\mathcal{N}_\mathbf{i}}$ corresponds to the union of $\mathbf{b_j}$ for $\mathbf{j} \in \mathcal{N}_\mathbf{i}$ (the set of neighbors of $\mathbf{i}$). Here, $\mathbf{b_i}$ may have been updated in the previous solution steps associated with the lower levels of $\mathcal{T}$ (see (3.19)). (We still use $\mathbf{b_i}$ for notational convenience.)

We first solve $\mathbf{L_i y_i} = \mathbf{b_i}$ with an ULV-type forward-substitution method in section 2.3. Then $U_{k+1}B_{k+1}U_k^T\mathbf{L_i}^{-T}\mathbf{y_i}$ is the contribution of separator $\mathbf{i}$ to its neighbors. That is, we update $\mathbf{b}_{\mathcal{N}_\mathbf{i}}$ by

$$\mathbf{b}_{\mathcal{N}_\mathbf{i}} \leftarrow \tilde{\mathbf{b}}_{\mathcal{N}_\mathbf{i}} = \mathbf{b}_{\mathcal{N}_\mathbf{i}} - U_{k+1}B_{k+1}U_k^T\mathbf{L}_i^{-T}\mathbf{y_i}. \tag{3.19}$$

Notice that the computation cost of (3.19) can be significantly reduced based on the idea of reduced matrices similar to Theorem 3.1.

THEOREM 3.3. *For node* $\mathbf{i}$ *in the assembly tree* $\mathcal{T}$*, denote* y_j *in* (2.10) *by* $\mathbf{y}_{\mathbf{i},j}$*, and* y_k *in* (2.11) *by* $\mathbf{y}_{\mathbf{i},k}$*, which are the solution pieces obtained by the forward-substitution procedure in section* 2.3 *applied to* $\mathbf{L_i y_i} = \mathbf{b_i}$*. Assume the same condition as in Theorem* 3.1 *holds. Then*

$$U_k^T\mathbf{L_i}^{-T}\mathbf{y_i} = \tilde{U}_k^T L_k^{-T}\mathbf{y}_{\mathbf{i},k}. \tag{3.20}$$

Therefore, (3.19) *can be computed as*

$$\tilde{\mathbf{b}}_{\mathcal{N}_\mathbf{i}} = \mathbf{b}_{\mathcal{N}_\mathbf{i}} - \Theta_k^T\mathbf{y}_{\mathbf{i},k}, \tag{3.21}$$

where Θ_k *is available from* (3.11).

Proof. Consider $H^{(l)}$ in the proof of Theorem 3.1. Assume $H^{(0)} = \mathbf{L}^{(0)}(\mathbf{L}^{(0)})^T$ is the Cholesky factorization of $H^{(0)} \equiv \tilde{D}_k$. Then according to (3.14), we have

$$H^{(l)} = \mathbf{L}^{(l)}(\mathbf{L}^{(l)})^T,$$

where $\mathbf{L}^{(l)}$ is recursively defined as

$$\mathbf{L}^{(l+1)} = (X^{(l)})^{-1}(\Omega^{(l)})^T \begin{pmatrix} I & \\ & \mathbf{L}^{(l)} \end{pmatrix} \Omega^{(l)},\ l = l_{\max}-1, l_{\max}-2, \ldots, 1, 0. \tag{3.22}$$

This gives the actual form of the ULV factorization

$$H = \mathbf{L_i}\mathbf{L_i}^T, \quad \mathbf{L_i} \equiv \mathbf{L}^{(l_{\max})}. \tag{3.23}$$

Also, assume the solution pieces $y_j \equiv \mathbf{y_i}_{,j}$ in (2.10) for all nodes j at each level l of $\mathbf{T}[k]$ form a vector $\bar{y}^{(l)}$ (Figure 2.5). That is, let

$$\bar{y}^{(l)} = \begin{pmatrix} y_{j_1}^T & \cdots & y_{j_\alpha}^T \end{pmatrix}^T \quad (j_1, \ldots, j_\alpha\text{: all nodes at level } l \text{ of } \mathbf{T}[k]), \tag{3.24}$$

$$y^{(0)} = y_k, \quad y^{(l+1)} = (\Omega^{(l)})^T \begin{pmatrix} \bar{y}^{(l+1)} \\ y^{(l)} \end{pmatrix}, \ l = 0, 1, \ldots, l_{\max} - 1. \tag{3.25}$$

Then it can be verified that

$$\mathbf{y_i} \equiv y^{(l_{\max})}. \tag{3.26}$$

According to (3.22) and (3.25), we have the following recursive relationship for $l = l_{\max} - 1, l_{\max} - 2, \ldots, 0$:

$$\begin{aligned}
&(R^{(l)})^T[(U^{(l+1)})^T(L^{(l+1)})^{-T}y^{(l+1)}] \\
&= (R^{(l)})^T \left[(U^{(l+1)})^T \left((X^{(l)})^{-1}(\Omega^{(l)})^T \begin{pmatrix} I & \\ & \mathbf{L}^{(l)} \end{pmatrix} \Omega^{(l)}\right)^{-T} (\Omega^{(l)})^T \begin{pmatrix} \bar{y}^{(l+1)} \\ y^{(l)} \end{pmatrix}\right] \\
&= (\Omega^{(l)} X^{(l)} U^{(l+1)} R^{(l)})^T \begin{pmatrix} I & \\ & (\mathbf{L}^{(l)})^{-T} \end{pmatrix} \begin{pmatrix} \bar{y}^{(l+1)} \\ y^{(l)} \end{pmatrix} \\
&= \begin{pmatrix} 0 & (U^{(l)})^T \end{pmatrix} \begin{pmatrix} I & \\ & (\mathbf{L}^{(l)})^{-T} \end{pmatrix} \begin{pmatrix} \bar{y}^{(l+1)} \\ y^{(l)} \end{pmatrix} \quad \text{(equation (3.13))} \\
&= (U^{(l)})^T(\mathbf{L}^{(l)})^{-T}y^{(l)}.
\end{aligned}$$

Therefore,

$$\begin{aligned}
\tilde{U}_k^T L_k^{-T} \mathbf{y_i}_{,k} &= (U^{(0)})^T(\mathbf{L}^{(0)})^{-T}y^{(0)} \\
&= (R^{(0)})^T[(U^{(1)})^T(L^{(1)})^{-T}y^{(1)}] \\
&= \cdots \\
&= \left(\prod_{l=l_{\max}-1}^{0} R^{(l)}\right)^T \left[\left(U^{(l_{\max})}\right)^T (\mathbf{L}^{(l_{\max})})^{-T} y^{(l_{\max})}\right] \\
&= \left(U^{(l_{\max})} \prod_{l=l_{\max}-1}^{0} R^{(l)}\right)^T (\mathbf{L}^{(l_{\max})})^{-T} y^{(l_{\max})} \\
&= U_k^T \mathbf{L_i}^{-T} \mathbf{y_i},
\end{aligned}$$

where equations (3.12), (3.23), and (3.26) are used. Then, (3.19) and (3.20) lead to (3.21). ∎

Again, since L_k is generally a much smaller triangular matrix, the computation with (3.21) is much faster than with (3.19). A direct solution of $\mathbf{L_i}^{-T}\mathbf{y_i}$ needs $O(rN)$ flops if H has size N and HSS rank r, while $L_k^{-T}y_k$ only costs $O(r^2)$. Such a savings is even more significant in a fully structured version (Remark 3.2).

In the backward-substitution stage, we solve intermediate systems of the following form for $\mathbf{x_i}$:

$$\begin{pmatrix} \mathbf{L_i}^T & \mathbf{L_i}^{-1}U_k B_k U_{k+1}^T \\ & I \end{pmatrix} \begin{pmatrix} \mathbf{x_i} \\ \mathbf{x}_{\mathcal{N}_\mathbf{i}} \end{pmatrix} = \begin{pmatrix} \mathbf{y_i} \\ \mathbf{x}_{\mathcal{N}_\mathbf{i}} \end{pmatrix},$$

where $\mathbf{x}_{\mathcal{N}_i}$ is already available from solution steps associated with upper-level separators. That is, we actually solve

$$\mathbf{L}_\mathbf{i}^T \mathbf{x}_\mathbf{i} = \tilde{\mathbf{y}}_\mathbf{i} \quad \text{with} \quad \tilde{\mathbf{y}}_\mathbf{i} = \mathbf{y}_\mathbf{i} - \mathbf{L}_\mathbf{i}^{-1} U_k B_k U_{k+1}^T \mathbf{x}_{\mathcal{N}_\mathbf{i}}.$$

This involves a ULV-type backward substitution in section 2.3. Similar to Theorem 3.3, we can prove that $\tilde{\mathbf{y}}_\mathbf{i}$ can be computed quickly as follows.

THEOREM 3.4. *Let* $\mathbf{y}_{\mathbf{i},j}$ *and* $\mathbf{y}_{\mathbf{i},k}$ *be given as in Theorem* 3.3. *The computation of* $\tilde{\mathbf{y}}_\mathbf{i} = \mathbf{y}_\mathbf{i} - \mathbf{L}_\mathbf{i}^{-1} U_k B_k U_{k+1}^T \mathbf{x}_{\mathcal{N}_\mathbf{i}}$ *can be done by updating only the piece* y_k *by*

$$y_k \leftarrow y_k - \Theta_k \mathbf{x}_{\mathcal{N}_\mathbf{i}}, \tag{3.27}$$

where Θ_k *is available from* (3.11). *Then merging all* y_j *pieces as in* (3.24)–(3.26) *to update* $\mathbf{y}_\mathbf{i}$.

For nonsymmetric problems, similar results can be shown.

COROLLARY 3.5. *Assume the conditions in Corollary* 3.2 *hold, and* (3.16) *is replaced by a nonsymmetric structured factorization* $A = \mathcal{LU}$. *Then in the solutions of* $\mathcal{L}\mathbf{y} = \mathbf{b}$ *and* $\mathcal{U}\mathbf{x} = \mathbf{y}$, (3.21) *and* (3.27) *are replaced by*

$$\tilde{\mathbf{b}}_{\mathcal{N}_\mathbf{i}} = \mathbf{b}_{\mathcal{N}_\mathbf{i}} - \Theta_k^T \mathbf{y}_{\mathbf{i},k}, \quad \textit{and} \quad y_k \leftarrow y_k - \Phi_k \mathbf{x}_{\mathcal{N}_\mathbf{i}},$$

respectively, where Θ_k *and* Φ_k *are available from* (3.15).

4. Algorithms and performance analysis with sparse rank relaxation. Before presenting the algorithms, we point out the major significance of the method, especially as compared with the method in [39]:

(1) The new method is designed to handle general sparse matrices with general nested dissection for the mesh (which may be irregular or in three dimensions) or the adjacency graph. The method in [39] focuses on 2D regular meshes since it uses the coordinates of the uniform mesh points for the ordering. Unlike [39], we do not require the sparse matrix to be SPD. For example, Corollaries 3.2 and 3.5 show how to apply the concept of reduced matrices to nonsymmetric factorizations and solutions, respectively.

(2) We present some improved HSS algorithms and propose the concept of reduced matrices for the intermediate HSS operations. That is, we avoid using direct HSS inversions and ULV solutions needed in [39], and replace them by simple operations on the small reduced matrices resulting from ULV factorizations. The concept is applicable to different types of ULV factorizations. See Theorems 3.1, 3.3, and 3.4. Detailed proofs are given to justify the concept. This idea is useful in reducing the hidden constant in the complexity of structured factorizations. See Remarks 3.1 and 3.2.

(3) Our method is much simpler to use. One reason is the replacement of HSS inversions and solutions by those on the reduced matrices. Another reason is that we avoid complex structured extend-add operations by using dense update matrices. This is suitable for arbitrary separator connectivities and helps enhance the flexibility of the methods. The method in [39] is very complex and difficult to implement, and cannot be easily extended to general matrices.

(4) We show an idea of sparse rank relaxation, so that the off-diagonal ranks of the intermediate Schur complements do not have to remain bounded by a small constant as in [39]. We give some relaxed rank patterns in practical

problems, so that the method yields about $O(n)$ flops in two dimensions and about $O(n^{4/3})$ in three dimensions, where n is the order of the sparse discretized matrix A. See section 4.2. For both two and three dimensions, the memory size and solution costs are nearly linear in n. The method in [39] only considers a special 2D case.

(5) In practice, the structured factorization only starts from a certain switching level $\mathbf{l}_s$ of the assembly tree $\mathcal{T}$, not only to avoid small block operations, but also to achieve nearly optimal complexity. This is similar to the method in [39]. However, we use different optimization criteria to choose $\mathbf{l}_s$ for 3D problems. See Theorems 4.1 and 4.3 below.

The structured factorization and solution algorithms are summarized as follows. For convenience, A is assumed to be SPD. Generalization to the nonsymmetric case can be considered similarly.

ALGORITHM 1. STRUCTURED SPARSE FACTORIZATION.

Apply general nested dissection to the mesh or adjacency graph
`for` node/separator $\mathbf{i}$ from 1 to root $(\mathcal{T})$ of the assembly tree $\mathcal{T}$
(1) `if` $\mathbf{i}$ is a leaf, form $\mathbf{F_i} \equiv \mathbf{F_i^0}$ as in (3.3)
(2) `if` $\mathbf{i}$ is at level $\mathbf{l} > \mathbf{l}_s$ of $\mathcal{T}$ ▷ *Exact factorization before* $\mathbf{l}_s$
  (a) Compute the (exact) Cholesky factorization $F_{\mathbf{i},\mathbf{i}} = \mathbf{L_i}\mathbf{L_i}^T$
  (b) Compute $\mathbf{L}_{\mathcal{N}_\mathbf{i},\mathbf{i}} = F_{\mathcal{N}_\mathbf{i},\mathbf{i}}\mathbf{L_i}^{-T}$
  (c) Compute (dense) $\mathbf{U_i} = F_{\mathcal{N}_\mathbf{i},\mathcal{N}_\mathbf{i}} - \mathbf{L}_{\mathcal{N}_\mathbf{i},\mathbf{i}}\mathbf{L}_{\mathcal{N}_\mathbf{i},\mathbf{i}}^T$
  `else` ▷ *Structured factorization after* $\mathbf{l}_s$
  (a) Compute an HSS approximation H to $\mathbf{F_i}$
  (b) Compute an ULV HSS factorization $F_{\mathbf{i},\mathbf{i}} = \mathbf{L_i}\mathbf{L_i}^T$
    ▷ *New ULV factorization in section 2.2 and producing reduced matrices*
  (c) Compute (dense) $\mathbf{U_i} = D_{k+1} - \Theta_k^T\Theta_k$, with $\Theta_k = (L_k^{-1}\tilde{U}_k)B_kU_{k+1}^T$
    ▷ *Dense Schur complement with the aid of the final reduced matrix*
(3) `if` $\mathbf{i}$ is a left node, push $\mathbf{U_i}$ onto the update matrix stack
  `else` ▷ *Exact extend-add operation*
  (a) Pop $\mathbf{U_j}$ from the update matrix stack for $\mathbf{j} = \text{sib}\,(\mathbf{i})$
  (b) Compute $\mathbf{F_p} = \mathbf{F_p^0} \oplus \mathbf{U_j} \oplus \mathbf{U_i}$ for $\mathbf{p} = \text{par}\,(\mathbf{i})$

ALGORITHM 2. STRUCTURED SPARSE SOLUTION.

(1) Partition $\mathbf{b}$ into $\mathbf{b}_i$ pieces according to the sizes of the leaf separators
(2) `for` node/separator $\mathbf{i}$ from 1 to root $(\mathcal{T})$ ▷ *Forward substitution*
  `if` $\mathbf{i}$ is at level $\mathbf{l} > \mathbf{l}_s$ of $\mathcal{T}$ ▷ *Exact solution before* $\mathbf{l}_s$
  (a) Solve a lower-triangular system $\mathbf{L_i}\mathbf{y_i} = \mathbf{b_i}$
  (b) Update $\mathbf{b}_{\mathcal{N}_\mathbf{i}}$ to $\tilde{\mathbf{b}}_{\mathcal{N}_\mathbf{i}} = \mathbf{b}_{\mathcal{N}_\mathbf{i}} - \mathbf{L}_{\mathcal{N}_\mathbf{i},\mathbf{i}}\mathbf{y}_{\mathbf{i},k}$
  `else` ▷ *Structured solution after* $\mathbf{l}_s$
  (a) Solve $\mathbf{L_i}\mathbf{y_i} = \mathbf{b_i}$ with the new ULV forward substitution in section 2.3
  (b) Update $\mathbf{b}_{\mathcal{N}_\mathbf{i}}$ to $\tilde{\mathbf{b}}_{\mathcal{N}_\mathbf{i}}$ in (3.21)
    ▷ *Fast update with the aid of the final reduced matrix*
(3) `for` node/separator $\mathbf{i}$ from root $(\mathcal{T})$ to 1 ▷ *Backward substitution*
  `if` i is at level $\mathbf{l} > \mathbf{l}_s$ of $\mathcal{T}$ ▷ *Exact solution before* $\mathbf{l}_s$
  (a) Solve an upper-triangular system $\mathbf{L_i}^T\mathbf{x_i} = \mathbf{y}_i$
  (b) `for` all nodes $\mathbf{j}$ such that $\mathbf{i} \in \mathcal{N}_\mathbf{j}$, update $\mathbf{y_j}$ to $\tilde{\mathbf{y}}_\mathbf{j} = \mathbf{y_j} - \mathbf{L}_{\mathbf{j},\mathbf{i}}\mathbf{y_i}$
  `else` ▷ *Structured solution after* $\mathbf{l}_s$
  (a) Solve $\mathbf{L_i}^T\mathbf{y_i} = \mathbf{b_i}$ with the new ULV backward substitution in section 2.3
  (b) `for` all nodes $\mathbf{j}$ such that $\mathbf{i} \in \mathcal{N}_\mathbf{j}$, update the piece y_k of $\mathbf{y_j}$ to $\tilde{y}_k$ in (3.27)
    ▷ *Fast update with the aid of the final reduced matrix*

TABLE 4.1
Factorization cost ξ_{fact}, solution cost ξ_{sol}, and storage σ_{mem} of the structured multifrontal method applied to a discretized matrix A of order n on a regular mesh.

	ξ_{fact}	ξ_{sol}	σ_{mem}
2D	$O(rn\log n)$	$O(n\log r)+O(n\log\log n)$	$O(n\log r)+O(n\log\log n)$
3D	$O(rn^{4/3})$	$O(r^{1/2}n)$	$O(r^{1/2}n)$

The algorithms and their variations can be applied to general sparse matrices. For simplicity, we only consider the complexity in terms of sparse matrices arising from the discretizations of 2D and 3D PDEs.

4.1. Fixed-rank complexity analysis. We first consider the usual case where the HSS ranks of all the frontal matrices are bounded by r, and then relax this requirement to get more flexible results.

THEOREM 4.1 (complexity optimization strategies). *Assume Algorithms* 1 *and* 2 *are applied to a sparse matrix A, and the HSS ranks of all the frontal matrices $\mathbf{F_l}$ in Algorithm* 1 *are bounded by r. Denote the structured multifrontal factorization cost, solution cost, and memory size by ξ_{fact}, ξ_{sol}, and σ_{mem}, respectively. Let* root $(\mathcal{T})$ *be at level* 0 *and the leaves of $\mathcal{T}$ be at level $\mathbf{l}_{\max}$. The switching level $\mathbf{l}_s$ $(0\le \mathbf{l}_s\le \mathbf{l}_{\max})$ is chosen to obtain optimal complexity as follows:*

- *If A is obtained from a 2D $N\times N$ mesh and $n=N^2$, the counts are given in row* 2 *of Table* 4.1. *The switching level $\mathbf{l}_s$ satisfies $\mathbf{l}_{\max}-\mathbf{l}_s=O(\log N)$, so that the* factorization costs *before and after the switching level are the same.*
- *If A is obtained from a 3D $N\times N\times N$ mesh and $n=N^3$, the results are given in row* 3 *of Table* 4.1. *The switching level $\mathbf{l}_s$ satisfies $\mathbf{l}_{\max}-\mathbf{l}_s=O(\log N)$, so that the* solution costs *before and after the switching level are the same.*

Proof. We briefly sketch the proof. For the 2D case, with a process similar to the proof in [39, Theorem 4.2], we can obtain the total cost for the factorization algorithm:

$$\xi_{\text{fact}}=\underbrace{\sum_{\mathbf{l}=\mathbf{l}_s+1}^{\mathbf{l}_{\max}}4^{\mathbf{l}}O\left(\left(\frac{N}{2^{\mathbf{l}}}\right)^3\right)}_{\text{before the switching level}}+\underbrace{\sum_{\mathbf{l}=0}^{\mathbf{l}_s}4^{\mathbf{l}}O\left(r\left(\frac{N}{2^{\mathbf{l}}}\right)^2\right)}_{\text{after the switching level}}=O\left(\frac{N^3}{2^{\mathbf{l}_s}}\right)+O(rN^2\mathbf{l}_s). \tag{4.1}$$

Assume N and $\mathbf{l}_{\max}$ are sufficiently large. ξ_{fact} is optimized when the costs before and after the switch level $\mathbf{l}_s$ are equal. That is,

$$O\left(\frac{N^3}{2^{\mathbf{l}_s}}\right)=O(rN^2\mathbf{l}_s)\ \text{ or }\ \mathbf{l}_s=\mathbf{l}_{\max}-O(\log r)-O(\log\mathbf{l}_s)=O(\log N). \tag{4.2}$$

Then $\xi_{\text{fact}}=O(rn\log n)$. In such a situation, the solution cost is

$$\begin{aligned}\xi_{\text{sol}}&=\sum_{\mathbf{l}=\mathbf{l}_s+1}^{\mathbf{l}_{\max}}4^{\mathbf{l}}O\left(\left(\frac{N}{2^{\mathbf{l}}}\right)^2\right)+\sum_{\mathbf{l}=0}^{\mathbf{l}_s}4^{\mathbf{l}}O\left(r\frac{N}{2^{\mathbf{l}}}\right)=O(n(\mathbf{l}_{\max}-\mathbf{l}_s))+O(r2^{\mathbf{l}_s}N)\\&=O(n\log r)+O(n\log\log n),\end{aligned}$$

where (4.2) is used. The memory size σ_{mem} can be similarly estimated.

For three dimensions we similarly have

$$\xi_{\text{fact}} = \sum_{\mathbf{l}=\mathbf{l}_s+1}^{\mathbf{l}_{\max}} 8^{\mathbf{l}} O\left(\left(\frac{N}{2^{\mathbf{l}}}\right)^6\right) + \sum_{\mathbf{l}=0}^{\mathbf{l}_s} 8^{\mathbf{l}} O\left(r\left(\frac{N}{2^{\mathbf{l}}}\right)^4\right) = O\left(n^2\left(\frac{1}{8}\right)^{l_s}\right) + O(rn^{4/3}),$$

$$\xi_{\text{sol}} = \sum_{\mathbf{l}=\mathbf{l}_s+1}^{\mathbf{l}_{\max}} 8^{\mathbf{l}} O\left(\left(\frac{N}{2^{\mathbf{l}}}\right)^4\right) + \sum_{\mathbf{l}=0}^{\mathbf{l}_s} 8^{\mathbf{l}} O\left(r\left(\frac{N}{2^{\mathbf{l}}}\right)^2\right) = O\left(n^{4/3}\frac{1}{2^{\mathbf{l}_s}}\right) + O(rn^{2/3}2^{\mathbf{l}_s}).$$

Here, if we minimize ξ_{fact}, we get $\xi_{\text{fact}} = O(rn^{4/3})$ and $\xi_{\text{sol}} = O(rn)$. This can be improved by minimizing ξ_{sol} instead. The optimality condition is

$$2^{\mathbf{l}_s} = O(N/r^{1/2}). \tag{4.3}$$

Then we get the results in row 3 of Table 4.1. ∎

According to this theorem, when r is small, our factorization method has performance close to the fully structured version (Remark 3.2). The solution costs and memory requirements of the two versions are in about the same orders in both 2D and 3D cases.

For 2D discrete elliptic equations, it is known that $r = O(1)$ [8]. Thus, our structured algorithms have nearly linear complexity. This similarly holds for 2D Helmholtz equations where the rank bound is $r = O(\log n)$ under certain assumptions [14]. For three dimensions, the bound is $r = O(n^{1/3})$ [8] and Theorem 4.1 indicates $\xi_{\text{fact}} = O(n^{5/3})$, $\xi_{\text{sol}} = O(n^{7/6})$. However, these bounds can be significantly improved as follows.

4.2. Sparse rank relaxation. The estimates above are useful when the HSS rank bound r is small. When r is large or depends on the HSS matrix size, the analysis can highly overestimate the actual costs. In fact, the numerical ranks of the individual HSS blocks at different levels of the HSS tree can be allowed to increase along the level, so that even if r is large, we can still achieve satisfactory complexity. This is explained as follows.

LEMMA 4.2 ([42] dense rank relaxation). *Suppose F is an $N \times N$ dense matrix, and $\mathbf{T}$ is a perfect binary tree with $l_{\max} = O(\log N)$ levels. Partition F into $O(\log N)$ levels of HSS blocks following $\mathbf{T}$ so that the HSS block rows corresponding to the nodes at level l of $\mathbf{T}$ have row dimensions $N_l \equiv O(N/2^l)$ and maximum numerical rank r_l. Then for an r_l value, the costs for the construction of an HSS form for F, its ULV factorization, and the ULV solution are $\tilde{\xi}_{\text{constr}}$, $\tilde{\xi}_{\text{fact}}$, and $\tilde{\xi}_{\text{sol}}$ flops, respectively, and the memory size is $\tilde{\sigma}_{\text{mem}}$, as shown in Table* 4.2.

We call each r_l in Table 4.2 a *rank pattern*. Such rank patterns have been observed in various practical problems, although an analytical proof is not yet available. As an example, Figure 4.1 shows r_l for two dense frontal matrices in the multifrontal factorization of a 3D discretized Helmholtz equation. We observe that, for each dense frontal matrix, r_l is roughly $O(N_l^{1/2})$.

Based on this lemma, we have the following result, whose earlier variations can be found in [37] and the report [41].

THEOREM 4.3 (sparse rank relaxation). *Assume A is a discretized matrix of order n. Suppose each order N frontal matrix $\mathbf{F_i}$ satisfies the condition of F in Lemma* 4.2. *Let the costs of Algorithms* 1–2 *applied to A be ξ_{fact} and ξ_{sol} flops, respectively, and the memory size be σ_{mem}. Then if r_l satisfies the patterns as in Lemma* 4.2, *we have the following results:*

TABLE 4.2

Costs and storage of HSS construction, ULV factorization, and ULV solution algorithms with rank relaxation, where $p \in \mathbb{N}$, $\alpha > 0$, and $r = \max r_l$ is the HSS rank of H.

r_l		$r = \max r_l$	$\tilde{\xi}_{\text{constr}}$	$\tilde{\xi}_{\text{fact}}$	$\tilde{\xi}_{\text{sol}}$	$\tilde{\sigma}_{\text{mem}}$
$O(1)$		$O(1)$	$O(N^2)$	$O(N)$	$O(N)$	$O(N)$
$O((\log N_l)^p)$		$O((\log N)^p)$				
$O(N_l^{1/p})$	$p > 3$	$O(N^{1/p})$				
	$p = 3$	$O(N^{1/3})$	$O(N^2)$	$O(N \log N)$		
	$p = 2$	$O(N^{1/2})$	$O(N^2 \log N)$	$O(N^{3/2})$	$O(N \log N)$	$O(N \log N)$
$O(\alpha^{l_{\max}-l} r_0)$	$\alpha < \sqrt[3]{2}$	$< O(N^{1/3})$	$O(N^2)$	$O(N)$	$O(N)$	$O(N)$
	$\alpha = 2^{1/3}$	$O(N^{1/3})$	$O(N^2)$	$O(N \log N)$		
	$2^{1/3} < \alpha < 2^{1/2}$	$< O(N^{1/2})$	$O(N^2)$	$O(N^{\log \alpha^3})$		
	$\alpha = 2^{1/2}$	$O(N^{1/2})$	$O(N^2 \log N)$	$O(N^{3/2})$	$O(N \log N)$	$O(N \log N)$

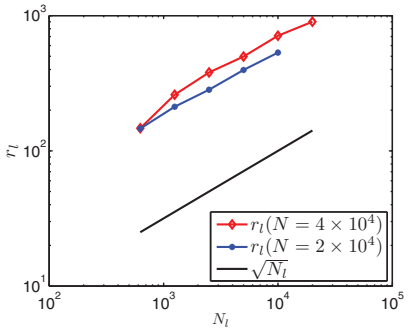


FIG. 4.1. *Numerical rank patterns r_l of the HSS blocks at level l for two dense frontal matrices of sizes $N = 2 \times 10^4$ and 4×10^4, respectively, in the multifrontal factorization of a 3D discretized Helmholtz equation.*

TABLE 4.3

Factorization cost ξ_{fact}, solution cost ξ_{sol}, and storage σ_{mem} of the structured multifrontal method applied to a discretized matrix A of order n on a 2D $N \times N$ mesh, where $p \in \mathbb{N}$ and $\alpha > 0$.

r_l		$r = \max r_l$	ξ_{fact}	ξ_{sol}	σ_{mem}
$O(1)$		$O(1)$	$O(n \log n)$	$O(n \log \log n)$	$O(n \log \log n)$
$O((\log N_l)^p)$		$O((\log N)^p)$			
$O(N_l^{1/p})$	$p \geq 3$	$O(N^{1/p})$			
	$p = 2$	$O(N^{1/2})$	$O(n \log^2 n)$		
$O(\alpha^{l_{\max}-l} r_0)$	$\alpha \leq 2^{1/3}$	$O(N^{1/3})$	$O(n \log n)$		
	$2^{1/3} < \alpha \leq 2^{1/2}$	$O(N^{1/2})$	$O(n \log^2 n)$		

- *If A is obtained from a 2D $N \times N$ mesh and $n = N^2$, the results are given in Table* 4.3. *The switching level $\mathbf{l}_s$ satisfies $\mathbf{l}_{\max} - \mathbf{l}_s = O(\log N)$, so that the* factorization costs *at the levels before and after $\mathbf{l}_s$ are the same.*
- *If A is obtained from a 3D $N \times N \times N$ mesh and $n = N^3$, the results are given in Table* 4.4. *The switching level $\mathbf{l}_s$ satisfies $\mathbf{l}_{\max} - \mathbf{l}_s = O(\log N)$, so that the* solution costs *at the levels before and after $\mathbf{l}_s$ are the same.*

Proof. The proof is similar to that of Theorem 4.1, except that the results in Lemma 4.2 are used to replace some operations. For example, when $r_l = O(N_l^{1/2})$ in

TABLE 4.4

Factorization cost ξ_{fact}, solution cost ξ_{sol}, and storage σ_{mem} of the structured multifrontal method applied to a discretized matrix A of order n on a 3D $N \times N \times N$ mesh, where $p \in \mathbb{N}$ and $\alpha > 0$.

r_l		$r = \max r_l$	ξ_{fact}	ξ_{sol}	σ_{mem}
$O(1)$		$O(1)$	$O(n^{4/3})$	$O(n)$	$O(n)$
$O((\log_2 N_l)^p)$		$O((\log_2 N)^p)$			
$O(N_l^{1/p})$, $p > 3$		$O(N^{1/p})$			
$O(N_l^{1/p})$	$p = 3$	$O(N^{1/3})$	$O(n^{4/3})$	$O(n \log^{1/2} n)$	$O(n \log^{1/2} n)$
	$p = 2$	$O(N^{1/2})$	$O(n^{4/3} \log n)$	$O(n \log n)$	$O(n \log n)$
$O(\alpha^{l_{\max}-l} r_0)$	$\alpha \le 2^{1/3}$	$O(N^{1/3})$	$O(n^{4/3})$	$O(n \log^{1/2} n)$	$O(n \log^{1/2} n)$
	$2^{1/3} < \alpha \le 2^{1/2}$	$O(N^{1/2})$	$O(n^{4/3} \log n)$	$O(n \log n)$	$O(n \log n)$

the 2D case, (4.1) is replaced by

$$\begin{aligned}\xi_{\text{fact}} &= \sum_{\mathbf{l}=\mathbf{l}_s+1}^{\mathbf{l}_{\max}} 4^{\mathbf{l}} O\left(\left(\frac{n^{1/2}}{2^{\mathbf{l}}}\right)^3\right) + \sum_{\mathbf{l}=0}^{\mathbf{l}_s} 4^{\mathbf{l}} O\left(\left(\frac{n^{1/2}}{2^{\mathbf{l}}}\right)^2 \log\left(\frac{n^{1/2}}{2^{\mathbf{l}}}\right)\right) \\ &= \sum_{\mathbf{l}=\mathbf{l}_s+1}^{\mathbf{l}_{\max}} 4^{\mathbf{l}} O\left(\left(\frac{n^{1/2}}{2^{\mathbf{l}}}\right)^3\right) + n \sum_{\mathbf{l}=0}^{\mathbf{l}_s} \left(\frac{1}{2}\log n - l\right) = \frac{n^{3/2}}{2^{\mathbf{l}_s}} + n\mathbf{l}_s(\log n - \mathbf{l}_s).\end{aligned}$$

The minimum is $O(n \log^2 n)$ when $O(n^{1/2}/[\mathbf{l}_s(\log n - \mathbf{l}_s)]) = 2^{\mathbf{l}_s}$. ∎

Remark 4.1. We can see that the HSS rank r of $\mathbf{F_i}$ does not have to be bounded by a small constant. For example, as long as the rank pattern $r_l = O(N_l^{1/2})$ at different hierarchical levels of $\mathbf{F_i}$ holds, A can be factored in about $\xi_{\text{fact}} = O(n^{4/3})$ flops, even though r is as large as $O(N^{1/2})$ and grows with N. Moreover, the solution costs and the storage are nearly linear in n. The discretized 3D Helmholtz equation mentioned in Figure 4.1 satisfies such a condition. On the other hand, the fixed-rank analysis in Theorem 4.1 would give $\xi_{\text{fact}} = O(n^{1/3} \cdot n^{4/3}) = O(n^{5/3})$, which overestimates the cost.

Remark 4.2. If the solver is used as a preconditioner, we can set a relatively large tolerance in the compression. This saves the costs of the structured factorizations after the switching level $\mathbf{l}_s$. According to the complexity optimization strategies, this also indicates that the number of structured factorization levels in $\mathcal{T}$ can be increased, so as to further improve the efficiency of the factorization.

5. Numerical experiments. We demonstrate the performance of our algorithms with some examples. In addition to the notation in section 4.2, we also use the following:

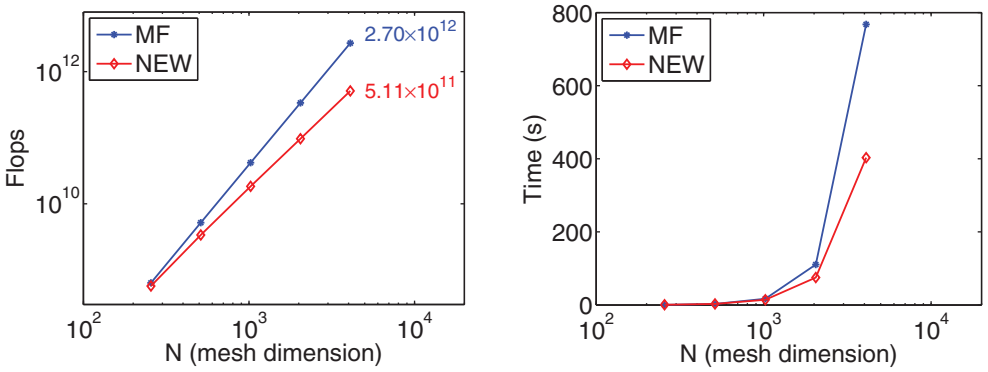
- `NEW`: The new structured multifrontal factorization and solution.
- `MF`: The exact multifrontal method.
- $e_2 = \frac{\|x-\tilde{x}\|_2}{\|x\|_2}$, $\gamma_2 = \frac{\|A\tilde{x}-b\|_2}{\|b\|_2}$, where x and $\tilde{x}$ are the exact and approximation solutions of $Ax = b$, respectively.

Example 1. We consider a Helmholtz equation

$$[-\Delta + \omega^2 c(\mathbf{x})^{-2}]\mathbf{u} = \mathbf{f}, \tag{5.1}$$

where ω is the angular frequency, $c(x)$ is the velocity field, and $\mathbf{f}$ is the forcing term.

First, consider the 2D case. The Helmholtz operator with $\omega = 5$ Hz is discretized on $N \times N$ meshes. We solve linear systems $Ax = b$ with the discretized matrix A of



(i) Flop counts ξ_{fact} (ii) Timing

FIG. 5.1. *Example* 1: *Flop counts and timing of* NEW *for* (5.1) *discretized on* $N \times N$ *meshes in two dimensions, as compared with* MF.

TABLE 5.1

Example 1: *Memory size* σ_{mem} *(number of nonzero entries in the factors) of* NEW *and* MF *for* (5.1) *discretized on* $N \times N$ *meshes in two dimensions.*

N $(n = N^2)$	256	512	1024	2048	4096
$\mathbf{l}_{\max}$	13	15	17	19	21
MF	5.36E6	2.54E7	1.18E84	5.34E8	2.40E9
NEW	4.88E6	2.06E7	8.59E7	3.53E8	1.44E9

order $n = N^2$. A sequential code in Fortran 90 for NEW is tested on a 2.33 GHz Intel E5410 processor. The flop counts and timing for the factorization method are shown in Figure 5.1. For $N = 2^8, 2^9, \ldots, 2^{12} = 4096$, we use $\mathbf{l}_{\max}$ as in Table 5.1. A relative tolerance $\tau = 10^{-6}$ and $\mathbf{l}_{\max} - \mathbf{l}_s = 9$ are used in NEW. Such a choice of $\mathbf{l}_s$ roughly gives the minimum factorization cost for each N. This is also consistent with Theorem 4.3, which indicates that $\mathbf{l}_{\max} - \mathbf{l}_s$ only changes very slowly for different N.

The memory sizes σ_{mem} of the algorithms are also reported in Table 5.1. σ_{mem} of both NEW and MF in the table scales roughly linearly, but NEW performs better. That is, when N doubles and n quadruples, σ_{mem} also roughly quadruples.

The solution costs and the accuracies of the algorithms are given in Table 5.2, where the right-hand side b in $Ax = b$ is obtained with a random x. Again, the solution cost of NEW scales linearly. NEW also produces modest accuracies. After a few steps of iterative refinements, high accuracies are obtained, and are comparable to those of MF.

Similarly, we test A discretized on $N \times N \times N$ meshes in the 3D case. The results are shown in Table 5.3, with the values of $\mathbf{l}_{\max}$. A relative tolerance $\tau = 10^{-6}$ and $\mathbf{l}_{\max} - \mathbf{l}_s = 10$ are used in NEW. Again, this choice of $\mathbf{l}_s$ roughly gives the minimum factorization cost for each N. The accuracies are similar to those in the 2D case and are not shown.

Example 2. To show that the method is more generally applicable, we test it on some matrices from the University of Florida sparse matrix collection [10]. These matrices arise from various backgrounds. See Table 5.4.

The performance of the method is shown in Table 5.5. For NEW, we choose $\tau = 10^{-6}$, and $\mathbf{l}_{\max} - \mathbf{l}_s = 7$, 7, 8, 6, 6, and 7 for the matrices from the left to the right in Table 5.5, respectively. The matrices are relatively small. However, when they are

TABLE 5.2

Example 1: *Solution costs* $\xi_{\rm sol}$ *and accuracies of the two algorithms discretized on* $N \times N$ *meshes in two dimensions.*

N $(n = N^2)$	256	512	1024	2048	4096
$\mathbf{l}_{\max}$	13	15	17	19	21
MF	$1.06E7$	$5.03E7$	$2.33E8$	$1.06E9$	$4.76E9$
NEW	$9.78E6$	$4.13E7$	$1.72E8$	$7.09E8$	$2.89E9$

(i) Solution flops $\xi_{\rm sol}$ of NEW and MF.

N $(n = N^2)$			256	512	1024	2048
MF		e_2	$1.25E-14$	$2.46E-14$	$7.61E-14$	$8.59E-14$
		γ_2	$2.13E-16$	$2.14E-16$	$2.93E-16$	$3.09E-16$
NEW	Original	e_2	$5.13E-6$	$1.17E-5$	$2.30E-5$	$3.39E-5$
		γ_2	$4.54E-8$	$6.59E-8$	$7.37E-8$	$8.78E-8$
	After 2 steps of iterative refinement	e_2	$1.01E-14$	$1.90E-14$	$7.92E-14$	$1.99E-13$
		γ_2	$1.39E-16$	$1.56E-16$	$3.63E-16$	$6.80E-16$

(ii) Accuracies of NEW and MF (relative error e_2 and relative residual γ_2).

TABLE 5.3

Example 1: *Costs (flops) and storage (number of nonzeros) of* NEW *and* MF *for* (5.1) *discretized on* $N \times N \times N$ *meshes in three dimensions.*

N $(n = N^3)$		48	96	192
$\mathbf{l}_{\max}$		12	15	18
$\xi_{\rm fact}$	MF	$1.58E11$	$1.05E13$	$6.84E14$
	NEW	$1.83E11$	$8.32E12$	$2.63E14$
$\sigma_{\rm mem}$	MF	$1.06E08$	$1.87E09$	$3.18E10$
	NEW	$9.82E07$	$1.24E09$	$1.40E10$
$\xi_{\rm sol}$	MF	$2.13E08$	$3.74E09$	$6.36E10$
	NEW	$2.01E08$	$2.56E09$	$2.89E10$

TABLE 5.4

Example 2: *Test matrices from the University of Florida sparse matrix collection* [10], *where nnz denotes the number of nonzeros of the matrix.*

Matrix	n	nnz	Description
apache2	715,176	4,817,870	3D SPD finite difference matrix from "APACHE small"
ecology2	999,999	4,995,991	Landscape ecology problem with circuit theory applied to modeling animal movement and gene flow
G3_circuit	1,585,478	7,660,826	Circuit simulation matrix from Okuyucu, AMD, Inc.
parabolic_fem	525,825	3,674,625	Parabolic finite element matrix for a diffusion-convection reaction in computational fluid dynamics
thermomech_dM	204,316	1,423,116	Finite element method for modeling the temperature and deformation of a steel cylinder
tmt_sym	726,713	5,080,61	Symmetric electromagnetic matrix from Isaak, Computational_EM_Works

TABLE 5.5

Example 2: *Costs and storage (number of nonzero entries in the factors) of* NEW *and* MF *for the test matrices in Table* 5.4.

Matrix A		apache2	ecology2	G3_circuit	parabolic_fem	thermomech_dM	tmt_sym
l_{max}		15	16	17	14	14	15
ξ_{fact}	MF	$2.52E11$	$2.59E10$	$8.77E10$	$9.14E9$	$8.56E8$	$1.33E10$
	NEW	$1.51E11$	$1.26E10$	$5.20E10$	$6.75E9$	$6.46E8$	$8.68E9$
σ_{mem}	MF	$1.77E8$	$4.88E7$	$1.14E8$	$2.50E07$	$6.51E6$	$3.31E7$
	NEW	$9.19E7$	$2.77E7$	$6.40E7$	$1.66E07$	$4.79E6$	$1.93E7$

TABLE 5.6

Example 2: *Accuracies of* NEW *and* MF *for the test matrices in Table* 5.5, *where* k_{ir} *is the number of iterative refinement steps.*

Matrix A			apache2	ecology2	G3_circuit	parabolic_fem	thermomech_dM	tmt_sym
MF		e_2	$2.51E-13$	$2.08E-13$	$1.33E-13$	$5.73E-14$	$2.90E-16$	$1.92E-11$
		γ_2	$3.49E-16$	$2.24E-16$	$2.45E-16$	$2.42E-16$	$2.26E-16$	$2.19E-16$
NEW	Original	e_2	$1.43E-4$	$1.46E-3$	$1.66E-4$	$2.39E-5$	$2.69E-9$	$1.31E-3$
		γ_2	$2.19E-8$	$2.87E-8$	$7.63E-9$	$1.62E-8$	$2.01E-9$	$2.08E-8$
	After iterative refinement	k_{ir}	4	5	4	3	2	5
		e_2	$2.13E-13$	$7.34E-12$	$1.04E-13$	$4.61E-14$	$1.67E-16$	$2.01E-12$
		γ_2	$1.66E-16$	$1.45E-16$	$1.09E-16$	$1.59E-16$	$2.86E-16$	$1.58E-16$

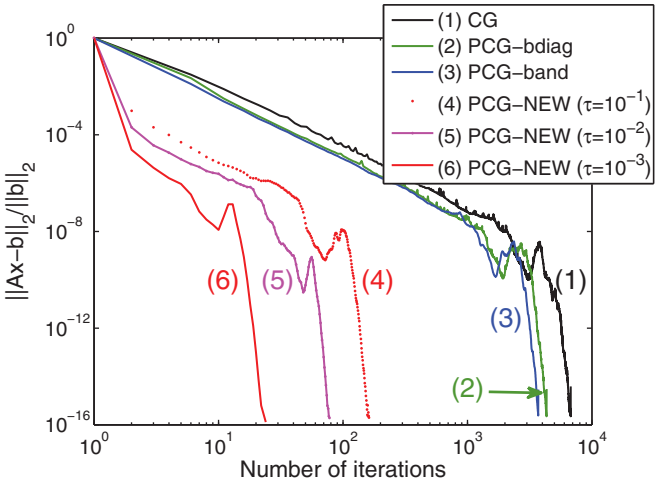


FIG. 5.2. *Example* 3: *Convergence of the conjugate method (*CG*), CG with a block diagonal preconditioner (*PCG-bdiag*), CG with a banded preconditioner (*PCG-band*), and CG with the new solver as the preconditioner (*PCG-NEW*), where a diagonal block size* 20 *is used in PCG-bdiag, a half-bandwidth* 20 *is used in PCG-band, and three* τ *values are used in PCG-NEW.*

factored, NEW is up to twice as fast as MF. NEW also requires less memory in the table. The accuracies are shown in Table 5.6. NEW reaches accuracies comparable to those of MF after few steps of iterative refinement.

Example 3. In this example, we test the effectiveness of the method as a preconditioner, when a larger relative tolerance τ is used in the compression. The matrix tmt_sym in Table 5.4 of Example 2 is tested. Its one norm condition number (returned by the MATLAB function condest) is about 1.1×10^9.

In Figure 5.2, we show the convergence behaviors of the conjugate method (CG), CG with a block diagonal preconditioner (PCG-bdiag), CG with a banded preconditioner

TABLE 5.7
Example 3: *Costs and accuracies for Figure* 5.2, *where the precomputation is to get the preconditioner (factorization).*

		Precomputation flops	Total flops	Number of iterations	γ_2
`CG`		N/A	$1.18E11$	$6,517$	$2.93E-15$
`PCG-bdiag`		$1.94E8$	$2.24E11$	$4,822$	$2.04E-15$
`PCG-band`		$3.39E8$	$1.96E11$	$3,714$	$2.70E-15$
`PCG-NEW`	$\tau=10^{-3}$	$5.73E9$	$1.06E10$	23	$1.31E-16$
	$\tau=10^{-2}$	$4.63E9$	$2.01E10$	77	$1.91E-16$
	$\tau=10^{-1}$	$3.88E9$	$3.51E10$	161	$1.97E-16$

(`PCG-band`), and `CG` with the new solver as the preconditioner (`PCG-NEW`). The matrix is ordered by nested dissection, so that `PCG-band` has a significant amount of off-diagonal entries as compared with `PCG-bdiag`. In `PCG-NEW`, $\mathbf{l}_{\max} - \mathbf{l}_s = 6$ is used. The final costs and accuracies are given in Table 5.7.

Clearly, the convergence of `PCG-NEW` with $\tau = 10^{-3}$, 10^{-2}, or 10^{-1} is significantly faster than that of the other methods, and the cost is lower, although `PCG-band` performs better than `PCG-bdiag`. (If τ is much smaller, the cost of `PCG-NEW` will be higher.) The structured preconditioner needs about 1.5 times more storage than the block diagonal one. The storage can be reduced if we use a smaller $\mathbf{l}_{\max} - \mathbf{l}_s$, which will slightly increase the number of iterations.

Remark 5.1. In our future research, we expect to compare systematically the method with multigrid preconditioners. As observed in [14, 35, 39], structured factorizations have a good potential in handling some situations which can cause difficulties for multigrid, such as multiple frequencies and incompressible limits [15, 20].

These examples are intended to show the complexity and accuracies of our new algorithms. An efficient reordering scheme for irregular meshes is expected to be done so that we can solve larger problems. Ordering schemes and pivoting strategies for nonsymmetric matrices will be considered in future work. Note that the structured factorization is not guaranteed to preserve the positive definiteness for an SPD matrix (although it is generally positive definite), since the ULV factorization is not guaranteed to do so for an SPD HSS matrix. We plan to combine the ideas here with some structured robustness techniques, say, in [21].

6. Conclusions. We show a new structured multifrontal method which is simpler and more general than some similar ones. In a strategy of combing the multifrontal method with rank-structured matrices, we show a concept of reduced matrices. This concept enables us to replace the operations on large structured matrices by those on simple compact ones. The performance is studied for both 2D and 3D discretized matrices. The sparse rank relaxation idea allows us to use our method to solve more general problems where the related off-diagonal ranks are unbounded and follow certain rank patterns.

These ideas are very useful in developing other structured methods, especially those avoiding dense update matrices, which will appear later. The idea of reduced matrices can help overcome the major computational cost for computing the update matrices in the factorization. We also expect to discuss systematically the parallel implementation and analysis in future work.

Acknowledgments. The author is very grateful to the anonymous referees for their valuable suggestions. The author would also like to thank Xiaoye S. Li for some discussions and thank Shen Wang for helping with the tests in Example 1.

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] T. Bella, Y. Eidelman, I. Gohberg, and V. Olshevsky, *Computations with quasiseparable polynomials and matrices*, Theoret. Comput. Sci., 409 (2008), pp. 158–179.

[4] S. Börm, *Efficient Numerical Methods for Non-local Operators*, European Math. Soc., Zurich, Switzerland, 2010.

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

[6] S. Chandrasekaran, P. Dewilde, M. Gu, W. Lyons, and T. Pals, *A fast solver for HSS representations via sparse matrices*, SIAM J. Matrix Anal. Appl., 29 (2006), pp. 67–81.

[7] 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.

[8] S. Chandrasekaran, P. Dewilde, M. Gu, and N. Somasunderam, *On the numerical rank of the off-diagonal blocks of Schur complements of discretized elliptic PDEs*, SIAM J. Matrix Anal. Appl., 31 (2010), pp. 2261–2290.

[9] 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.

[10] T. A. Davis and Y. Hu, *University of Florida Sparse Matrix Collection*, http://www.cise.ufl.edu/research/sparse/matrices/index.html.

[11] J. W. Demmel, J. R. Gilbert, and X. S. Li, *SuperLU Users' Guide*, http://crd.lbl.gov/~xiaoye/SuperLU/, 2003.

[12] 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.

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

[14] B. Engquist and L. Ying, *Sweeping preconditioner for the Helmholtz equation: Hierarchical matrix representation*, Pure Appl. Math., LXIV (2011), pp. 697–735.

[15] Y. A. Erlangga, C. W. Oosterlee, and C. Vuik, *A novel multigrid based preconditioner for heterogeneous Helmholtz problems*, SIAM J. Sci. Comput., 27 (2006), pp. 1471–1492.

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

[17] J. R. Gilbert and S.-H. Teng, *MESHPART, A Matlab Mesh Partitioning and Graph Separator Toolbox*, http://aton.cerfacs.fr/algor/Softs/MESHPART/.

[18] L. Grasedyck, R. Kriemann, and S. Le Borne, *Parallel Black Box Domain Decomposition Based $\mathcal{H}$-LU Preconditioning*, Technical report 115, Max Planck Institute for Mathematics in the Sciences, Leipzig, Germany, 2005.

[19] L. Grasedyck, R. Kriemann, and S. Le Borne, *Domain-decomposition based $\mathcal{H}$-LU preconditioners*, in Domain Decomposition Methods in Science and Engineering XVI, O. B. Widlund and D. E. Keyes, eds., Lect. Notes Comput. Sci. Eng. 55, Springer, Berlin, 2006, pp. 661–668.

[20] M. Griebel, D. Oeltz, and M. A. Schweitzer, *An algebraic multigrid method for linear elasticity*, SIAM J. Sci. Comput., 25 (2003), pp. 385–407.

[21] M. Gu, X. S. Li, and P. S. Vassilevski, *Direction-preserving and Schur-monotonic semiseparable approximations of symmetric positive definite matrices*, SIAM. J. Matrix Anal. Appl., 31 (2010), pp. 2650–2664.

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

[23] F. Harary, *A graph theoretic approach to matrix inversion by partitioning*, Numer. Math., 4 (1962), pp. 128–135.

[24] 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.

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

[26] W. Lyons, *Fast Algorithms with Applications to PDEs*, Ph.D. thesis, University of California, Santa Barbara, CA, 2005.

[27] P. G. Martinsson, *A fast direct solver for a class of elliptic partial differential equations*, J. Sci. Comput., 3, 2009, pp. 316–330

[28] P. G. Martinsson and V. Rokhlin, *A fast direct solver for boundary integral equations in two dimensions*, J. Comput. Phys., 205 (2005), pp. 1–23.
[29] METIS, *Family of Multilevel Partitioning Algorithms*, http://glaros.dtc.umn.edu/gkhome/views/metis.
[30] S. Parter, *The use of linear graphs in Gauss elimination*, SIAM Rev., 3 (1961), pp. 119–130.
[31] P.-O. Persson and G. Strang, *A simple mesh generator in MATLAB*, SIAM Rev., 46 (2004), pp. 329–345.
[32] P. G. Schmitz and L. Ying, *A fast direct solver for elliptic problems on general meshes in* $2D$, J. Comput. Phys., 231 (2012), pp. 1314–1338.
[33] R. P. Tewarson, *The product form of inverses of sparse matrices and graph theory*, SIAM Rev., 9 (1967), pp. 91–99.
[34] R. Vandebril, M. Van Barel, G. Golub, and N. Mastronardi, *A bibliography on semiseparable matrices*, Calcolo, 42 (2005), pp. 249–270.
[35] S. Wang, M. V. de Hoop, and J. Xia, *Acoustic inverse scattering via Helmholtz operator factorization and optimization*, J. Comput. Phys., 229 (2010), pp. 8445–8462.
[36] S. Wang, X. S. Li, J. Xia, M. V. de Hoop, and Y. Situ, *Efficient scalable algorithms for hierarchically semiseparable matrices*, SIAM J. Sci. Comput., submitted.
[37] J. Xia, *Robust and efficient multifrontal solver for large discretized PDEs*, High-Perform. Sci. Comput., M. W. Berry et al., eds., Springer, London, 2012, pp. 199–217.
[38] J. Xia, *A robust inner-outer HSS preconditioner*, Numer. Linear Algebra Appl., 19 (2012), pp. 992–1016.
[39] J. Xia, S. Chandrasekaran, M. Gu, and X. S. Li, *Superfast multifrontal method for large structured linear systems of equations*, SIAM J. Matrix Anal. Appl., 31 (2010), pp. 1382–1411.
[40] J. Xia, S. Chandrasekaran, M. Gu, and X. S. Li, *Fast algorithms for hierarchically semiseparable matrices*, Numer. Linear Algebra Appl., 17 (2010), pp. 953–976.
[41] J. Xia, *Efficient structured multifrontal factorization for large sparse matrices*, preprint, Purdue University, West Lafayette, IN, 2010; also available online from http://www.math.purdue.edu/~xiaj/work/mfhss.pdf.
[42] J. Xia, *On the complexity of some hierarchical structured matrix algorithms*, SIAM J. Matrix Anal. Appl., 33 (2012), pp. 388–410.