Proceedings of the Project Review, Geo-Mathematical Imaging Group (Purdue University, West Lafayette IN), Vol. 1 (2013) pp. 139-156.

SUPERFAST STRUCTURED SELECTED INVERSION FOR LARGE SPARSE MATRICES

JIANLIN XIA*, YUANZHE XI†, STEPHEN CAULEY‡, AND VENKATARAMANAN BALAKRISHNAN§

Abstract. We propose a structured selected inversion method for extracting the diagonal (and certain off-diagonal) blocks of the inverse of a sparse symmetric matrix A, using the multifrontal method and rank structures. A structured multifrontal LDL factorization is computed for A with a forward traversal of the assembly tree, which yields a sequence of data-sparse factors. The factors are used in a backward traversal of the tree for the structured inversion. We show that, when A arises from the discretization of certain PDEs, the intermediate matrices in the inversion can be approximated by hierarchically semiseparable (HSS) or low-rank matrices. Due to the data sparsity, the inversion has nearly $O(n)$ complexity for some 2D and 3D discretized matrices (and is thus said to be superfast), after about $O(n)$ and $O(n^{4/3})$ flops, respectively, for the structured factorization, where n is the size of the matrix. The memory requirement is also about $O(n)$. In comparison, existing inversion methods cost $O(n^{1.5})$ in 2D and $O(n^2)$ in 3D for both the factorization and the selected inversion of the matrix, with $O(n \log n)$ and $O(n^{4/3})$ memory, respectively. Numerical tests on various PDEs and sparse matrices from a sparse matrix collection are done to demonstrate the performance.

Key words. Structured selected inversion, structured multifrontal method, data sparsity, low-rank property, linear complexity, reduced matrix

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

1. Introduction. Extracting selected entries of the inverse of a sparse matrix, often called selected inversion, is critical in many scientific computing problems. Examples include uncertainty quantification in risk analysis [2], electronic structure calculations within the density functional theory framework [14], and the quantum mechanical modeling of nanotransistors and the atomistic level simulation of silicon nanowires [4]. In these examples, the diagonal entries of the matrix inverse are needed. In some other applications such as condition estimations [3], certain off-diagonal entries are also desired. The aim of this paper is to present an efficient method for computing the diagonal (denoted $\mathrm{diag}(A^{-1})$) as well as the diagonal blocks of A^{-1} for an $n \times n$ large sparse symmetric matrix A. The method also produces some off-diagonal blocks of A^{-1}, and can be modified to compute the off-diagonal entries. For convenience, we usually just mention $\mathrm{diag}(A^{-1})$.

If A is also diagonally dominant and/or positive definite, A^{-1} may have many small entries. Based on this property, a probing method is proposed in [18]. It exploits the pattern of the sparsified matrix inverse together with some standard graph theories, and computes $\mathrm{diag}(A^{-1})$ by solving a sequence of linear systems with a preconditioned Krylov subspace algorithm. Later, several approaches are proposed for more general matrices. The fast inverse with nested dissection (FIND) method in [11] and the selected inversion method in [13] use domain decomposition and compute some hierarchical Schur complements of the interior points for each subdomain. This is followed by the extraction of the diagonal entries in a top-down pass. The Selinv method in [14, 15] uses a supernode left-looking LDL factorization of A to improve the efficiency. The method in [1] focuses on the computation of a subset of A^{-1} by accessing only part of the factors where the LU or LDL factorization of A is held in out-of-core storage. The methods in [1, 11, 13, 15] all belong to the class of direct methods. For iterative methods, a Lanczos type algorithm is first used in [20]. Later, a divide-and-conquer (DC) method and a domain decomposition (DD) method are presented in [19]. The DC method assumes that the matrix can be decomposed into a 2×2 block-diagonal matrix and a low-rank matrix recursively, where the decomposed problem is solved and corrected by the

*Department of Mathematics, Purdue University, West Lafayette, IN 47907, xiaj@math.purdue.edu. The research of Jianlin Xia was supported in part by NSF grants DMS-1115572 and CHE-0957024.

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

‡Athinoula A. Martinos Center for Biomedical Imaging, Department of Radiology, Massachusetts General Hospital, Harvard University, Charlestown, MA 02129, stcauley@nmr.mgh.harvard.edu.

§School of Electrical and Computer Engineering, Purdue University, West Lafayette, IN 47907, ragu@ecn.purdue.edu.

Sherman-Morrison-Woodbury formula at each recursion level. The DD method solves each local subdomain problem and then modifies the result by a global Schur complement. Both methods use iterative solvers and sparse approximation techniques to speed up the computations.

Just like Selinv in [14, 15], our approach is also a direct method. We incorporate various sparse and structured matrix techniques, especially the multifrontal method and a rank-structured inversion to gain significant efficiency and storage benefits, as outlined below.

(1) *General sparse matrices.* Our method is applicable to symmetric discretized matrices on both 2D and 3D domains, as well as general symmetric sparse matrices. As in [22], the nested dissection ordering [8] is applied to the mesh or adjacency graph by calling some graph partitioning tools [9, 16]. Thus, our method does not rely on the special shape of the computational domain, and is more applicable than the methods in [13, 19].

(2) *Structured multifrontal method.* The multifrontal method [7] converts the overall sparse matrix factorization into a sequence of operations on some intermediate dense matrices, called frontal matrices. Its benefits includes a hierarchical tree structure (called assembly tree), nice data locality, and good parallelism. Here, we perform the LDL factorization stage of A with the structured multifrontal method in [22], which further approximates the frontal matrices in the multifrontal method by hierarchically semiseparable (HSS) forms. This converts the local dense operations into a series of structured or data-sparse matrix operations. For the convenience of our later inversion, we prove a fast Schur complement computation formula in Theorem 3.1 based on an HSS inversion procedure in [10], using a concept similar to the reduced HSS matrix in [22] and the idea that certain common basis matrices are shared between some diagonal and off-diagonal blocks.

(3) *Nearly linear complexity structured selected inversion.* After the structured multifrontal factorization, our factors are represented by a sequence of HSS or low-rank forms. The inversion procedure is performed with these data-sparse forms instead of dense ones. This significantly improves the efficiency and storage. We also derive a formula to quickly apply the inverses of the diagonal blocks to some off-diagonal blocks. See Theorem 3.2. The cost of the inversion algorithm is analyzed with a complexity optimization strategy in [22, 24] and a rank relaxation idea in [22]. That is, for certain 2D and 3D discretized problems where the frontal matrices satisfy certain rank patterns [21], we can optimize the total inversion complexity by choosing a switching level in the assembly tree (to shift from dense local factorizations to HSS ones). For these 2D and 3D matrices of order n, the inversion costs only about $O(n)$ flops, after the LDL factorization costs of about $O(n)$ and $O(n^{4/3})$, respectively. On the contrary, the methods in [11, 13, 14, 15] need $O(n^{1.5})$ and $O(n^2)$ flops for 2D and 3D, respectively, for both the LDL factorization and the selected inversion. Due to this reason, we say our method is superfast, following the terminology in [24] and in Toeplitz solutions. Similarly, our storage requirement is only about $O(n)$ for these problems. The methods in [11, 13, 14, 15] need $O(n^{4/3})$ storage. Due to the data sparsity, our method has the potential to be extended to the extraction of the off-diagonal entries of A^{-1}.

The outline of the presentation is as follows. Section 2 briefly reviews the structured multifrontal factorization with HSS techniques. Section 3 describes the structured selected inversion algorithm in detail. We provide the complexity optimization in Section 4. The numerical results for various test problems are shown in Section 5. The following notation is used:

- $F|_{\mathbf{I}\times\mathbf{J}}$ represents a submatrix of F specified by the row index set $\mathbf{I}$ and the column index set $\mathbf{J}$, and $F|_{\mathbf{I}}$ consists of a subset of the rows of F specified by the row index set $\mathbf{I}$;
- T (or $\mathcal{T}$) denotes a full binary tree with its root root(T) (or root($\mathcal{T}$)), and sib(i) and par(i) denote the sibling and parent of a node i in T, respectively.

2. Structured sparse block LDL factorization. We first briefly review the structured multifrontal methods in [22, 24] and also a block LDL variation.

2.1. HSS matrix and algorithms. The structured multifrontal methods incorporates HSS structures into the multifrontal method. An $N \times N$ HSS matrix F with a corresponding HSS tree T looks like the following [5, 21, 25]. Let each node i of a full binary tree T be associated with a consecutive index set $t_i \subset \mathcal{I} \equiv \{1 : N\}$, which satisfies $t_i \cup t_j = t_p$, $t_i \cap t_j = \emptyset$ for

$j = \mathrm{sib}(i),\ p = \mathrm{par}(i)$, and $t_{\mathrm{root}(T)} = \mathcal{I}$. An HSS matrix F is recursively defined by

$$F \equiv D_{\mathrm{root}(T)}, \quad D_i = F|_{t_i \times t_i} = \begin{pmatrix} D_{c_1} & U_{c_1} B_{c_1} V_{c_2}^T \\ U_{c_2} B_{c_2} V_{c_1}^T & D_{c_2} \end{pmatrix},$$

where c_1 and c_2 are the children of a non-leaf node i, and

$$U_i = \begin{pmatrix} U_{c_1} & \\ & U_{c_2} \end{pmatrix} \begin{pmatrix} R_{c_1} \\ R_{c_2} \end{pmatrix}, \quad V_i = \begin{pmatrix} V_{c_1} & \\ & V_{c_2} \end{pmatrix} \begin{pmatrix} W_{c_1} \\ W_{c_2} \end{pmatrix}. \tag{2.1}$$

Here, D_i, U_i, R_i, etc. are called the (HSS) generators associated with node i. The HSS rank of F is defined to be

$$r = \max_{i:\ \text{node of } T} (\max(\mathrm{rank}\, F|_{t_i \times (\mathcal{I} \setminus t_i)}, \mathrm{rank}\, F|_{(\mathcal{I} \setminus t_i) \times t_i})),$$

where $F|_{t_i \times (\mathcal{I} \setminus t_i)}$ and $F|_{(\mathcal{I} \setminus t_i) \times t_i}$ are called the ith HSS block row and column, respectively. An illustration can be found in Figure 1 later.

If r is small, we can quickly perform various HSS operations. For example, the ULV-type algorithms [5, 25] can factorize F in $O(r^2 N)$ flops, so that the solution of the linear system costs only $O(rN)$ flops. Similarly, the multiplication and the explicit inversion of HSS matrices [5, 10] can also be done in $O(r^2 N)$ flops. Recently, HSS techniques are embedded into sparse matrix computations and lead to some superfast linear solvers and efficient preconditioners [23, 22, 24]. The sparse factorization algorithm described in the next subsection is an example.

2.2. Structured multifrontal block LDL factorization. The method in [22, 24] can be immediately modified to compute an approximate multifrontal LDL factorization for a symmetric sparse matrix A

$$A \approx LDL^T,$$

which is used in our selected inversion algorithm later. This is briefly described as follows. For simplicity, we write such approximations as equalities and do not distinguish between a matrix and its structured approximation.

A is first reordered with nested dissection to reduce fill-in [8]. The variables or mesh points are grouped into subgroups, called separators. That is, the separators recursively divide the mesh or adjacency graph into smaller ones. As in [22], we use some graph partitioning tools [9, 16] to partition the graph, so that our method is not restricted to any particular shape of the domain or structure of the mesh.

The multifrontal method [7] performs the sparse factorization via some local factorizations on a series of dense matrices called frontal matrices and update matrices. A tree structured called assembly tree is then formed to organize the elimination of the separators. Label the nodes of the assembly tree $\mathcal{T}$ (and the separators) with $\mathbf{i} = 1, 2, \ldots, \mathrm{root}(\mathcal{T})$. Let $\mathcal{N}_\mathbf{i}$ be the set of ancestors of a node $\mathbf{i}$ which are connected to $\mathbf{i}$ (via the separators), and $\mathbf{t_i}$ be the index set of mesh points corresponding to $\mathbf{i}$.

If $\mathbf{i}$ is a leaf of $\mathcal{T}$, define a frontal matrix

$$\mathbf{F_i} \equiv \mathbf{F_i^0} = \begin{pmatrix} A|_{t_\mathbf{i} \times t_\mathbf{i}} & (A|_{\mathbf{t_i} \times \mathbf{t_i}})^T \\ A|_{\mathbf{t_i} \times \mathbf{t_i}} & 0 \end{pmatrix},$$

If $\mathbf{i}$ is a non-leaf node with children $\mathbf{c}_1$ and $\mathbf{c}_2$, then the frontal matrix $\mathbf{F_i}$ is formed by an assembly operation called extend-add [7]:

$$\mathbf{F_i} = \mathbf{F_i^0} \oplus \mathbf{U_{c_1}} \oplus \mathbf{U_{c_2}}, \tag{2.2}$$

where the operator $\oplus$ means that the matrices are permuted and extended to match the overall index set.

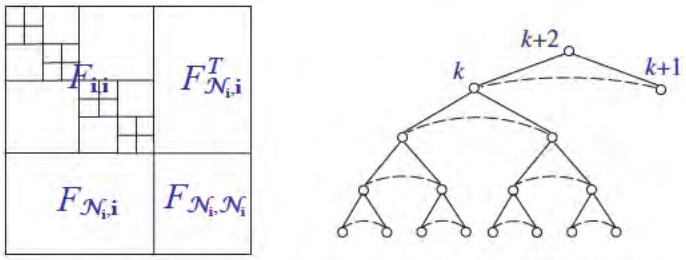


FIG. 1. *Illustration of an HSS form and the corresponding HSS tree used for a frontal matrix* $\mathbf{F_i}$.

The structured multifrontal LDL factorization follows the framework in [22]. Partition $\mathbf{F_i}$ as

$$\mathbf{F_i} \equiv \begin{pmatrix} F_{\mathbf{i},\mathbf{i}} & F^T_{\mathcal{N}_\mathbf{i},\mathbf{i}} \\ F_{\mathcal{N}_\mathbf{i},\mathbf{i}} & F_{\mathcal{N}_\mathbf{i},\mathcal{N}_\mathbf{i}} \end{pmatrix}, \tag{2.3}$$

so that the size of $F_{\mathbf{i},\mathbf{i}}$ matches the length of separator $\mathbf{i}$. Construct an HSS approximation to $\mathbf{F_i}$ with generators D_i, U_i, etc., where $F_{\mathcal{N}_\mathbf{i},\mathcal{N}_\mathbf{i}}$ corresponds to the largest indexed leaf node $k+1$ of the HSS tree T. See Figure 1. A more detailed figure can be found in [22].

Next, compute a block LDL factorization

$$\mathbf{F_i} = \begin{pmatrix} I & \\ L_{\mathcal{N}_\mathbf{i},\mathbf{i}} & I \end{pmatrix} \begin{pmatrix} F_{\mathbf{i},\mathbf{i}} & \\ & \mathbf{U_i} \end{pmatrix} \begin{pmatrix} I & (L_{\mathcal{N}_\mathbf{i},\mathbf{i}})^T \\ & I \end{pmatrix},$$

where

$$L_{\mathcal{N}_\mathbf{i},\mathbf{i}} = F_{\mathcal{N}_\mathbf{i},\mathbf{i}} F^{-1}_{\mathbf{i},\mathbf{i}}, \tag{2.4}$$

and $\mathbf{U_i}$ is the update matrix of the form

$$\mathbf{U_i} = F_{\mathcal{N}_\mathbf{i},\mathcal{N}_\mathbf{i}} - F_{\mathcal{N}_\mathbf{i},\mathbf{i}} F^{-1}_{\mathbf{i},\mathbf{i}} F^T_{\mathcal{N}_\mathbf{i},\mathbf{i}}.$$

Since we are interested in the selected inversion, unlike the method in [22], an HSS inversion for $F_{\mathbf{i},\mathbf{i}}$ is computed here (see Section 3.1). Then we can quickly form

$$\mathbf{U_i} = F_{\mathcal{N}_\mathbf{i},\mathcal{N}_\mathbf{i}} - U_{k+1} B_k^T (U_k^T F^{-1}_{\mathbf{i},\mathbf{i}} U_k) B_k U_{k+1}^T. \tag{2.5}$$

(The actual computation is done following Theorem 3.1 below.)

For our later convenience of describing the structured selected inversion algorithm, we outline the structured LDL factorization in Theorem 1.

3. Structured sparse selected inversion. We then describe our structured selected inversion for A. We start with the discussion of an HSS inversion algorithm, as needed in the scheme.

3.1. HSS inversion and fast update matrix computation. An HSS inversion algorithm is proposed in [10], and the idea is to recursively apply the Sherman-Morrison-Woodbury formula, since an HSS matrix can be written as a block diagonal matrix (formed by smaller HSS matrices) plus a low-rank update. A simplification of the algorithm for a symmetric HSS matrix F is outlined as follows. It includes two stages, where we assume all appropriate inverses exist without any stability issue.

In a bottom-up traversal of the HSS tree T, if a node i is a leaf, set $\bar{D}_i \equiv D_i$, $\bar{U}_i \equiv U_i$, and compute

$$\hat{D}_i = (\bar{U}_i^T \bar{D}_i^{-1} \bar{U}_i)^{-1}, \quad G_i = \bar{D}_i^{-1} - \bar{D}_i^{-1} \bar{U}_i \hat{D}_i \bar{U}_i^T \bar{D}_i^{-T}, \tag{3.1}$$
$$\tilde{U}_i = \bar{D}_i^{-1} \bar{U}_i \hat{D}_i.$$

Algorithm 1 Structured multifrontal LDL factorization for A (a variation of the method in [22])

procedure SMF
  for node/separator **i** from 1 to root $(\mathcal{T}) - 1$ **do**
    if i is a leaf of $\mathcal{T}$ **then**
      $\mathbf{F_i} = \mathbf{F_i^0}$ ▷ *Initial frontal matrix*
    end if
    if i is at level $\mathbf{l} > \mathbf{l}_s$ of $\mathcal{T}$ **then** ▷ *Exact factorization below* $\mathbf{l}_s$
      Compute $F_{\mathbf{i},\mathbf{i}}^{-1}$ ▷ *Exact Inversion*
      $L_{\mathcal{N}_\mathbf{i},\mathbf{i}} = F_{\mathcal{N}_\mathbf{i},\mathbf{i}} F_{\mathbf{i},\mathbf{i}}^{-1}$
      $\mathbf{U_i} = F_{\mathcal{N}_\mathbf{i},\mathcal{N}_\mathbf{i}} - L_{\mathcal{N}_\mathbf{i},\mathbf{i}} F_{\mathcal{N}_\mathbf{i},\mathbf{i}}^T$ ▷ *Exact update matrix*
    else ▷ *Structured factorization above* $\mathbf{l}_s$
      Compute an HSS approximation to $\mathbf{F_i}$
      Compute the HSS generators for $F_{\mathbf{i},\mathbf{i}}^{-1}$
      Compute $\mathbf{U_i}$ with (2.5) and Theorem 3.1
    end if
    if i is a left node **then**
      Push $\mathbf{U_i}$ onto a stack ▷ *For later extend-add operation*
    else ▷ *Assembly of* $\mathcal{F}_\mathbf{p}$
      Pop $\mathbf{U_j}$ from the stack for $\mathbf{j} = \text{sib}\,(\mathbf{i})$
      $\mathbf{F_p} = \mathbf{F_p^0} \hat{\oplus} \mathbf{U_j} \hat{\oplus} \mathbf{U_i}$ ▷ *Exact extend-add operation*
    end if
  end for
  Compute an HSS approximation to $\mathbf{F}_{\text{root}(\mathcal{T})}$
end procedure

Otherwise, let

$$\bar{D}_i = \begin{pmatrix} \hat{D}_{c_1} & B_{c_1} \\ B_{c_1}^T & \hat{D}_{c_2} \end{pmatrix}, \quad \bar{U}_i = \begin{pmatrix} R_{c_1} \\ R_{c_2} \end{pmatrix}, \tag{3.2}$$

where c_1 and c_2 are the children of i. Then compute (3.1) and

$$\begin{pmatrix} \tilde{R}_{c_1} \\ \tilde{R}_{c_2} \end{pmatrix} = \bar{D}_i^{-1} \begin{pmatrix} R_{c_1} \\ R_{c_2} \end{pmatrix} \hat{D}_i.$$

For the node $k = \text{root}(T)$, only compute

$$G_k \equiv \bar{D}_k^{-1}.$$

In a top-down traversal of T, if i is a non-leaf node with children c_1 and c_2, let $\hat{G}_i \equiv G_i$. Partition $\hat{G}_i$ conformably following (3.2) as

$$\hat{G}_i = \begin{pmatrix} \hat{G}_{i;1,1} & \hat{G}_{i;1,2} \\ \hat{G}_{i;2,1} & \hat{G}_{i;2,2} \end{pmatrix}.$$

Then let

$$\tilde{B}_{c_1} = \hat{G}_{i;1,2}.$$

Also, if $\text{par}(i) \neq k$, let

$$\hat{G}_i = G_i + \begin{pmatrix} \tilde{R}_{c_1} \\ \tilde{R}_{c_2} \end{pmatrix} \hat{G}_{\text{par}(i);1,1} \begin{pmatrix} \tilde{R}_{c_1}^T & \tilde{R}_{c_2}^T \end{pmatrix}.$$

Repeat these steps until all non-leaf nodes are visited. Then for each leaf i, set

$$\tilde{D}_i \equiv \hat{G}_i.$$

Then it can be verified that $\tilde{D}_i, \tilde{U}_i, \tilde{R}_i, \tilde{B}_i$ are the HSS generators of F^{-1}. This HSS inversion method is used when we computed the diagonal blocks of the inverse of A.

In addition, the information computed in this inversion process can also benefit the computation of the update matrix $\mathbf{U_i}$ in (2.5), as well as some additional computations later in the selected inversion. In fact, to compute $\mathbf{U_i}$, instead of using the HSS form of $F_{\mathbf{i,i}}^{-1}$ directly, we can use an fast way similar to the idea of reduced matrices in [22]. See Theorem 3.1.

THEOREM 3.1. *Let the HSS generators of* $\mathbf{F_i}$ *be* D_i, U_i, *etc. Then*

$$U_k^T F_{\mathbf{i,i}}^{-1} U_k = \bar{U}_k^T \bar{D}_k^{-1} \bar{U}_k,$$

where $\bar{D}_k$ *is given in (3.2) with* $i = k$.

Proof. We prove this theorem by induction on the height h of the HSS tree T associated with $\mathbf{F_i}$. If $h = 3$, then

$$F_{\mathbf{i,i}}^{-1} = \operatorname{diag}(G_1, G_2) + \operatorname{diag}(\tilde{U}_1, \tilde{U}_2) G_3 \operatorname{diag}(\tilde{U}_1^T, \tilde{U}_2^T).$$

It is obvious that

$$\begin{aligned} U_3^T F_{\mathbf{i,i}}^{-1} U_3 =& \begin{pmatrix} R_1^T & R_2^T \end{pmatrix} \operatorname{diag}(U_1^T G_1 U_1, U_2^T G_2 U_2) \begin{pmatrix} R_1 \\ R_2 \end{pmatrix} \\ &+ \begin{pmatrix} R_1^T & R_2^T \end{pmatrix} \operatorname{diag}(U_1^T \tilde{U}_1, U_2^T \tilde{U}_2) G_3 \operatorname{diag}(\tilde{U}_1^T U_1, \tilde{U}_2^T U_2) \begin{pmatrix} R_1 \\ R_2 \end{pmatrix} \\ =& \begin{pmatrix} R_1^T & R_2^T \end{pmatrix} G_3 \begin{pmatrix} R_1 \\ R_2 \end{pmatrix} \\ =& \bar{U}_3^T \bar{D}_3^{-1} \bar{U}_3. \end{aligned}$$

Then assume that the theorem is true for any HSS tree with height $3, 4, \ldots, h-1$. Consider

$$F_{\mathbf{i,i}} = \operatorname{diag}(D_{k_1}, D_{k_2}) + \operatorname{diag}(U_{k_1}, U_{k_2}) \begin{pmatrix} & B_{k_1} \\ B_{k_1}^T & \end{pmatrix} \operatorname{diag}(U_{k_1}^T, U_{k_2}^T),$$

where k_1 and k_2 are the children of $k = \operatorname{root}(T)$. According to a variation of the Sherman-Morrison-Woodbury formula in [10], we obtain

$$\begin{aligned} F_{\mathbf{i,i}}^{-1} =& \operatorname{diag}(D_{k_1}^{-1}, D_{k_2}^{-1}) - \operatorname{diag}(D_{k_1}^{-1}, D_{k_2}^{-1}) \operatorname{diag}(U_{k_1}, U_{k_2}) \mathbf{D}_k^{-1} \\ &\times \operatorname{diag}(U_{k_1}^T, U_{k_2}^T) \operatorname{diag}(D_{k_1}^{-1}, D_{k_2}^{-1}) \\ &+ \operatorname{diag}(D_{k_1}^{-1}, D_{k_2}^{-1}) \operatorname{diag}(U_{k_1}, U_{k_2}) \mathbf{D}_k^{-1} \left[\begin{pmatrix} & B_{k_1} \\ B_{k_1}^T & \end{pmatrix} + \mathbf{D}_k^{-1} \right]^{-1} \mathbf{D}_k^{-1} \\ &\times \operatorname{diag}(U_{k_1}^T, U_{k_2}^T) \operatorname{diag}(D_{k_1}^{-1}, D_{k_2}^{-1}), \end{aligned}$$

where

$$\mathbf{D}_k = \operatorname{diag}(U_{k_1}^T D_{k_1}^{-1} U_{k_1}, U_{k_2}^T D_{k_2}^{-1} U_{k_2}).$$

Clearly,

$$\begin{aligned} U_k^T F_{\mathbf{i,i}}^{-1} U_k =& \begin{pmatrix} R_{k_1}^T & R_{k_2}^T \end{pmatrix} \mathbf{D}_k \begin{pmatrix} R_{k_1} \\ R_{k_2} \end{pmatrix} - \begin{pmatrix} R_{k_1}^T & R_{k_2}^T \end{pmatrix} \mathbf{D}_k \mathbf{D}_k^{-1} \mathbf{D}_k \begin{pmatrix} R_{k_1} \\ R_{k_2} \end{pmatrix} \\ &+ \begin{pmatrix} R_{k_1}^T & R_{k_2}^T \end{pmatrix} \mathbf{D}_k \mathbf{D}_k^{-1} \left[\begin{pmatrix} & B_{k_1} \\ B_{k_1}^T & \end{pmatrix} + \mathbf{D}_k^{-1} \right]^{-1} \mathbf{D}_k^{-1} \mathbf{D}_k \begin{pmatrix} R_{k_1} \\ R_{k_2} \end{pmatrix} \\ =& \begin{pmatrix} R_{k_1}^T & R_{k_2}^T \end{pmatrix} \begin{pmatrix} (U_{k_1}^T D_{k_1}^{-1} U_{k_1})^{-1} & B_{k_1} \\ B_{k_1}^T & (U_{k_2}^T D_{k_2}^{-1} U_{k_2})^{-1} \end{pmatrix}^{-1} \begin{pmatrix} R_{k_1} \\ R_{k_2} \end{pmatrix}. \end{aligned}$$

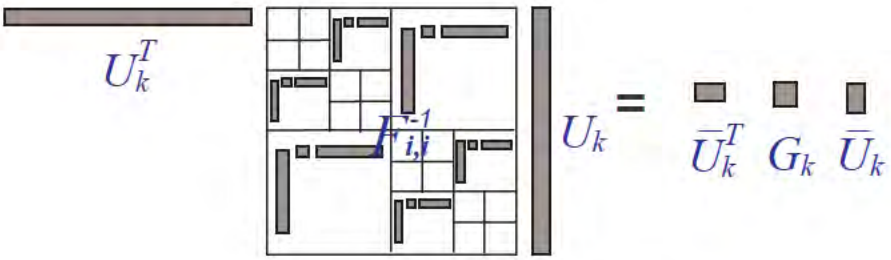


FIG. 2. *A pictorial demonstration for the Theorem 3.1*

By induction, we have

$$U_{k_1}^T D_{k_1}^{-1} U_{k_1} = \bar{U}_{k_1}^T \bar{D}_{k_1}^{-1} \bar{U}_{k_1} = \hat{D}_{k_1}^{-1},$$
$$U_{k_2}^T D_{k_2}^{-1} U_{k_2} = \bar{U}_{k_2}^T \bar{D}_{k_2}^{-1} \bar{U}_{k_2} = \hat{D}_{k_2}^{-1}.$$

Thus,

$$\begin{aligned} U_k^T F_{\mathbf{i},\mathbf{i}}^{-1} U_k &= \begin{pmatrix} R_{k_1}^T & R_{k_2}^T \end{pmatrix} \begin{pmatrix} \hat{D}_{k_1} & B_{k_1} \\ B_{k_1}^T & \hat{D}_{k_2} \end{pmatrix}^{-1} \begin{pmatrix} R_{k_1} \\ R_{k_2} \end{pmatrix} \\ &= \bar{U}_k^T \bar{D}_k^{-1} \bar{U}_k. \end{aligned}$$

For node k, we further have $\bar{D}_k^{-1} = G_k$. □

That is, $\bar{D}_k$ plays a role similar to the final reduced matrix defined in [22] when $F_{\mathbf{i},\mathbf{i}}$ is factorized by ULV-type algorithms [5]. Similarly, the HSS generators of $F_{\mathbf{i},\mathbf{i}}^{-1}$ and U_k have inherent relations. In fact, the nested representation of the basis matrix U_k is also used for the off-diagonal blocks of $F_{\mathbf{i},\mathbf{i}}$. Or we can vaguely say that U_k and $F_{\mathbf{i},\mathbf{i}}$ share some common bases. If $F_{\mathbf{i},\mathbf{i}}$ has size N and HSS rank r, this theorem can be utilized to reduce the complexity of computing $U_k^T F_{\mathbf{i},\mathbf{i}}^{-1} U_k$ with HSS inversion from $O(r^2 N)$ to only $O(r^3)$ with a simple $\bar{U}_k^T \bar{D}_k^{-1} \bar{U}_k$. See Figure 2 for an illustration.

3.2. Basic ideas for structured sparse selected inversion. The extraction of the diagonal of $C \equiv A^{-1}$ includes two stages, a forward one (block LDL factorization) and a backward one (inversion). The basic idea of the inversion can be illustrated with a simple case.

Consider a symmetric matrix A after nested dissection as follows:

$$A = \begin{pmatrix} A_{11} & & A_{13} \\ & A_{22} & A_{23} \\ A_{31} & A_{32} & A_{33} \end{pmatrix}.$$

A block LDL factorization of A looks like

$$A = \begin{pmatrix} I & & \\ & I & \\ L_{31} & L_{32} & I \end{pmatrix} \begin{pmatrix} A_{11} & & \\ & A_{22} & \\ & & \mathbf{F}_3 \end{pmatrix} \begin{pmatrix} I & & L_{31}^T \\ & I & L_{32}^T \\ & & I \end{pmatrix},$$

where

$$L_{31} = A_{31} A_{11}^{-1}, \quad L_{32} = A_{32} A_{22}^{-1}, \quad \mathbf{F}_3 = A_{33} - L_{31} A_{11} L_{31}^T - L_{32} A_{22} L_{32}^T.$$

Then

$$C = \begin{pmatrix} I & & -L_{31}^T \\ & I & -L_{32}^T \\ & & I \end{pmatrix} \begin{pmatrix} A_{11}^{-1} & & \\ & A_{22}^{-1} & \\ & & \mathbf{F}_3^{-1} \end{pmatrix} \begin{pmatrix} I & & \\ & I & \\ -L_{31} & -L_{32} & I \end{pmatrix}$$
$$= \begin{pmatrix} \begin{pmatrix} A_{11}^{-1} & \\ & A_{22}^{-1} \end{pmatrix} + \begin{pmatrix} -L_{31}^T \\ -L_{32}^T \end{pmatrix} \mathbf{F}_3^{-1} \begin{pmatrix} -L_{31} & -L_{32} \end{pmatrix} & \begin{pmatrix} -L_{31}^T\mathbf{F}_3^{-1} \\ -L_{32}^T\mathbf{F}_3^{-1} \end{pmatrix} \\ \begin{pmatrix} -\mathbf{F}_3^{-1}L_{31} & -\mathbf{F}_3^{-1}L_{32} \end{pmatrix} & \mathbf{F}_3^{-1} \end{pmatrix}.$$

Thus,

$$\begin{aligned} \operatorname{diag}(C) &= \begin{pmatrix} A_{11}^{-1} + L_{31}^T\mathbf{F}_3^{-1}L_{31} & A_{22}^{-1} + L_{32}^T\mathbf{F}_3^{-1}L_{32} & D_{33}^{-1} \end{pmatrix} \\ &= \begin{pmatrix} A_{11}^{-1} + L_{31}^T C_{31} & A_{22}^{-1} + L_{32}^T C_{32} & D_{33}^{-1} \end{pmatrix}. \end{aligned} \tag{3.3}$$

This means that $\operatorname{diag}(C)$ is determined by the nonzero structure of the L factor from the LDL factorization [13]. The idea can then be recursively applied.

In our proposed new method, both the factorization and the inversion stages are performed in structured forms, so that the appropriate blocks are approximated by either HSS or low-rank forms.

3.3. General scheme. Our method for the sparse selected inversion includes two stages, a structured multifrontal LDL factorization stage as in Section 2.2, and a structured inversion stage. In the first stage, we traverse the assembly tree $\mathcal{T}$ in a postorder, and in the second stage, we traverse $\mathcal{T}$ in a reverse postorder. Let

$$C \equiv A^{-1}.$$

If $\mathbf{i} = \mathbf{k} \equiv \operatorname{root}(\mathcal{T})$, let

$$C_{\mathbf{k},\mathbf{k}} = F_{\mathbf{k}}^{-1}.$$

The diagonal entries of $C_{\mathbf{k},\mathbf{k}}$ can be obtained directly from the $\tilde{D}_i$ generators of the HSS form of $F_{\mathbf{k}}^{-1}$.

If $\mathbf{i} < \mathbf{k}$, first update $L_{\mathcal{N}_\mathbf{i},\mathbf{i}}$ to obtain $C_{\mathcal{N}_\mathbf{i},\mathbf{i}}$, which is lower triangular part in C. According to (2.3), $L_{\mathcal{N}_\mathbf{i},\mathbf{i}}$ is approximated by a low-rank form:

$$L_{\mathcal{N}_\mathbf{i},\mathbf{i}} = F_{\mathcal{N}_\mathbf{i},\mathbf{i}} F_{\mathbf{i},\mathbf{i}}^{-1} = U_{k+1} B_k^T U_k^T F_{\mathbf{i},\mathbf{i}}^{-1}, \tag{3.4}$$

where U_k as in (2.1) is recursively available. Here, we need to compute $U_k^T F_{\mathbf{i},\mathbf{i}}^{-1}$, which can be quickly obtained like in Theorem 3.1.

THEOREM 3.2. *With the same notation as in Theorem 3.1, we have*

$$P_k^T \equiv U_k^T F_{\mathbf{i},\mathbf{i}}^{-1} = \bar{U}_k \bar{D}_k^{-1} \operatorname{diag}(\tilde{U}_{k_1}^T, \tilde{U}_{k_2}^T) = \begin{pmatrix} R_{k_2}^T \tilde{B}_{k_1}^T \tilde{U}_{k_1}^T & R_{k_1}^T \tilde{B}_{k_1} \tilde{U}_{k_2}^T \end{pmatrix}.$$

Proof. We still prove this corollary by induction on the height h of the HSS tree T for $\mathbf{F}_\mathbf{i}$. It is clear that the claim is true when $h = 3$. Suppose it is also true for all the HSS trees with heights $3, 4, \ldots, h-1$.

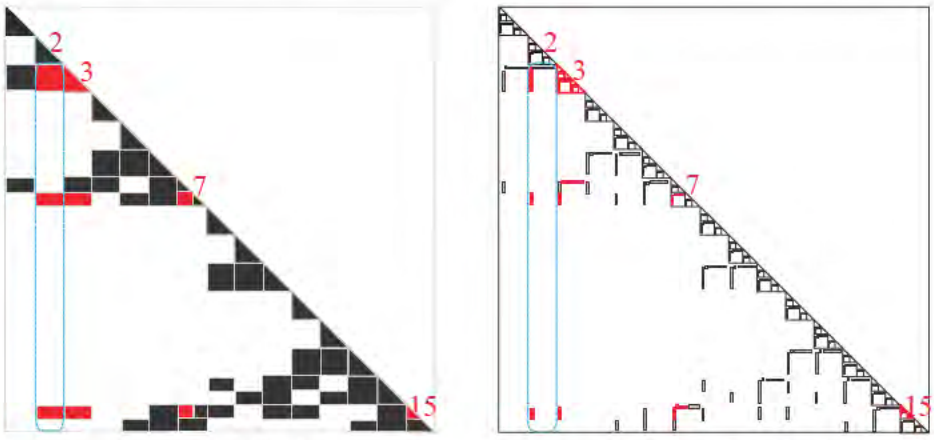
Based on the results in the proof of Theorem 3.1, it can be verified that

$$U_k^T F_{\mathbf{i},\mathbf{i}}^{-1} = \bar{U}_k^T \bar{D}_k^{-1} \operatorname{diag}(\hat{D}_{k_1} U_{k_1}^T D_{k_1}^{-1}, \hat{D}_{k_2} U_{k_2}^T D_{k_2}^{-1}).$$

By induction, we know

$$U_{k_i}^T D_{k_i}^{-1} = \bar{U}_{k_i}^T \bar{D}_{k_i}^{-1} \operatorname{diag}(\tilde{U}_{k_{i,1}}^T, \tilde{U}_{k_{i,2}}^T),$$

where $k_{i,1}$ and $k_{i,2}$ are the children of k_i for $i = 1, 2$.



Exact method Structured method

FIG. 3. *The non-zero patterns of the L factor from the exact and structured multifrontal LDL factorization of a reordered matrix A, respectively. The red blocks are the ones needed in the computation of the lower-triangular inverse part* $C_{\mathcal{N}_{2,2}}$ *(within the blue dotted areas).*

Therefore,

$$U_k^T F_{\mathbf{i},\mathbf{i}}^{-1} = \bar{U}_k^T \bar{D}_k^{-1} \operatorname{diag}(\hat{D}_{k_1} \bar{U}_{k_1}^T \bar{D}_{k_1}^{-1} \operatorname{diag}(\tilde{U}_{k_{1,1}}^T, \tilde{U}_{k_{1,2}}^T), \hat{D}_{k_2} \bar{U}_{k_2}^T \bar{D}_{k_2}^{-1} \operatorname{diag}(\tilde{U}_{k_{2,1}}^T, \tilde{U}_{k_{2,2}}^T)).$$

Notice that

$$\hat{D}_{k_i} \bar{U}_{k_i}^T \bar{D}_{k_i}^{-1} = \begin{pmatrix} \tilde{R}_{k_{i,1}}^T & \tilde{R}_{k_{i,2}}^T \end{pmatrix},$$

and this results in

$$U_k^T F_{\mathbf{i},\mathbf{i}}^{-1} = \bar{U}_k \bar{D}_k^{-1} \operatorname{diag}(\tilde{U}_{k_1}^T, \tilde{U}_{k_2}^T).$$

For node k, we can further simplify the above equation

$$\bar{U}_k \bar{D}_k^{-1} \operatorname{diag}(\tilde{U}_{k_1}^T, \tilde{U}_{k_2}^T) = \begin{pmatrix} R_{k_2}^T \tilde{B}_{k_1}^T \tilde{U}_{k_1}^T & R_{k_1}^T \tilde{B}_{k_1} \tilde{U}_{k_2}^T \end{pmatrix}.$$

□

Then, $L_{\mathcal{N}_\mathbf{i},\mathbf{i}}$ is in the form

$$L_{\mathcal{N}_\mathbf{i},\mathbf{i}} = U_{k+1} B_k^T P_k^T,$$

and the update of $L_{\mathcal{N}_\mathbf{i},\mathbf{i}}$ reduces to that of its column basis U_{k+1}.

Next, consider the computation of

$$P_{k+1} = \tilde{C}_{\mathcal{N}_\mathbf{i},\mathcal{N}_\mathbf{i}} U_{k+1}, \tag{3.5}$$

where $\tilde{C}_{\mathcal{N}_\mathbf{i},\mathcal{N}_\mathbf{i}}$ is a matrix formed by grouping certain inverse generators together that are associated with $\mathcal{N}_\mathbf{i}$ based on the index set $\tilde{t}_i$. This procedure and the comparison with the exact method are demonstrated in Figure 3.

In order to carry out (3.5) more efficiently, we partition the rows of U_{k+1} and obtain p disjoint sub-matrices $U_{\mathbf{j_1},\mathbf{i}}, \ldots, U_{\mathbf{j_p},\mathbf{i}}$ for each $\mathbf{j_s}$ in $\mathcal{N}_\mathbf{i}$ such that the row dimension of $U_{\mathbf{j_s},\mathbf{i}}$ equals the cardinality of $t_{\mathbf{j_s},\mathbf{i}}$. This partition is also applied when forming the diagonal blocks and off-diagonal blocks of $\tilde{C}_{\mathcal{N}_\mathbf{i}\mathcal{N}_\mathbf{i}}$. See Figure 4 for an example.

In our algorithm, the symmetric matrix $\tilde{C}_{\mathcal{N}_\mathbf{i}\mathcal{N}_\mathbf{i}}$ is in a structured form and the generators of its lower triangular part are stored in separate arrays associated with $\mathcal{N}_\mathbf{i}$. In order to reduce the

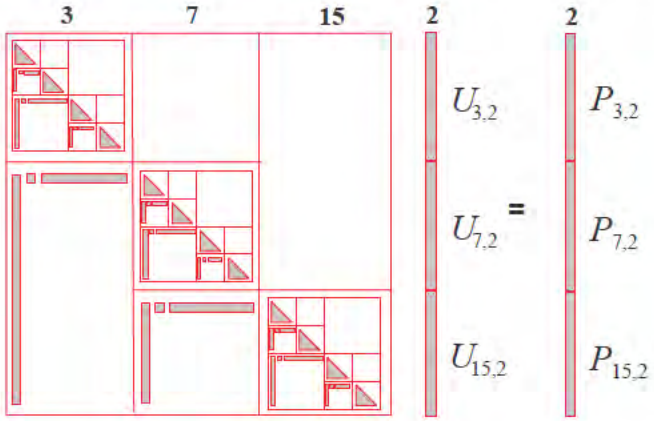


FIG. 4. *The structure of $\tilde{C}_{\mathcal{N}_2\mathcal{N}_2}$, U_k, and P_k in the computation of (3.5) for the matrix in Figure 3.*

memory reference, we traverse the nodes in $\mathcal{N}_\mathbf{i}$ for $\mathbf{j_1}, \mathbf{j_2}, \ldots, \mathbf{j_p}$ and complete all the operations associated with $\mathbf{j_s}$ $(1 \le s \le p)$ in (3.5) if $\mathbf{j_s}$ is currently being visited.

More specifically, when $\mathbf{j_s}$ is reached, first form a submatrix $\tilde{C}_{\mathbf{j_s},\mathbf{j_s}}$ from $C_{\mathbf{j_s},\mathbf{j_s}}$. The extraction is based on the relative location of $t_{\mathbf{j_s},\mathbf{i}}$ in $t_{\mathbf{j_s}}$. We use $\mathbf{I_1}$ to store this information. For convenience, this procedure is denoted as

$$\mathbf{I_1} = \mathrm{Indp}(t_{\mathbf{j_s},\mathbf{i}}, t_{\mathbf{j_s}}).$$

Then compute

$$P_{\mathbf{j_s},\mathbf{i}} \leftarrow P_{\mathbf{j_s},\mathbf{i}} + C_{\mathbf{j_s},\mathbf{j_s}}|_{\mathbf{I_1},\mathbf{I_1}} U_{\mathbf{j_s},\mathbf{i}}. \tag{3.6}$$

As $C_{\mathbf{j_s},\mathbf{j_s}}$ is stored in the form of (3.12), (3.6) only involves a partial HSS matrix-vector multiplication and a low-rank update. Next, multiply the off-diagonal part below $\tilde{C}_{\mathbf{j_s},\mathbf{j_s}}$ in $\tilde{C}_{\mathcal{N}_\mathbf{i}\mathcal{N}_\mathbf{i}}$ with $U_{\mathbf{j_s},\mathbf{i}}$. This requires to form a proper submatrix from $C_{\mathcal{N}_{\mathbf{j_s}},\mathbf{j_s}}$ in (3.12). Similar to the extraction operation for $C_{\mathbf{j_s},\mathbf{j_s}}$, we find

$$\mathbf{I_2} = \mathrm{Indp}(\tilde{t}_{\mathbf{j_s}}, \tilde{t}_\mathbf{i}),$$

and compute

$$\begin{bmatrix} P_{\mathbf{j_{s+1}},\mathbf{i}} \\ \vdots \\ P_{\mathbf{j_p},\mathbf{i}} \end{bmatrix} \leftarrow \begin{bmatrix} P_{\mathbf{j_{s+1}},\mathbf{i}} \\ \vdots \\ P_{\mathbf{j_p},\mathbf{i}} \end{bmatrix} + \underbrace{P_{k+1}|_{\mathbf{I_2}} B_k^T (P_k|_{\mathbf{I_1}})^T}_{\text{associated with } \mathbf{j_s}} U_{\mathbf{j_s},\mathbf{i}}. \tag{3.7}$$

Finally, $P_{k+1}|_{\mathbf{I_2}} B_k^T P_k^T|_{\mathbf{I_1}}$ is used as its transpose block in $\tilde{C}_{\mathcal{N}_\mathbf{i}\mathcal{N}_\mathbf{i}}$ to finish the computation for block $P_{\mathbf{j_s},\mathbf{i}}$:

$$P_{\mathbf{j_s},\mathbf{i}} \leftarrow P_{\mathbf{j_s},\mathbf{i}} + P_k|_{\mathbf{I_1}} B_k (P_{k+1}|_{\mathbf{I_2}})^T \begin{bmatrix} U_{\mathbf{j_{s+1}},\mathbf{i}} \\ \vdots \\ U_{\mathbf{j_p},\mathbf{i}} \end{bmatrix}. \tag{3.8}$$

After that, the rest operations in (3.5) does not involve the generators associated with $\mathbf{j_s}$ any more. See Algorithm 2 for a more detailed description, where a switching level $\mathbf{l_s}$ in the assembly tree is used as in [24], so that dense and structured factorizations are done before and after $\mathbf{l_s}$, respectively.

Algorithm 2 Compute $C_{\mathcal{N}_\mathbf{i},\mathbf{i}}$ in Step 5 and 8 of Algorithm 3

procedure
  if node/separator $\mathbf{i}$ is at level $\mathbf{l} < \mathbf{l}_s$ of $\mathcal{T}$ **then** ▷ *low rank form of* $C_{\mathcal{N}_\mathbf{i},\mathbf{i}}$
    Partition U_{k+1} into $|\mathcal{N}_\mathbf{i}|$ pieces
    for s from 1 to $|\mathcal{N}_\mathbf{i}|$ **do**
      Compute $P_{k+1} = \tilde{C}_{\mathcal{N}_\mathbf{i},\mathcal{N}_\mathbf{i}} U_{k+1}$ as in (3.6) to (3.8)
      Store $C_{\mathcal{N}_\mathbf{i},\mathbf{i}}$ in the form of $P_{k+1} B_k^T P_k^T$
    end for
  else ▷ *dense block* $C_{\mathcal{N}_\mathbf{i},\mathbf{i}}$
    Partition $L_{\mathcal{N}_\mathbf{i},\mathbf{i}}$ into $|\mathcal{N}_\mathbf{i}|$ pieces
    for s from 1 to $|\mathcal{N}_\mathbf{i}|$ **do**
      if node/separator $\mathbf{j_s}$ in $\mathcal{N}_\mathbf{i}$ is at level $\mathbf{l} < \mathbf{l}_s$ of $\mathcal{T}$ **then**
        Treat $L_{\mathcal{N}_\mathbf{i},\mathbf{i}}$ as U_{k+1} and compute (3.6) to (3.8)
      else ▷ *dense matrix operation*
        Treat $\tilde{C}_{\mathbf{j_s},\mathbf{j_s}}$ and $\tilde{C}_{\mathcal{N}_{\mathbf{j_s}},\mathbf{j_s}}$ as dense block and compute (3.6) to (3.8)
      end if
    end for
    store $C_{\mathcal{N}_\mathbf{i},\mathbf{i}} \equiv P_{k+1}$
  end if
end procedure

Clearly, $C_{\mathcal{N}_\mathbf{i},\mathbf{i}}$ is still in a low-rank form

$$C_{\mathcal{N}_\mathbf{i},\mathbf{i}} = P_{k+1} B_k^T P_k^T. \tag{3.9}$$

Finally, we compute

$$C_{\mathbf{i},\mathbf{i}} = F_{\mathbf{i},\mathbf{i}}^{-1} - L_{\mathcal{N}_\mathbf{i},\mathbf{i}}^T C_{\mathcal{N}_\mathbf{i},\mathbf{i}}. \tag{3.10}$$

According to (3.4) and (3.9),

$$C_{\mathbf{i},\mathbf{i}} = F_{\mathbf{i},\mathbf{i}}^{-1} - P_k (B_k U_{k+1}^T P_{k+1} B_k^T) P_k^T.$$

Here, we simply compute the matrix multiplication inside the parenthesis

$$\Theta = B_k U_{k+1}^T P_{k+1} B_k^T, \tag{3.11}$$

and store $C_{\mathbf{i},\mathbf{i}}$ in the form of

$$C_{\mathbf{i},\mathbf{i}} = F_{\mathbf{i},\mathbf{i}}^{-1} - P_k \Theta P_k^T. \tag{3.12}$$

Since $F_{\mathbf{i},\mathbf{i}}^{-1}$ is an HSS matrix, the diagonal entries of $C_{\mathbf{i},\mathbf{i}}$ can be obtained by subtracting the diagonal entries of $P_k \Theta P_k^T$ from those of the generators $\tilde{D}$ associated with $F_{\mathbf{i},\mathbf{i}}^{-1}$.

Algorithm 3 Structured inversion for extracting the diagonal blocks of A^{-1}

procedure SINV
  Compute $C_{\mathbf{k},\mathbf{k}} = F_{\mathbf{k}}^{-1}$ for $\mathbf{k} \equiv \text{root}\,(\mathcal{T})$
  for node/separator $\mathbf{i}$ from root $(\mathcal{T}) - 1$ to 1 **do**
    if $\mathbf{i}$ is at level $\mathbf{l} < \mathbf{l}_s$ of $\mathcal{T}$ **then** ▷ *Low-rank form of* $L_{\mathcal{N}_\mathbf{i},\mathbf{i}}$ *above* $\mathbf{l}_s$
      Compute $C_{\mathcal{N}_\mathbf{i},\mathbf{i}}$ with (3.9) and Corollary 3.2 ▷ *Low-rank* $C_{\mathcal{N}_\mathbf{i},\mathbf{i}}$ *and* $C_{\mathbf{i},\mathbf{i}}$
      Compute $C_{\mathbf{i},\mathbf{i}}$ with (3.12)
    else ▷ *Dense matrix* $L_{\mathcal{N}_\mathbf{i},\mathbf{i}}$ *below* $\mathbf{l}_s$
      Compute $C_{\mathcal{N}_\mathbf{i},\mathbf{i}} = \tilde{C}_{\mathcal{N}_\mathbf{i},\mathcal{N}_\mathbf{i}} L_{\mathcal{N}_\mathbf{i},\mathbf{i}}$ ▷ *Dense matrix* $C_{\mathcal{N}_\mathbf{i},\mathbf{i}}$ *and* $C_{\mathbf{i},\mathbf{i}}$
      Compute $C_{\mathbf{i},\mathbf{i}} = F_{\mathbf{i},\mathbf{i}}^{-1} - L_{\mathcal{N}_\mathbf{i},\mathbf{i}}^T C_{\mathcal{N}_\mathbf{i},\mathbf{i}}$
    end if
  end for
end procedure

4. Complexity optimization for the structured inversion. For the purpose of our later comparison, we restate the complexity and memory usage for algorithms in [11, 13, 14, 15] as well the inversion algorithm based on the multifrontal method (without structured operations). Assume A is discretized one a 2D $N \times N$ mesh ($n = N^2$) or a 3D $N \times N \times N$ mesh ($n = N^3$). Then the factorization cost ξ_{fact}, selected inversion cost ξ_{inv}, and storage σ_{mem} are reported in Table 4.1.

Then we turn to the analysis for our structured method, which is similar to the one for the structured linear solvers in [22, 24]. We first present the case when the HSS rank in the frontal matrices is bounded by a constant r, and then show some similar results by letting r grow under some assumptions.

LEMMA 4.1. *Suppose our algorithm is applied to an order n matrix A discretized on a 2D $N \times N$ mesh ($n = N^2$) or a 3D $N \times N \times N$ mesh ($n = N^3$), and the HSS ranks of the frontal matrices in the multifrontal method are bounded by a constant r. If we choose the switching level $\mathbf{l_s} = O(\log N)$ so that the inversion costs before and after $\mathbf{l_s}$ are the same, then after about $O(n)$ flops in 2D and about $O(n^{4/3})$ flops in 3D for the structured factorization, the minimum structured inversion cost is about $O(n)$, and the memory requirement is about $O(n)$. The details are given in Table 4.2.*

Proof. The proof is similar to those in [22, 24], except that here we optimize the inversion costs. The total number of levels in $\mathcal{T}$ is $\mathbf{l}_{\max} = O(\log N)$. For the 2D case, the factorization and inverse

TABLE 4.1
Factorization cost ξ_{fact}, inverse cost ξ_{inv} and storage σ_{mem} of the exact method for extracting the diagonal entries of a discretized matrix A on a regular mesh, as in [11, 13, 14, 15].

	ξ_{fact}	ξ_{inv}	σ_{mem}
2D	$O(n^{1.5})$	$O(n^{1.5})$	$O(n \log n)$
3D	$O(n^2)$	$O(n^2)$	$O(n^{4/3})$

TABLE 4.2
Factorization cost ξ_{fact}, inverse cost ξ_{inv} and storage σ_{mem} of the structured method for extracting the diagonal entries of a discretized matrix A on a regular mesh, r is the maximal HSS rank in all frontal matrices.

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

costs satisfy the following estimations, respectively:

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

$$\xi_{\text{inv}} = \underbrace{\sum_{\mathbf{l}=\mathbf{l}_s+1}^{\mathbf{l}_{\max}} 4^{\mathbf{l}} O\left(\left(\frac{N}{2^{\mathbf{l}}}\right)^3\right)}_{\text{before the switching level}} + \underbrace{\sum_{\mathbf{l}=\mathbf{0}}^{\mathbf{l}_s} 4^{\mathbf{l}} O\left(r^2\left(\frac{N}{2^{\mathbf{l}}}\right)\right)}_{\text{after the switching level}} = O\left(\frac{N^3}{2^{\mathbf{l}_s}}\right) + O(r^2N2^{\mathbf{l}_s}). \tag{4.2}$$

We minimize the inverse cost by letting

$$O\left(\frac{N^3}{2^{\mathbf{l}_s}}\right) = O(r^2N2^{\mathbf{l}_s}),$$

and obtain the optimal condition

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

Thus, the minimal inverse cost is $\xi_{\text{inv}} = O(rn)$ and factorization cost is $\xi_{\text{fact}} = O(rn\log n)$ for the 2D case.

Similarly, the estimated costs for 3D problems are

$$\xi_{\text{fact}} = \sum_{\mathbf{l}=\mathbf{l}_s+1}^{\mathbf{l}_{\max}} 8^{\mathbf{l}} O\left(\left(\frac{N}{2^{\mathbf{l}}}\right)^6\right) + \sum_{\mathbf{l}=\mathbf{0}}^{\mathbf{l}_s} 8^{\mathbf{l}} O\left(r\left(\frac{N}{2^{\mathbf{l}}}\right)^4\right) = O\left(\frac{N^6}{8^{\mathbf{l}_s}}\right) + O(rN^4), \tag{4.3}$$

$$\xi_{\text{inv}} = \sum_{\mathbf{l}=\mathbf{l}_s+1}^{\mathbf{l}_{\max}} 8^{\mathbf{l}} O\left(\left(\frac{N}{2^{\mathbf{l}}}\right)^6\right) + \sum_{\mathbf{l}=\mathbf{0}}^{\mathbf{l}_s} 8^{\mathbf{l}} O\left(r^2(\frac{N}{2^{\mathbf{l}}})^2\right) = O\left(\frac{N^6}{8^{\mathbf{l}_s}}\right) + O(r^2N^2 2^{\mathbf{l}_s}). \tag{4.4}$$

Here, the optimality condition is still obtained by minimizing ξ_{inv}:

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

The storage requirement is the same as that in [22] and the results are summarized in the fourth column of Table 4.2. ∎

According to Lemma 4.1, if the HSS rank r for each frontal matrix is small, our inversion algorithm is much faster than the exact method (as in Table 4.1), and has nearly linear complexity for both 2D and 3D cases. However, in practice, the HSS ranks of the frontal matrices are not bounded by a small constant, or depend on the mesh size N in the discretized PDE problems. For example, researchers have observed that the HSS rank in the 3D Helmholtz equation is bounded by $O(N)$ [6]. Recently, a *rank relaxation* idea in [21] studies the off-diagonal ranks at each level l of the HSS tree T, called *rank pattern* r_l, and shows that even though r_l varies for different l, the total complexity can still be satisfactory. This idea is then further extended to a sparse rank relaxation idea in [22]. Based on these, we have the performance results of our method for more general problems.

THEOREM 4.2. *Use the same notation as in Lemma 4.1, and assume the maximum rank at level l of the HSS tree for any frontal matrix follows the rank pattern r_l in the first column of Table 4.4. If we choose the switching level $\mathbf{l}_s = O(\log N)$ such that the inversion costs before and after $\mathbf{l}_s$ are equal, then after about $O(n)$ flops in 2D and about $O(n^{4/3})$ flops in 3D for the structured factorization, the minimum structured inversion cost is about $O(n)$, and the memory requirement is about $O(n)$. The details are given in Tables 4.3 and 4.4.*

Note that our selected inversion costs about $O(n)$ flops for all these cases. On the other hand, the counts in Table 4.1 for [11, 13, 14, 15] are significantly higher, which are $O(n^{1.5})$ in 2D and $O(n^2)$ in 3D.

TABLE 4.3
Cost $\xi_{\rm inv}$ and storage $\sigma_{\rm mem}$ of the structured method for extracting the diagonal entries of a discretized matrix A of order n on a 2D $n^{1/2}\times n^{1/2}$ mesh, where $p\in\mathbb{N}$, $\alpha>0$, and r_l is the rank pattern for the frontal matrices as in [22].

r_l		$r=\max r_l$	$\xi_{\rm inv}$	$\sigma_{\rm mem}$
$O(1)$		$O(1)$	$O(n)$	$O(n\log\log n)$
$O((\log N_l)^p)$		$O((\log N)^p)$		
$O(N_l^{1/p})$	$p>3$	$O(N^{1/p})$		
	$p=3$	$O(N^{1/3})$	$O(n\log n)$	
	$p=2$	$O(N^{1/2})$	$O(n)$	
$O(\alpha^{l_{\max}-l}r_0)$	$\alpha<\alpha<2^{1/3}$	$<O(N^{1/3})$		
	$\alpha=2^{1/3}$	$O(N^{1/3})$	$O(n\log n)$	
	$2^{1/3}<\alpha<2^{1/2}$	$<O(N^{1/2})$	$O(n)$	
	$\alpha=2^{1/2}$	$O(N^{1/2})$		

TABLE 4.4
Cost $\xi_{\rm inv}$ and storage $\sigma_{\rm mem}$ of the structured method for extracting the diagonal entries of a discretized matrix A of order n on a 3D $n^{1/3}\times n^{1/3}\times n^{1/3}$ mesh, where $p\in\mathbb{N}$, $\alpha>0$, and r_l is the rank pattern for the frontal matrices as in [22].

r_l		$r=\max r_l$	$\xi_{\rm inv}$	$\sigma_{\rm mem}$
$O(1)$		$O(1)$	$O(n)$	$O(n)$
$O((\log_2 N_l)^p)$		$O((\log_2 N)^p)$		
$O(N_l^{1/p}),\ p>3$		$O(N^{1/p})$		
$O(N_l^{1/p})$	$p=3$	$O(N^{1/3})$	$O(n\log n)$	$O(n\log^{1/2} n)$
	$p=2$	$O(N^{1/2})$	$O(n\log n)$	$O(n\log n)$
$O(\alpha^{l_{\max}-l}r_0)$	$\alpha<2^{1/3}$	$<O(N^{1/3})$	$O(n)$	$O(n\log^{1/2} n)$
	$\alpha=2^{1/3}$	$O(N^{1/3})$	$O(n\log n)$	$O(n\log^{1/2} n)$
	$2^{1/3}<\alpha<2^{1/2}$	$<O(N^{1/2})$	$O(n)$	$O(n\log n)$
	$\alpha=2^{1/2}$	$O(N^{1/2})$	$O(n\log n)$	$O(n\log n)$

5. Numerical experiments. In this section, we test our algorithms on two model problems as well as various matrices from the University of Florida Sparse Matrix Collection. The algorithms are implemented in Matlab and carried out on a macbook pro with a 2.3GHz Intel quad-core i7 processor and 8GB DDR3 memory. The following notation is used through out this section.

- `MF`: selected inversion with the exact multifrontal method.
- `NEW`: new structured selected inversion.
- x: vector formed by the diagonal entries of A^{-1} computed by `MF`.
- $\tilde{x}$: vector formed by the diagonal entries of A^{-1} computed by `NEW`.
- $e=\frac{\|x-\tilde{x}\|_2}{\|x\|_2}$: relative accuracy.
- τ: relative compression tolerance used in the HSS construction.

In the examples, we also report the structured multifrontal LDL factorization costs, although similar results can be found in [22]. The main reason is the completeness in comparison, and another reason is the subtle difference due to the use of HSS inversion and Theorem 3.1 here.

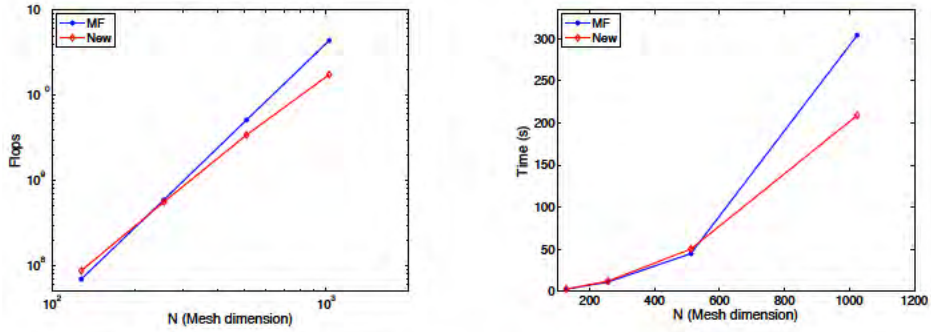
EXAMPLE 1. Consider the Poison's equation

$$\Delta\mathbf{u}=\mathbf{f}.$$

We first discretize the Laplacian operator on a 2D $N\times N$ mesh, and the matrix A has the order $n=N^2$. For $N=128$, 256, 512, and 1024, we fix $\tau=10^{-5}$ and $\mathbf{l}_{\max}-\mathbf{l}_{\rm s}=9$. The factorization performance is recorded in Table 5.1 and also plotted in Figure 5. We find that when n gets larger,

TABLE 5.1
Factorization flops ξ_{fact} and timing (in seconds) for Example 1 with various dimensions, where $\mathrm{l}_{\max} - \mathrm{l}_{\mathrm{s}} = 9$ and the compression tolerance $\tau = 10^{-5}$.

$n\ (= N^2)$		128^2	256^2	512^2	1024^2
$\mathrm{l}_{\max}$		11	13	15	17
ξ_{fact}	MF	4.32e7	3.74e8	3.27e9	2.79e10
	NEW	4.31e7	3.03e8	1.91e9	1.06e10
Time(s)	MF	6.96	3.03e1	1.56e2	4.27e3
	NEW	7.29	3.01e1	1.56e2	3.04e3



Flops Time

FIG. 5. *Example 1: Factorization flops and timing (in seconds) for NEW and MF with various n.*

TABLE 5.2
Memory (number of nonzeros in the factors) for Example 1 with various dimensions, where $\mathrm{l}_{\max} - \mathrm{l}_{\mathrm{s}} = 9$ and the compression tolerance $\tau = 10^{-5}$.

$n\ (= N^2)$		128^2	256^2	512^2	1024^2
$\mathrm{l}_{\max}$		11	13	15	17
Memory	MF	6.38e5	3.06e6	1.44e7	6.67e7
	NEW	6.37e5	2.86e6	1.23e7	5.14e7

the flops ratio of NEW and MF approach 4 and 8, respectively, which is consistent with our analysis in Theorem 4.2.

The memory sizes ξ_{mem} are reported in Table 5.2. The ratio for NEW is smaller than that of MF, and scales almost linearly.

The inversion costs and accuracy are shown in Table 5.3. Like the factorization performance, the flops ratios for NEW and MF scales as $O(n)$ and $O(n^{1.5})$, respectively. We also plot the inversion flops cost and timing in Figure 6.

EXAMPLE 2. Consider the Helmholtz equation

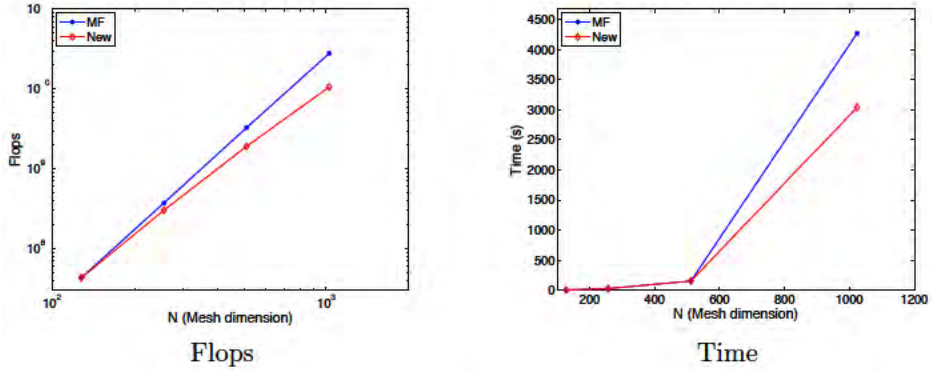
$$[-\Delta - \omega^2 c(\mathbf{x})^{-2}]\mathbf{u} = \mathbf{f}. \tag{5.1}$$

We then discretize the Helmholtz operator with $\omega = 2$Hz on a $N \times N$ mesh and the resulting matrix A is an indefinite matrix. We choose $\tau = 10^{-7}$ and still fix $\mathrm{l}_{\max} - \mathrm{l}_{\mathrm{s}} = 9$. The factorization performance, memory sizes, and inversion performance are shown in Table 5.4, 5.5, and 5.6, respectively. Although the rank pattern of Helmholtz problems is different with Poison problems, we still get the same scaling for ξ_{fac}, ξ_{mem}, and ξ_{inv}. This confirms our analysis in Theorem 4.2 numerically.

TABLE 5.3

Selected inversion flops ξ_{inv} and timing (in seconds) for Example 1 with various dimensions, where $e = \frac{\|x-\bar{x}\|_2}{\|x\|_2}$, and $\mathrm{l}_{\max} - \mathrm{l}_{\mathrm{s}} = 9$.

$n\ (= N^2)$		128^2	256^2	512^2	1024^2
$\mathrm{l}_{\max}$		11	13	15	17
ξ_{inv}	MF	6.97e7	5.94e8	5.14e9	4.38e10
	NEW	8.73e7	5.62e8	3.43e9	1.75e10
Time(s)	MF	2.59	1.10e1	4.49e1	3.04e2
	NEW	2.84	1.21e1	5.04e1	2.09e2
e		2.45e−16	1.91e−6	8.06e−6	6.69e−5



Flops Time

FIG. 6. *Example 1: Inversion flops and timing (in seconds) for NEW and MF with various n.*

TABLE 5.4

Factorization flops ξ_{fact} and timing (in seconds) for Example 2 with various dimensions, where $\mathrm{l}_{\max} - \mathrm{l}_{\mathrm{s}} = 9$ and the compression tolerance $\tau = 10^{-7}$.

$n\ (= N^2)$		128^2	256^2	512^2	1024^2
$\mathrm{l}_{\max}$		12	14	16	18
ξ_{fact}	MF	4.14e7	3.68e8	3.27e9	2.74e10
	NEW	4.03e7	3.68e8	2.95e9	1.25e10
Time(s)	MF	9.00	3.80e1	1.87e2	2.98e2
	NEW	9.04	3.82e1	1.99e2	3.61e2

TABLE 5.5

Memory (number of nonzeros in the factors) for Example 2 with various dimensions, where $\mathrm{l}_{\max} - \mathrm{l}_{\mathrm{s}} = 9$ and the compression tolerance $\tau = 10^{-7}$.

$n\ (= N^2)$		128^2	256^2	512^2	1024^2
$\mathrm{l}_{\max}$		12	14	16	18
Memory	MF	5.76e5	2.83e6	1.36e7	6.31e7
	NEW	5.60e5	2.55e6	1.12e7	4.73e7

TABLE 5.6
Selected inversion flops ξ_{inv} and timing (in seconds) for Example 2 with various dimensions, where $e = \frac{\|x-\tilde{x}\|_2}{\|x\|_2}$, and $\mathbf{l}_{\max} - \mathbf{l}_{\text{s}} = 9$.

$n\ (= N^2)$		128^2	256^2	512^2	1024^2
$\mathbf{l}_{\max}$		12	14	16	18
ξ_{inv}	`MF`	$6.57e7$	$5.79e8$	$5.06e9$	$4.29e10$
	`NEW`	$7.42e7$	$5.73e8$	$3.85e9$	$2.16e10$
Time(s)	`MF`	3.40	$1.42e1$	$5.67e1$	$2.29e2$
	`NEW`	3.77	$1.63e1$	$6.83e1$	$2.84e2$
e		$3.83e{-6}$	$3.55e{-5}$	$5.04e{-4}$	$1.19e{-1}$

EXAMPLE 3. To show our algorithm has broader applications, we apply it to some 2D and 3D matrices from the University of Florida Sparse Matrix Collection. See Table 5.7 for the characteristics of the test matrices.

TABLE 5.7
Test matrices in 2D and 3D from University of Florida Sparse Matrix Collection, where nnz stands for number of non-zeros in the matrix.

Matrix	n	nnz	Description
`bodyy6`	19,366	77,057	From NASA, collected by Alex Pothen
`wathen120`	36,441	301,101	Random matrix from Andy Wathen, Oxford Univ
`2cubes_sphere`	101,492	1,647,264	FEM 3D electromagnetic problem from Evan Um, Geophysics, Stanford
`pwtk`	217,918	11,524,432	Stiffness matrix from pressurized wind tunnel problem
`parabolic_fem`	525,825	3,674,625	Parabolic FEM from a constant homogeneous diffusion problem
`tmt_sym`	726,713	2,903,837	Symmetric electromagnetics problem from David Isaak, Computational_EM_Works
`ecology2`	999,999	4,995,991	Matrix obtained when circuit theory applied to animial/gene flow
`G3_circuit`	1,585,478	7,660,826	Circuit simulation problem from Ufuk Okuyucu, AMD , Inc

We fix $\tau = 10^{-5}$ and $\mathbf{l}_{\max} - \mathbf{l}_{\text{s}} = 9$ for all the test matrices. Also, as the dimension of test matrices grows from the top to the bottom in Table 5.8, we increase $\mathbf{l}_{\max}$ accordingly. We report the factorization and inversion flops for `NEW` and `MF` in the table and find `NEW` always gives better performance. The relative error is given in the last column of Table 5.8. Even though the error for `tmt_sym` and `ecology2` has only two digits, this problem can be easily solved if we use a smaller τ instead.

REFERENCES

[1] P. AMESTOY, I. DUFF, J. L'EXCELLENT, Y. ROBERT, F. ROUET AND B. UÇAR, *On computing inverse entries of a sparse matrix in an out-of-core environment*, SIAM J. Sci. Comput., 34 (2012), pp. A1975–A1999.

[2] C. BEKAS, A. CURIONI, AND I. FEDULOVA, *Low cost high performance uncertainty quantification*, in Proceedings of the 2nd Workshop on High Performance Computational Finance, ACM, 2009, p. 8.

[3] Å. BJÖRCK, *Numerical Methods for Least Squares Problems*, SIAM, Philadelphia, 1996.

[4] S. CAULEY, J. JAIN, C. K. KOH, AND V. BALAKRISHNAN, *A scalable distributed method for quantum-scale device simulation*, J. Appl. Phys., 101 (2007), p. 123715.

TABLE 5.8
Computational flops for the test matrices in Example 3, where $\tau = 10^{-5}$, $\mathbf{l}_{\max} - \mathbf{l}_{\rm s} = 9$, *and* $\xi_{\rm fact}$ *and* $\xi_{\rm inv}$ *represent the factorization and selected inversion flops, respectively.*

Matrix	$\mathbf{l}_{\max}$	MF		NEW		e
		$\xi_{\rm fact}$	$\xi_{\rm inv}$	$\xi_{\rm fact}$	$\xi_{\rm inv}$	
bodyy6	14	$5.31e7$	$8.42e7$	$4.46e7$	$7.52e7$	$1.98e-8$
wathen120	13	$3.33e8$	$5.28e8$	$2.68e8$	$4.16e8$	$1.74e-11$
2cubes_sphere	14	$7.24e10$	$1.11e11$	$4.72e10$	$9.05e10$	$3.75e-4$
pwtk	13	$4.36e10$	$6.81e10$	$4.00e10$	$6.55e10$	$2.22e-10$
parabolic_fem	14	$1.01e10$	$1.72e10$	$6.46e9$	$1.19e10$	$5.96-6$
tmt_sym	15	$1.56e10$	$2.56e10$	$7.91e9$	$1.42e10$	$5.78-2$
ecology2	16	$2.83e10$	$4.49e10$	$1.17e10$	$2.04e10$	$5.23e-2$
G3_circuit	16	$1.11e11$	$1.77e11$	$5.33e10$	$9.30e10$	$1.37-5$

[5] S. CHANDRASEKARAN, P. DEWILDE, M. GU, AND T. PALS, *A fast ULV decomposition solver for hierarchically semiseparable representations*, SIAM J. Matrix Anal. Appl., 28 (2006), pp. 603–622.

[6] 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.

[7] I. S. DUFF AND J. K. REID, *The multifrontal solution of indefinite sparse symmetric linear equations*, ACM Bans. Math. Software, 9 (1983), pp. 302–325.

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

[9] J. R. GILBERT AND S.-H. TENG, *MESHPART, A Matlab Mesh Partitioning and Graph Separator Toolbox*, http://aton.cerfacs.fr/algor/Softs/MESHPART/.

[10] A. GILLMAN, P. YOUNG, AND P. G. MARTINSSON, *A direct solver with $O(N)$ complexity for integral equations on one-dimensional domains*, Front. Math. China, 7 (2012), pp. 217–247.

[11] S. LI, S. AHMED, G. KLIMECK, AND E. DARVE, *Computing entries of the inverse of a sparse matrix using the FIND algorithm*, J. Comput. Phys., 227 (2008), pp. 9408–9427.

[12] S. LI AND E. DARVE, *Extension and optimization of the FIND algorithm: Computing Green's and less-than Green's functions*, J. Comput. Phys., 231 (2012), pp. 1121–1139.

[13] L. LIN, J. LU, L. YING, R. CAR, AND W. E, *Fast algorithm for extracting the diagonal of the inverse matrix with application to the electronic structure analysis of metallic systems*, Commun. Math. Sci. 7 (2009), pp. 755–777.

[14] L. LIN, C. YANG, J. LU, L. YING, AND W. E, *A fast parallel algorithm for selected inversion of structured sparse matrices with application to 2D electronic structure calculations*, SIAM J. Sci. Comput., 33 (2010), pp. 1329–1351.

[15] L. LIN, C. YANG, J. MEZA, J. LU, L. YING, AND W. E, *SelInv–An algorithm for selected inversion of a sparse symmetric matrix*, ACM Trans. Math. Software, 37 (2011), pp. 40:1–40:19.

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

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

[18] J. TANG AND Y. SAAD, *A probing method for computing the diagonal of the matrix inverse*, Technical report umsi-2010-42, Minnesota Supercomputer Institute, University of Minnesota, Minneapolis, 2009.

[19] J. TANG AND Y. SAAD, *Domain-decomposition-type methods for computing the diagonal of a matrix inverse*, SIAM J. Sci. Comput., 33 (2011), pp. 2823–2847.

[20] R. B. SIDJE AND Y. SAAD, *Rational approximation to the Fermi-Dirac function with applications in density functional theory*, Tech. Rep. umsi-2008-279, Minnesota Supercomputer Institute, University of Minnesota, 2008.

[21] J. XIA, *On the complexity of some hierarchical structured matrix algorithms*, SIAM J. Matrix Anal. Appl., 33 (2012), pp. 388–410.

[22] J. XIA, *Efficient structured multifrontal factorization for general large sparse matrices*, SIAM J. Sci. Comput., 35 (2013), pp. A832–A860.

[23] J. XIA, *Randomized sparse direct solvers*, SIAM J. Matrix Anal. Appl., 34 (2013), pp. 197–227.

[24] 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 (2009), pp. 1382–1411.

[25] 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.