Proceedings of the Project Review, Geo-Mathematical Imaging Group (Purdue University, West Lafayette IN), Vol. 1 (2013) pp. 171-183.

MATRIX-FREE SPARSE DIRECT SOLVERS

JIANLIN XIA*

Abstract. Existing direct solvers for large linear systems, especially sparse ones, require the matrices to be explicitly available. In practical computations, often only matrix-vector products instead of the matrix are available, which makes iterative methods the only choice. Here, we derive matrix-free sparse direct solvers based on matrix-vector products. Two stages are involved. The first stage is to reconstruct the sparse matrix with a multi-nested dissection ordering. This idea recursively reorders the matrix and the separators with nested dissection, so that a compact probing strategy can reconstruct the matrix entries via the simultaneous recovery of multiple blocks with a small number of vectors. For discretized matrices in 2D or 3D, each dimension thus corresponds to one layer of nested dissection. The number of matrix-vector products required is $O(\log^d N)$, where d is the dimension and N is the mesh dimension (e.g., 2D $N \times N$ or $N \times N \times N$ mesh), and the reconstruction is thus said to be superfast. A simplified fast scheme can also be used, which uses $O(N \log^{d-1} N)$ In the second stage, the matrix is factorized in a randomized multifrontal method based on rank structures and randomized sampling. The overall solver costs about $O(n)$ and $O(n^{4/3})$ flops for some 2D and 3D problems, respectively, where n is the matrix size (e.g., $n = N^2$ in 2D and N^3 in 3D). The solver has a potential to work for varying parameters. For example, when the diagonal or few entries of the matrix change, we can reuse at least part of the previous factorizations, which is nearly impossible in classical direct factorizations. The multi-nested dissection idea also has other benefits such as in the structured solutions.

Key words. Matrix-free sparse solver, sparse reconstruction, multi-nested dissection, multi-assembly tree, compact probing, structured multifrontal method

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

1. Introduction.

The solution of large sparse linear systems is one of the most critical tasks in modern scientific or engineering computations. Two classes of sparse solvers are typically used, and their benefits and limitations have been thoroughly studied [1, 7, 8, 9, 13].

Among one of the limitations of sparse direct solvers is the need of the explicit matrix A. However, in practical applications such as sparse grid methods [22] and iterative mesh refinements [2, 6, 26], it may often be difficult or expensive to form the matrix. On the other hand, in such applications, the products of A with vectors can often be conveniently computed in about $O(n)$ flops, where A is $n \times n$. For such cases, iterative methods are then usually used.

On the other hand, for these cases where A is not explicitly available, direct methods may have significant advantages due to the robustness and the efficiency for multiple right-hand sides. Sometimes, it may also need to approximately reconstruct A so as to form effective preconditioners for iterative solutions.

Thus in this work, we propose some matrix-free sparse direct solvers, which includes a fast or superfast reconstruction of A with matrix-vector products, followed by a fast structured factorization. A *superfast* sparse reconstruction needs about $O(\log^d N)$ matrix-vector products for A discretized on a $N \times N \times \cdots \times N$ mesh. (We name it superfast similarly to a structured solution in [30] and Toeplitz solutions.) A *fast* sparse reconstruction needs about $O(N \log^{d-1} N)$ products instead, but is simpler. Here, we assume the mesh or adjacency graph is known, well shaped [20, 21, 24], and only locally connected. The local connectivity means that each mesh or graph point is connected to only a finite number of other points nearby, and thus each row or column of A has a finite number of nonzero entries. For simplicity, we also assume A is symmetric, since our primary consideration is the connectivity of the graph points.

Our reconstruction scheme includes the following major features.

1. A multi-nested dissection (or multi-layer nested dissection) ordering is proposed to extend standard nested dissection [11] to multiple layers. That is, the separators in nested dissection are recursively reordered with nested dissection again, until the bottom layer separators are one dimensional (1D) lines. This strategy is not only useful in our reconstruction of the

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

matrix, but also in rank structured solutions [5, 25]. Correspondingly, we also extend the concept of assembly trees [10, 18] to multi-assembly trees.

2. The diagonal blocks can be reconstructed simultaneously with a small number of matrix-vector products, since multi-nested dissection helps distribute the diagonal blocks into different rows and columns. This avoids using multiple matrix-vector products for reconstructing multiple diagonal blocks. A basic sparse reconstruction scheme is then shown and analyzed.
3. A compact probing strategy is given to efficiently reconstruct the off-diagonal blocks of A. A fundamental idea is to separate those blocks in the off-diagonal block that do not have overlapping column indices from those that do. Blocks without overlaps are reconstructed simultaneously. We then cycle through all the descendants of an (outer layer) node in the multi-assembly tree. Each cycling peels off some connections.

Under certain mild assumptions, we can show that the reconstruction is superfast. In fact, we may also simplify the scheme to get a fast version. This is especially attractive for 3D problems, where the overall reconstruction cost is no more than the factorization cost, even with the fast solvers in [28, 29].

In the factorization stage, we employ a fast randomized sparse direct solver in [29], which is based on the multifrontal method, rank structures, and randomized sampling. Due to the use of matrix-vector products both here and in [29], we mainly emphasize the potential of updating the factorization when A is varied due to some parameters. (In standard LU factorizations, this is usually impossible.) Examples include $A - sI$ with a shift sI, or the consideration of the Helmholtz discretized matrix $A - \sigma I$, where σ is related to the frequency.

The remaining part of the paper includes these sections. Section 2 briefly reviews nested dissection and then shows a basic sparse reconstruction. Our superfast reconstruction with multi-nested dissection is given in Section 3. The sparse factorization together with the potential of updating the factorization are shown in Section 4. Section 5 presents the numerical tests.

2. Basic sparse reconstruction scheme and nested dissection. Consider a sparse matrix A with the corresponding adjacency graph G, which may also be a mesh. The nested dissection ordering (ND) [11] of is one of the most useful ordering methods to reduce fill-in in sparse direct solutions. It uses separators or subsets of G to recursively divide G into smaller graphs. The separators are then organized into multiple levels with the aid of a binary tree $\mathbf{T}$, which may be used as an assembly tree [10] in sparse factorizations. Upper level separators are ordered later and thus eliminated later. See Figure 1. We say this ND is a one-layer version. After ND, the matrix looks like Figure 2(i), with the corresponding assembly tree $\mathbf{T}$ in Figure 2(ii). For convenience, assume $\mathbf{T}$ is in its postordering with the nodes denote by $\mathbf{i} = 1, 2, \ldots, \mathbf{k} \equiv \text{root}(\mathbf{T})$. Also assume $\mathbf{s_i}$ is the set of all mesh points corresponding to node or separator $\mathbf{i}$, $|\mathbf{s_i}|$ is its cardinality, and

$$\mathcal{N}_\mathbf{i} = \{\mathbf{j} | \mathbf{j} \text{ is an ancestor of } \mathbf{i} \text{ and } A|_{\mathbf{s_i} \times \mathbf{s_j}} \neq 0\},$$

where $A|_{\mathbf{s_i} \times \mathbf{s_j}}$ denotes a submatrix of A selected by the row index set $\mathbf{s_i}$ and the column index set $\mathbf{s_j}$. (Similarly, we use $A|_{:\times \mathbf{s_j}}$ to mean the selection of all the entries of A in columns $\mathbf{s_j}$.) $\mathcal{N}_\mathbf{i}$ is the set of all upper level neighbors [28] of $\mathbf{i}$.

We can then present a basic (and less efficient) scheme to reconstruct the matrix A via matrix-vector products. This is done in a top-down levelwise traversal of $\mathbf{T}$. For convenience, we reconstruct the block column corresponding to each separator $\mathbf{i}$. For $\mathbf{k} = \text{root}(\mathbf{T})$ at level 1 of $\mathbf{T}$, let

$$V^{(1)} = \begin{pmatrix} 0 \\ I \end{pmatrix}_{n \times |\mathbf{s_k}|}.$$

Then clearly,

$$A|_{:\times \mathbf{s_k}} = AV^{(1)}.$$

See Figure 3(i)–(ii).

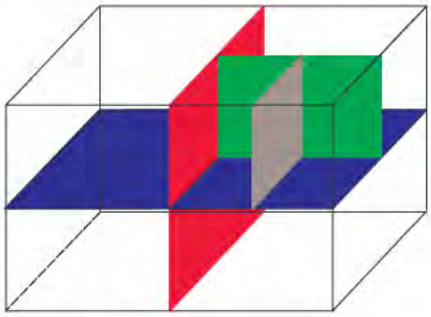
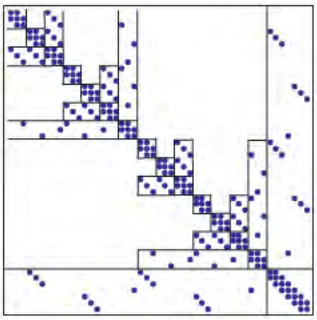
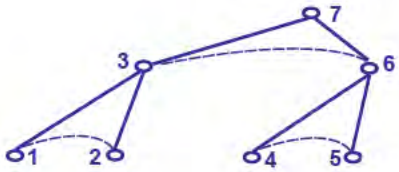


FIG. 1. *Nested dissection of a 3D mesh.*





(i) A matrix after nested dissection (ii) The corresponding assembly tree

FIG. 2. *A matrix pattern after nested dissection and the corresponding assembly tree for the separators. (For clarity, the matrix is taken from a 2D example and does not necessarily corresponds to Figure 1.)*

Then reconstruct the block columns corresponding to the nodes at level $\mathbf{l}$ of $\mathbf{T}$. We use a term of disjoint separators.

DEFINITION 2.1. *Two separators* $\mathbf{i}$ *and* $\mathbf{j}$ *are disjoint if they are not connected, or* $A|_{\mathbf{s_i}\times\mathbf{s_j}} = 0$, *or* $\mathbf{s_i}$ *and* $\mathbf{s_j}$ *do not include any graph points which are neighbors of* G.

Due to nested dissection, any nodes $\mathbf{i}$ and $\mathbf{j} = \mathrm{sib}(\mathbf{i})$ are disjoint. Similarly, this also holds for all the nodes at level $\mathbf{l}$ of $\mathbf{T}$. For simplicity, assume $n^{(\mathbf{l})} = |\mathbf{s_i}|$ for all nodes $\mathbf{i}$ at level $\mathbf{l}$. (Otherwise, we will attach appropriate zero columns to the identity matrices in $V^{(\mathbf{l})}$ below.) Let $V^{(\mathbf{l})}$ be an $n \times n^{(\mathbf{l})}$ skinny matrix, which is a zero matrix except

$$V^{(\mathbf{l})}|_{\mathbf{s_i}} = I \quad (\mathbf{i}\text{: at level } \mathbf{l}).$$

Then compute matrix-vector products

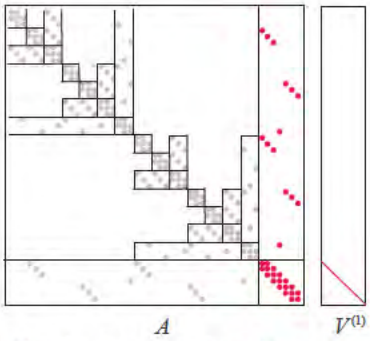
$$W = AV^{(\mathbf{l})}.$$

(This should come from a user-supplied matrix-vector product evaluation, from which A is to be reconstructed.) Clearly,

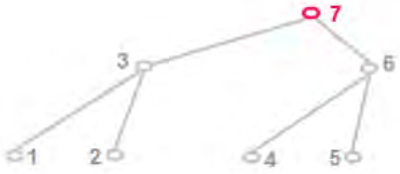
$$A|_{\mathbf{s_j}\times\mathbf{s_i}} = W|_{\mathbf{s_j}} \quad (\mathbf{i} \in \mathcal{N}_{\mathbf{j}}).$$

See Figure 3(iii)–(vi).

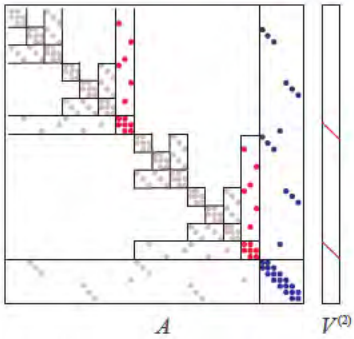
This basic scheme may need the multiplication of A with the matrices $V^{(\mathbf{l})}$ with the column sizes as large as the separator size. Thus, the overall number of matrix-vector multiplication may be large, especially for low-dimensional problems, as verified by the following lemma.



(i) Blocks correspond to the top level

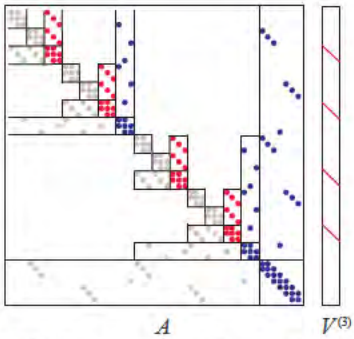


(ii) Top level nodes in T

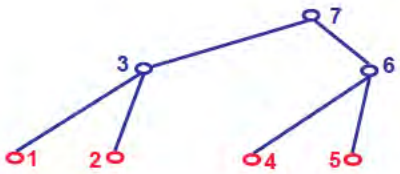


(iii) Blocks correspond to the 2nd level

(iv) 2nd level nodes in T



(v) Blocks correspond to the 3rd level



(vi) 3rd level nodes in T

FIG. 3. *Reconstructing the blocks corresponding to the first three levels of* $\mathbf{T}$. *(Other nodes and blocks to be reconstructed are grayed out.)*

LEMMA 2.2. *Assume the (one-layer) nested dissection is applied to a d-dimensional* $N \times N \times \cdots \times N$ *mesh. Then it needs* $O(N^{d-1})$ *matrix-vector products to reconstruct the matrix* A.

Proof. Let $\mathrm{l}_{\max} = O(\log(N))$ be the total number of levels in the nested dissection ordering (and in the assembly tree $\mathbf{T}$). Then the total number of matrix-vector products needed is

$$\sum_{\mathrm{l}=1}^{\mathrm{l}_{\max}} 2^{(d-1)\lfloor(\mathrm{l}_{\max}-\mathrm{l})/d\rfloor} = O(2^{(d-1)\mathrm{l}_{\max}}) = O(N^{d-1}).$$

□

For example, for 2D and 3D problems, $O(N) = O(n^{1/2})$ and $O(N^2) = O(n^{2/3})$ matrix-vector multiplications are needed, respectively. For sparse problems, if we assume each matrix-vector

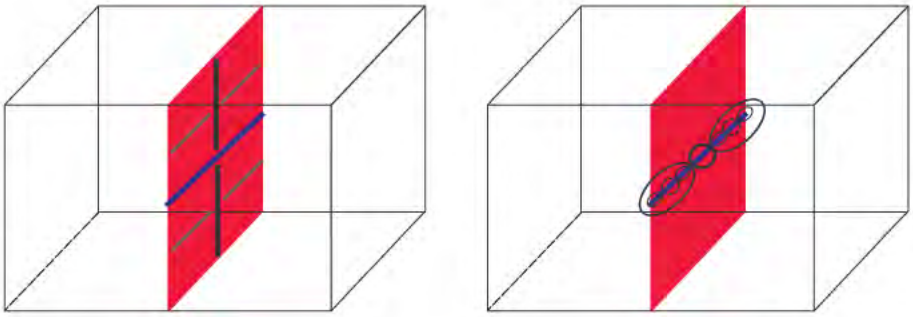
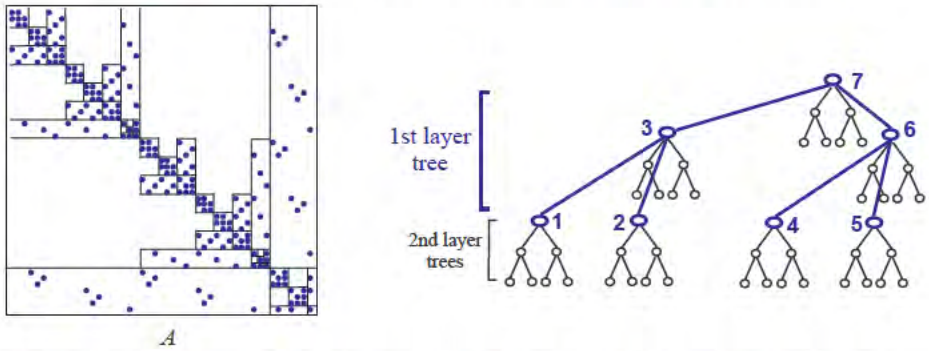


FIG. 4. *Two additional layers in the multi-nested dissection ordering for Figure 1.*



(i) A matrix after multi-nested dissection (ii) The corresponding multi-assembly tree

FIG. 5. *A matrix pattern after multi-nested dissection and the corresponding assembly tree for the separators, as compared with Figure 2. (For the illustration purpose, the second layer trees have more levels than what the matrix blocks actually have.)*

multiplication costs $O(n)$ flops, the cost becomes $O(n^{3/2})$ in 2D and $O(n^{5/3})$ in 3D. In 2D, the cost has the same order as that of the direct factorization. In 3D, it is slightly lower, but is still higher than certain structured solvers [28].

3. Superfast sparse reconstruction and multi-nested dissection. Then we present a superfast sparse reconstruction scheme which can reduce the cost in Lemma 2.2 to only $O(\log^d N)$.

3.1. Multi-nested dissection. We establish a strategy of multi-layer nested dissection, or *multi-nested dissection* (MND). That is, we extend ND to multiple layers. For example, for a 3D mesh G, the top layer ND uses separators which correspond to 2D graphs. Such 2D separators are then further ordered with ND repeatedly. In general, for a d-dimensional mesh, a separator at each inner layer is one dimension less than an outer layer one. The bottom layer or innermost layer ND is applied to 1D lines.

Our MND version for a 3D mesh is illustrated in Figure 4 with three layers of ND. In fact, the two-layer ND has been used in [25] for factorizing 3D discretized matrices, and is shown to be useful in improving certain rank structures in [5]. Another major benefit of MND is to enable us to conveniently reconstruct the blocks (especially the diagonal blocks) of A as in the next subsection.

A multi-layer assembly tree or *multi-assembly tree* can be defined. That is, an outer layer assembly tree corresponds to the outer layer ND. Each node of the outer layer tree is also associated with an inner layer tree for the ND (or MND) of the separator. We illustrate a simple example in Figure 5 after MND, as compared with Figure 2.

To consider the efficient reconstruction of A, it suffices to look at a 2D mesh G and the corresponding matrix and multi-assembly tree, just like those in Figure 2. The idea can be naturally

TABLE 3.1
Notation used in multi-nested dissection and our matrix reconstruction.

Symbol	Meaning
$\mathbf{T}$	The first (or outer) layer assembly tree
$\mathbf{i}$	A node (or separator) of T
$\mathbf{l}$	A level of $\mathbf{T}$, with root($\mathbf{T}$) at level $\mathbf{l}=1$
$\mathbf{l}_{\max}$	The maximum number of levels in $\mathbf{T}$
$\mathbf{s_i}$	The set of all mesh points corresponding to a first layer node $\mathbf{i}$
$\mathbf{T_i}$	The assembly tree in the second (or inner) layer corresponding to node $\mathbf{i}$ of $\mathbf{T}$
i	A node of $\mathbf{T_i}$
l	A level of $\mathbf{T_i}$ for certain $\mathbf{i}$
$l_{\max}(\mathbf{T_i})$	The maximum number of levels of $\mathbf{T_i}$
$(\mathbf{l},l)$	A level $\mathbf{l}$ of T_i for all nodes i at level $\mathbf{l}$ of $\mathbf{T}$
$(\mathbf{l},l)_{\max}$	The maximum number of levels of all $\mathbf{T_i}$ for i at level $\mathbf{l}$
$\mathbf{s}_{\mathbf{i},i}$	The set of all mesh points corresponding to a node i of $\mathbf{T_i}$
$n^{(\mathbf{l},l)}$	$\max_{i:\text{ node at level }(\mathbf{l},l)}\{\lvert\mathbf{s}_{\mathbf{i},i}\rvert\}$
$V^{(\mathbf{l},l)}$	The skinny matrix used to recover the separators at level l of $\mathbf{T_i}$ for all $\mathbf{i}$ at level $\mathbf{l}$ of $\mathbf{T}$
$\mathcal{N}_{\mathbf{i},i}$	$\{j \mid j$ is an ancestor of i in $\mathbf{T_i}$ and $A\lvert_{\mathbf{s}_{\mathbf{i},i}\times\mathbf{s}_{\mathbf{i},j}}\neq 0\}$

generalized to higher dimensions. To distinguish the notation among different layers of the multi-assembly tree, we define or restate some notation in Table 3.1.

3.2. Diagonal block reconstruction. We first consider the reconstruction of the diagonal blocks corresponding to the first layer nodes $\mathbf{i}$ of $\mathbf{T}$. For the node $\mathbf{i}=\text{root}(\mathbf{T})$ at level $\mathbf{l}=1$, we can simply apply the basic reconstruction scheme in Section 2 to $A|_{\mathbf{s_i}\times\mathbf{s_i}}$. See Figure 6(i)–(ii).

Clearly, all the nodes $\mathbf{i}$ at level $\mathbf{l}>1$ are disjoint and thus $A|_{\mathbf{s_i}\times\mathbf{s_i}}$ can be reconstructed simultaneously. This is done for all the nodes of the inner trees $\mathbf{T_i}$ at the same levels. That is, let V be an $n\times n^{(\mathbf{l},l)}$ skinny matrix, which is a zero matrix except

$$V^{(\mathbf{l},l)}|_{\mathbf{s}_{\mathbf{i},i}} = I \quad (i\text{: at level } (\mathbf{l},l)). \tag{3.1}$$

Then compute matrix-vector products

$$W = AV^{(\mathbf{l},l)}.$$

Clearly, for all nodes $\mathbf{i}$ at level $\mathbf{l}$ of $\mathbf{T}$,

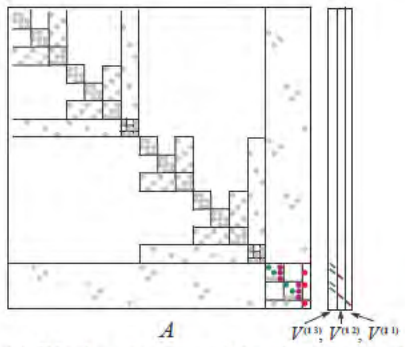
$$A|_{\mathbf{s}_{\mathbf{i},j}\times\mathbf{s}_{\mathbf{i},i}} = W|_{\mathbf{s}_{\mathbf{i},i}} \quad (i\in\mathcal{N}_{\mathbf{i},j}).$$

See Figure 3(iii)–(vi).

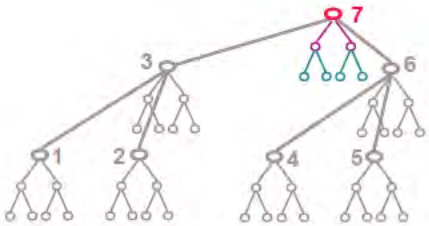
That is, we simultaneously visit all the nodes at level $(\mathbf{l},l)$, which are across multiple inner layer tress $\mathbf{T_i}$. This can be recursively applied to the multiple layers for high dimensional problems, and the total number of matrix-vector products required to recover all $A|_{\mathbf{s_i}\times\mathbf{s_i}}$ blocks is given below.

LEMMA 3.1. *Assume the (d-layer) multi-nested dissection is applied to a d-dimensional $N\times N\times\cdots\times N$ mesh. Then it needs $O(\log^d N)$ matrix-vector products to reconstruct all the diagonal blocks $A|_{\mathbf{s_i}\times\mathbf{s_i}}$ of the matrix A.*

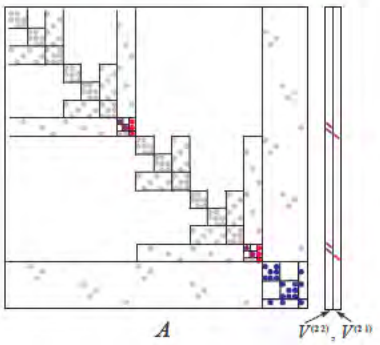
3.3. Off-diagonal block reconstruction and compact probing. The diagonal block reconstruction in the previous subsection follows a general idea of probing, where a small number of vectors are used to extract certain information with matrix-vector products (see, e.g., [23]). This can be extended to a method which we call *compact probing*.



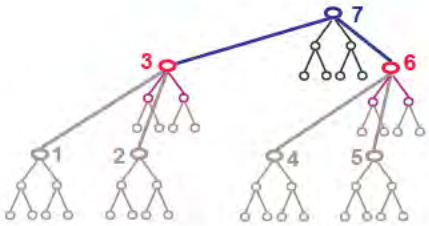
(i) Blocks correspond to the top level



(ii) Top level nodes in T



(iii) Blocks correspond to the 2nd level



(iv) 2nd level nodes in T

FIG. 6. *Reconstructing the blocks corresponding to the first three levels of* $\mathbf{T}$. *(Other nodes and blocks to be reconstructed are grayed out.)*

DEFINITION 3.2. *A compact probing is a method for finding certain skinny matrices* V *satisfying the following conditions in order to extract certain information (e.g., matrix entries) of a matrix* A*:*

- *The nonzero blocks of* V *are only identity matrices;*
- *The number of such matrices* V *is as small as possible.*

When the blocks to be reconstructed are disjoint as in the previous subsection, it is convenient to select the compact probing vectors $V^{(1,l)}$. For the off-diagonal blocks of A, the blocks are generally not disjoint, or have overlaps (in their column indices). In the latter case, compact probing can be illustrated through a simple example:

$$\begin{pmatrix} A_1 & A_2 \\ & A_3 \quad A_4 \end{pmatrix}, \tag{3.2}$$

which we try to reconstruct with matrix-vector products. Assume each nonzero block in (3.2) has size $n_1 \times n_1$. A straightforward way is to multiply (3.2) by an identity matrix of size $3n_1$. However, since

$$\begin{pmatrix} A_1 & A_2 \\ & A_3 \quad A_4 \end{pmatrix} \begin{pmatrix} I & \\ & I \\ I & \end{pmatrix} = \begin{pmatrix} A_1 & A_2 \\ A_4 & A_3 \end{pmatrix},$$

we can thus multiply (3.2) by a compact probing matrix with $2n_1$ columns instead.

This example suggests that we can simultaneously reconstruct the disjoint subblocks in an off-diagonal block. The blocks with overlaps in their column indices are reconstructed with separate probing vectors.

Thus, to recover the off-diagonal blocks $A|_{\mathbf{s_j}\times\mathbf{s_i}}$ with $\mathbf{i}\in\mathcal{N}_\mathbf{j}$ for all $\mathbf{j}$, we cycle through all the descendants $\mathbf{j}$ of $\mathbf{i}$ in a bottom-up order to identify the disjoint blocks. Due to the local connectivity assumption in Section 1, each mesh point in separator $\mathbf{i}$ is only connected to finite number of mesh points in other separators. Thus, the overlaps are generally small, especially between the separators at the same level. The removal of certain blocks with overlaps at one cycle creates new disjoint blocks. Then this is repeated until all blocks are reconstructed.

The major steps for finding the probing vectors V (associated with each node $\mathbf{i}$ of $\mathbf{T}$) are listed as follows. For each descendent $\mathbf{j}$ of $\mathbf{i}$,

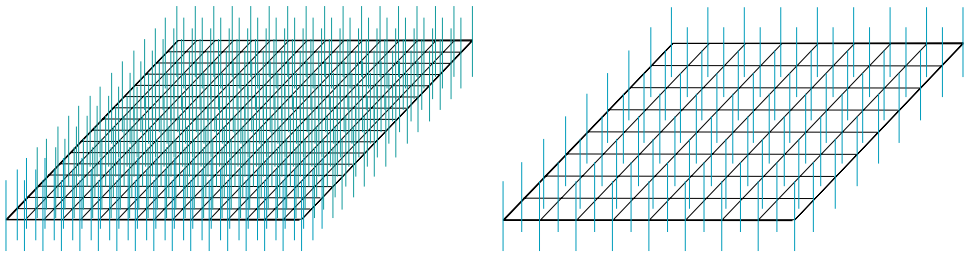
1. find the nonzero column indices in $A|_{\mathbf{s_j}\times\mathbf{s_i}}$;
2. find the nonzero column indices $\hat{\mathbf{c}}_{\mathbf{i,j}}$ and $\mathbf{c}_{\mathbf{i,j}}$ in $A|_{\mathbf{s_j}\times\mathbf{s_i}}$ that have and have no overlaps with any other blocks in $A|_{:\times\mathbf{s_i}}$, respectively;
3. find the nonzero row indices $\hat{\mathbf{r}}_{\mathbf{i,j}}$ and $\mathbf{r}_{\mathbf{i,j}}$ in $A|_{\mathbf{s_j}\times\mathbf{s_i}}$ that have and have no overlaps with any other blocks in $A|_{:\times\mathbf{s_i}}$, respectively;
4. update the nonzero column indices $\hat{\mathbf{g}}_\mathbf{i}$ and $\mathbf{g}_\mathbf{i}$ in $A|_{:\times\mathbf{s_i}}$ that have and have no overlaps, respectively;
5. set $V|_{\mathbf{c_{i,j}}} = I$.

Figure 7 illustrates how the connections between $\mathbf{i}$ and its descendants are removed or peeled off in the cycling. That is, in an ideal situation, the pieces in $\mathbf{i}$ to be removed at each level happens to match those at the multiple levels of $\mathbf{T_i}$. The details are given in Algorithm 1. For convenience, the following simple routines are used:

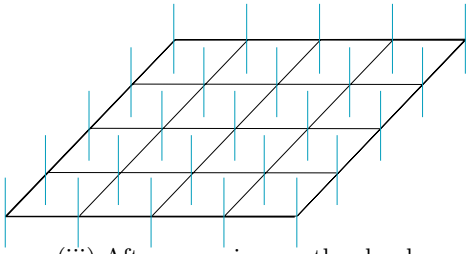
$W = \mathtt{mv}(V)$ – evaluating the product of A with V;

$\mathbf{r} = \mathtt{any}(\mathbf{s_i}, \mathbf{s_j}, d)$ – the nonzero row or column indices of $A|_{\mathbf{s_i}\times\mathbf{s_j}}$ for $d = 2$ or 1, as used in Matlab;

$[\mathbf{c}, \hat{\mathbf{c}}] = \mathtt{setdiff}(\mathbf{s_i}, \mathbf{s_j})$ – the set $\mathbf{c}$ that is in $\mathbf{s_i}$ but not $\mathbf{s_j}$, and the set $\hat{\mathbf{c}}$ that is in both $\mathbf{s_i}$ and $\mathbf{s_j}$, similarly to the routine in Matlab.



(i) Connections between $\mathbf{i}$ and its descendants (ii) After removing one level



(iii) After removing another level

FIG. 7. *Cycling through the descendants of node* $\mathbf{i}$ *of* $\mathbf{T}$ *to remove the connections between disjoint pieces of* $\mathbf{i}$ *and* $\mathbf{i}$*'s descendants.*

Algorithm 1 Finding the probing vectors due to the contribution of a node $\mathbf{i}$

procedure $[V, \mathbf{r_{i,j}}, \hat{\mathbf{r}}_{\mathbf{i,j}}, \mathbf{c_{i,j}}, \hat{\mathbf{g}}_{\mathbf{i}}]$ = `compactprobe`(G,$\mathbf{T}$,$\mathbf{i}$)
  for $\tilde{\mathbf{l}} = \mathbf{l}_{\max}, \mathbf{l}_{\max} - 1, \ldots, \mathbf{l}$ **do** ▷ *Cycling all the descendants of* $\mathbf{i}$ *at level* $\mathbf{l}$
    $V \leftarrow 0$
    for node $\mathbf{j}$ at level $\tilde{\mathbf{l}}$ of $\mathbf{T_i}$ **do**
      $t \leftarrow \mathtt{any}(\mathbf{s_j}, \mathbf{s_i}, 1)$ ▷ *With* G
      $[\mathbf{c_{i,j}}, \hat{\mathbf{c}}_{\mathbf{i,j}}] \leftarrow \mathtt{setdiff}(t, \mathbf{g_i})$
      $\hat{\mathbf{r}}_{\mathbf{i,j}} \leftarrow \mathtt{any}(\mathbf{s_j}, \hat{\mathbf{c}}_{\mathbf{i,j}}, 2)$ ▷ *With* G
      $\mathbf{r_{i,j}} \leftarrow \mathtt{setdiff}(\mathbf{s_j}, \tilde{\mathbf{s}}_{\mathbf{j}})$
      $\mathbf{g_i} \leftarrow [\mathbf{g_i}, \mathbf{c_{i,j}}]$
      $t \leftarrow \mathtt{setdiff}(\mathbf{s_i}, \mathbf{c_{i,j}})$
      $\hat{\mathbf{g}}_{\mathbf{i}} \leftarrow [\hat{\mathbf{g}}_{\mathbf{i}}, t]$
      $V|_{\mathbf{c_{i,j}}} \leftarrow I$
    end for
  end for
end procedure

After we find V, the reconstruction of the off-diagonal blocks of A from $W = \mathtt{mv}(V)$ can be conveniently done. Again, we cycle through the descendants of $\mathbf{i}$. The details are shown in the algorithm in the next subsection. The cost is given as follows.

LEMMA 3.3. *Assume the (d-layer) multi-nested dissection is applied to a d-dimensional* $N \times N \times \cdots \times N$ *mesh, and assume the pieces in a top layer separator* $\mathbf{i}$ *to be removed at each level happens to match those at the multiple levels of* $\mathbf{T_i}$*. Then it needs* $O(\log^d N)$ *matrix-vector products to reconstruct all the off-diagonal blocks in* $A|_{:\times\mathbf{s_i}}$ *of the matrix A.*

REMARK 3.1. For some cases, the off-diagonal blocks may also be treated as low-rank blocks, and can be reconstructed with random probing vectors as in [14, 32].

3.4. Algorithm summary and superfast and fast matrix reconstruction. We summarize the method in Algorithm 2. Again, for notational convenience, we show the algorithm with a two-layer assembly tree.

About the total number of matrix-vector products needed, we have the following proposition.

PROPOSITION 3.4. *Assume the conditions in Lemmas 3.1 and 3.3 hold. Then it needs* $O(\log^d N)$ *flops to recover the matrix A.*

Due to this result, we say the reconstruction is *superfast*. In fact, sometimes, we can simplify the scheme slightly. For example, for a 3D problem, we can apply a two layer MND instead. This can help simplify the coding. Similarly, we can use a $(d-1)$-layer MND for a d-dimensional problem. Then it can be verified that the total number of matrix-vector products $O(N \log^{d-1} N)$. For 3D problems, assume each matrix-vector multiplication costs $O(n)$ flops, then the total cost is $O(n^{4/3} \log n)$, which is comparable to the structured factorization cost in [28, 29]. Thus, this simplified reconstruction is said to be a *fast* scheme.

REMARK 3.2. Similar to the methods in [28, 29, 30], we may use a switching level $\mathbf{l_s}$ so as to avoid working on small separators. That is, below $\mathbf{l_s}$, we follow the strategy in the fast reconstruction, and above $\mathbf{l_s}$, we follow the superfast version.

REMARK 3.3. Although our discussions and pictorial illustrates use regular meshes, the discussions are applicable to more general meshes, which may make the analysis more complicated.

4. Direct factorization and factorization update for varying parameters. After the reconstruction of A, we can then directly factorize it, or construct a preconditioner based on it. Here, we are interested in fast solvers based on rank structures, as developed in [28, 29, 30]. We

Algorithm 2 Matrix reconstruction from matrix-vector products

procedure $A = \texttt{mv2mat}(G,\mathbf{T},\texttt{mv})$
  for $\mathbf{l} = 1, 2, \ldots, \mathbf{l}_{\max}$ **do** ▷ *First/outer layer loop*
    for $l = 1, 2, \ldots, (\mathbf{l}, l)_{\max}$ **do** ▷ *Diagonal block reconstruction*
      $V \leftarrow 0$ ▷ *Initializing probing vectors*
      for node i at level $(\mathbf{l}, l)$ **do**
        $V|_{\mathbf{s}_{i,i}} \leftarrow I$ ▷ *As in (3.1)*
      end for
      $W \leftarrow \texttt{mv}(V)$ ▷ *Multiplication of A with V*
      for node i at level $(\mathbf{l}, l)$ **do**
        for node j so that $i \in \mathcal{N}_{\mathbf{i},j}$ **do**
          $A|_{\mathbf{s}_{i,j}\times\mathbf{s}_{i,i}} \leftarrow W|_{\mathbf{s}_{i,i}}$ ▷ *All the blocks of $A|_{\mathbf{s}_\mathbf{i}\times\mathbf{s}_\mathbf{i}}$*
        end for
      end for
    end for
  end for
  for $\mathbf{l} = \mathbf{l}_{\max}, \mathbf{l}_{\max}-1, \ldots, 1$ **do** ▷ *Off-diagonal block reconstruction*
    for node $\mathbf{i}$ at level $\mathbf{l}$ **do**
      $[V_\mathbf{i}, \mathbf{r}_{\mathbf{i},\mathbf{j}}, \hat{\mathbf{r}}_{\mathbf{i},\mathbf{j}}, \mathbf{c}_{\mathbf{i},\mathbf{j}}, \hat{\mathbf{g}}_\mathbf{i}] = \texttt{compactprobe}(G, \mathbf{T}, \mathbf{i})$
      $V \leftarrow V + V_\mathbf{i}$ ▷ *Putting more nonzeros in (not an actual addition)*
    end for
    $W \leftarrow \texttt{mv}(V)$ ▷ *Multiplication of A with V*
    for node $\mathbf{i}$ at level $\mathbf{l}$ **do**
      for $\tilde{\mathbf{l}} = \mathbf{l}_{\max}, \mathbf{l}_{\max}-1, \ldots, \mathbf{l}$ **do**
        for node $\mathbf{j}$ at level $\tilde{\mathbf{l}}$ of $\mathbf{T}_\mathbf{i}$ **do**
          $A|_{\mathbf{r}_{\mathbf{i},\mathbf{j}}\times\mathbf{c}_{\mathbf{i},\mathbf{j}}} \leftarrow W|_{\mathbf{r}_{\mathbf{i},\mathbf{j}}}$ ▷ *An off-diagonal block in $A|_{:\times\mathbf{s}_\mathbf{i}}$*
          $\hat{\mathbf{r}}_{\mathbf{i},\mathrm{par}(\mathbf{j})} \leftarrow \hat{\mathbf{r}}_{\mathbf{i},\mathrm{par}(\mathbf{j})} \cup \hat{\mathbf{r}}_{\mathbf{i},\mathbf{j}}$
        end for
      end for
      $\hat{\mathbf{r}}_{\mathbf{i},\mathbf{i}} \leftarrow \hat{\mathbf{g}}_\mathbf{i}$
    end for
    for node $\mathbf{i}$ at level $\mathbf{l}$ **do** ▷ *The remaining nonzero entries*
      $[\mathbf{r}_{\mathbf{i},\mathbf{i}}, \mathbf{c}_{\mathbf{i},\mathbf{j}}] \leftarrow \mathrm{any}\Big(\bigcup_{\mathbf{j}:\text{ child of }\mathbf{i}} \hat{\mathbf{r}}_{\mathbf{i},\mathbf{j}},\ \hat{\mathbf{r}}_{\mathbf{i},\mathbf{i}}\Big)$
      $V|_{\mathbf{c}_{\mathbf{i},\mathbf{j}}} \leftarrow I$
    end for
    $W \leftarrow \texttt{mv}(V)$
    for node $\mathbf{i}$ at level $\mathbf{l}$ **do**
      $A|_{\mathbf{r}_{\mathbf{i},\mathbf{j}}\times\mathbf{c}_{\mathbf{i},\mathbf{j}}} \leftarrow W|_{\mathbf{r}_{\mathbf{i},\mathbf{j}}}$
    end for
  end for
end procedure

focus on the method in [29] due to the efficiency and flexibility. The method combines three types of techniques:

1. multifrontal factorizations [10] of large sparse matrices after nested dissection;
2. hierarchically semiseparable (HSS) [3, 4, 31] and rank structured matrices;
3. randomized sampling [14, 16] for fast matrix compression.

We briefly review the major steps of the method as follows, following an assembly tree:

1. use randomized sampling as in [17, 19, 32] to compute an HSS approximation to the intermediate dense matrices (called frontal matrices);

2. partially factorize the HSS frontal matrix and form the Schur complement (called an update matrix);
3. pass the product of the update matrix with some random vectors to the parent node of the assembly tree;
4. assemble the child matrix-vector products together in an operation consisting of permutation, expansion, and addition (called an extend-add operation);
5. form selected entries of the parent frontal matrix as necessary.

When certain conditions are satisfied, some discretized matrices on 2D ($N \times N$) and 3D ($N \times N \times N$) meshes can be factorized in about $O(n)$ and $O(n^{4/3})$ flops, respectively, where $n = N^2$ in 2D and N^3 in 3D. The solution costs and memory requirements about linear in n for both 2D and 3D.

Here, we would like to emphasize some additional benefits. That is, it is possible to (at least partially) update the factorization when the matrix is varied due to some parameters (e.g., shifts sI). There are two major reasons.

1. Since the matrix A is reconstructed from matrix-vector products (and most entries stay the same), and the method in [29] also uses matrix-vector products for intermediate operations, we can quickly update the matrix-vectors products to update A or certain intermediate matrices.
2. If A is changed to $A - sI$ by sI due to a shift or a frequency, it is even possible to (at least partially) update the factorization. For example, in the structured sparse factorizations of A, we may keep certain bottom level separators to be large, so that their corresponding frontal matrices are approximated by large HSS matrices. HSS matrices have a very useful feature in that, if the diagonal is varied by sI, the ULV-type factorization [4] can be updated, and this saves about 60% of the work [27].

Thus, depending on the actual requirements, different variations of the solvers can be used, together with the fast or superfast sparse reconstruction.

5. Numerical experiments. We focus on the performance of the sparse reconstruction, since the sparse factorization performance can be found in [29]. Clearly, the adjacency graph instead of the nature of the problem is our main concern. Thus, we only show the stencil or mesh, and skip the descriptions of the actual PDEs.

First, we look at some 2D problems discretized on 7-point finite element meshes with the function `grid7` in [12]. We report the numbers of matrix-vector products ($\#_{\text{mv}}$) required to fully reconstruct the discretized matrix A in Table 6.1. $\#_{\text{mv}}$ is also compared with the line $O(\log^2 N)$ in Figure 8. The line for $\#_{\text{mv}}$ is not very far from that for $O(\log^2 N)$, and we expect them to be consistent for large n.

Next, consider some 3D problems discretized on tetrahedral finite element meshes with the function `grid3dt` in [12]. We similarly report $\#_{\text{mv}}$ in Table 6.2. Again, when n increases by a factor of 8, the growth of $\#_{\text{mv}}$ is quite slow.

6. Conclusions. We primarily demonstrated a sparse reconstruction method based on matrix-vector products. The method is superfast or needs only $O(N \log^d N)$ matrix-vector products. Our near future work is to further consider the update of the factorization for varying parameters, without using large bottom level separators. This is especially useful for eigenvalue solutions with shifts or for problems with multiple frequencies as in full waveform inversion in seismic imaging.

Acknowledgments. The research of the author was supported in part by NSF grants DMS-1115572 and CHE-0957024.

TABLE 6.1
Numbers of matrix-vector products ($\#_{\text{mv}}$) required to fully reconstruct the discretized matrix A for the 7-point finite different meshes in 2D, where $\mathbf{l}_{\max} - \mathbf{l}_s = 5$.

n	31^2	63^2	127^2	255^2	511^2
$\mathbf{l}_{\max}$	7	9	11	13	15
$\#_{\text{mv}}$	121	213	325	470	632

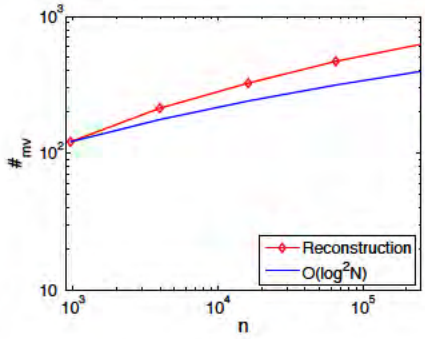


FIG. 8. *Numbers of matrix-vector products ($\#_{\text{mv}}$) as in Table 6.1.*

TABLE 6.2
Numbers of matrix-vector products ($\#_{\text{mv}}$) required to fully reconstruct the discretized matrix A for the tetrahedral finite element meshes in 3D.

n	8^3	16^3	32^3	64^3
$\mathbf{l}_{\max}$	4	7	10	13
$\mathbf{l}_{\max} - \mathbf{l}_s$	4	5	5	5
$\#_{\text{mv}}$	164	724	2,404	4,783

REFERENCES

[1] Z. BAI, J. DEMMEL, J. DONGARRA, A. RUHE, AND H. VORST, EDS., *Templates for the solution of algebraic eigenvalue problems: a practical guide*, SIAM, Philadelphia, 2000.

[2] Z. CAI AND S. ZHANG, *Recovery-based error estimator for interface problems: conforming linear elements*, SIAM J. Numer. Anal., 47 (2009), pp. 2132–2156.

[3] 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.

[4] 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.

[5] 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.

[6] Z. CHEN, L. WANG, AND W. ZHENG, *An adaptive multilevel method for time-harmonic Maxwell equations with singularities*, SIAM J. Sci. Comput., 29 (2007), pp. 118–138.

[7] T. A. DAVIS, *Direct Methods for Sparse Linear Systems (Fundamentals of Algorithms)*, SIAM, 2006.

[8] J. W. DEMMEL, *Applied Numerical Linear Algebra*, SIAM, Philadelphia, PA, 1997.

[9] I. S. DUFF, A. M. ERISMAN, J. K. REID, *Direct Methods for Sparse Matrices (Monographs on Numerical Analysis)*, Oxford University Press, USA, 1987.

[10] 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.

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

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

[13] G. H. GOLUB AND C. V. LOAN, *Matrix Computations*, The Johns Hopkins University Press, Baltimore, MD, 4rd edition, 2013.

[14] N. Halko, P.G. Martinsson, and J. Tropp, *Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions*, SIAM Review, 53 (2011), pp. 217–288.
[15] 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.
[16] E. Liberty, F. Woolfe, P. G. Martinsson, V. Rokhlin, and M. Tygert, *Randomized algorithms for the low-rank approximation of matrices*, Proc. Natl. Acad. Sci. USA, 104 (2007), pp. 20167–20172.
[17] L. Lin, J. Lu, and L. Ying, *Fast construction of hierarchical matrix representation from matrix-vector multiplication*, J. Comput. Phys., 230 (2011), pp. 4071–4087.
[18] J. W. H. Liu, *The multifrontal method for sparse matrix solution: Theory and practice*, SIAM Review, 34 (1992), pp. 82–109.
[19] P. G. Martinsson, *A fast randomized algorithm for computing a hierarchically semiseparable representation of a matrix*, SIAM. J. Matrix Anal. Appl., 32 (2011), pp. 1251–1274.
[20] G. L. Miller, S.-H. Teng, W. Thurston, and S. A. Vavasis, *Automatic mesh partitioning, in Sparse Matrix Computations: Graph Theory Issues and Algorithms*, A. George, J. Gilbert, and J. Liu, eds., IMA Volumes in Mathematics and its Applications 56, Springer-Verlag, New York, (1993), pp. 57–84.
[21] G. L. Miller, S.-H. Teng, W. Thurston, and S. Vavasis, *Geometric separators for finite element meshes*, SIAM J. Sci. Comput., 19 (1998), pp. 364–386.
[22] J. Shen and H. Yu, *Efficient spectral sparse grid methods and applications to high dimensional elliptic problems*, SIAM J. Sci. Comput., 32 (2010), pp. 3228–3250.
[23] J. M. Tang and Y. Saad, *A probing method for computing the diagonal of a matrix inverse*, Numer. Linear Algebra Appl., 19 (2012), pp. 485–501.
[24] S. Teng, *Fast nested dissection for finite element meshes*, SIAM J. Matrix Anal. Appl., 18 (1997), pp. 552–565.
[25] S. Wang, M. V. de Hoop, and J. Xia, *On 3D modeling of seismic wave propagation via a structured parallel multifrontal direct Helmholtz solver*, Geophys. Prospect., 59 (2011), pp. 857–873.
[26] H. Wei, L. Chen, Y. Huang, and B. Zheng, *Adaptive mesh refinement and superconvergence for two dimensional interface problems*, Submitted, 2012.
[27] Y. Xi, J. Xia, and R. Chan, *A fast structured eigensolver for symmetric Toeplitz matrices and more via adaptive randomized sampling*, SIAM J. Matrix Anal. Appl., submitted, (2013).
[28] J. Xia, *Efficient structured multifrontal factorization for general large sparse matrices*, SIAM J. Sci. Comput., 35 (2013), pp. A832–A860.
[29] J. Xia, *Randomized sparse direct solvers*, SIAM J. Matrix Anal. Appl., 34 (2013), pp. 197–227.
[30] 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.
[31] 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.
[32] J. Xia, Y. Xi, and M. Gu, *A superfast structured solver for Toeplitz linear systems via randomized sampling*, SIAM J. Matrix Anal. Appl., 33 (2012), pp. 837–858.