Proceedings of the Project Review, Geo-Mathematical Imaging Group (Purdue University, West Lafayette IN), Vol. 1 (2012) pp. 81-95.

MFRS: AN ALGORITHM FOR THE STRUCTURED MULTIFRONTAL SOLUTION OF LARGE SPARSE MATRICES VIA RANDOMIZED SAMPLING

SHEN WANG*, JIANLIN XIA†, MAARTEN V. DE HOOP‡, AND XIAOYE S. LI§

Abstract. This paper presents strategies for the development of an efficient algorithm (MFRS) for the direct solutions of large sparse linear systems. The algorithm is based on a structured multifrontal method with randomized sampling. We propose data structures and access schemes for a type of rank structured matrices, called Hierarchically SemiSeparable (HSS) forms. A data tree structure is used for HSS matrices. Similar strategies are built into a multifrontal factorization method. This yields efficient nested data trees. The fast operations for certain important operations in the factorizations are shown.

Key words. multifrontal method, sparse direct solver, randomized sampling, HSS structure

1. Introduction. Large sparse linear systems arise frequently in scientific and engineering computations. Direct solutions of such systems are often found attractive due to their robustness and suitability for multiple right-hand sides. However, the main disadvantage of direct solutions is the fill-in, or loss of sparsity.

In recent years, rank structured direct methods are used to compute approximate direct solutions. Examples include $\mathcal{H}$- and $\mathcal{H}^2$-matrix methods [1, 13] and structured multifrontal methods [18, 14]. In particular, the structured multifrontal methods combine various sparse matrix techniques and structured matrix methods, and are suitable for parallel computing. A sparse matrix is first ordered with nested dissection [4] to reduce fill-in, and is then factorized with the multifrontal method [3]. In [18], the intermediate dense matrices, called frontal matrices and update matrices [3], are approximated by rank structured forms called hierarchically semiseparable (HSS) forms [2, 19]. However, this is generally complicated due to an operation to assemble matrices (called extend-add operation), and is limited to certain special problems. Simplifications are made in [14], where the update matrices or intermediate Schur complements are kept dense. This is significantly easier to implement.

Recently, a new structured multifrontal solver based on randomized sampling was proposed in [17]. This work employs randomized techniques to perform HSS matrix operations. The HSS structures are constructed following a scheme in [11]. Then HSS operations such as the extend-add operation are replaced by those in terms of certain matrix-vector products. This avoids complicated HSS matrix operations, and makes structured multifrontal methods more easily and generally applicable. Here, we focus on a series of techniques that utilize the method in [17]. We develop data structures for both randomized HSS matrix methods and randomized sparse solutions.

A data structure is designed for HSS matrices, called a data tree, Here the data is stored in a contiguous array, with pointers pointing to the locations of the individual blocks (called HSS generators) that define an HSS matrix. A similar data tree structure is used for the dense intermediate matrices of the multifrontal method. Thus, the structured solver uses two levels of nested data tree structures, which help the algorithm achieve high efficiency and scalability.

Two tall and skinny random matrices are generated, with row sizes equal to the size of the matrix. The intermediate factorization steps only need to access the memory allocated to these random matrices. One important operation used is to reconstruct certain entries of an HSS structure. This is expensive in a straightforward implementation. Here, we introduce a concept of data ownership. Row and column indices of the desired entries are grouped so as to reuse the data as much as possible.

*Department of Mathematics, Purdue University, West Lafayette, IN 47907.
†Department of Mathematics, Purdue University, West Lafayette, IN 47907.
‡Center for Computational and Applied Mathemematics, Purdue University, West Lafayette, IN 47907 (mdehoop@purdue.edu).
§Computational Research Division, Lawrence Berkeley National Laboratory (LBNL).

The remaining sections are organized as follows. Section 2 reviews structured multifrontal methods and related applications. Section 3 discusses techniques for efficiently handling HSS representations and performing randomized HSS construction. The ideas for utilizing an efficient randomized multifrontal algorithm is presented in detail in Section 4. Section 5 draws some conclusions.

2. Structured multifrontal solvers and their applications. In this section, we briefly review some structured multifrontal methods for the approximate direct solutions of large sparse matrices. Here, we focus on the approach adopted in [14, 16]. Applications of such methods to practical problems such as Helmholtz equations are also described.

2.1. The discretization of the Helmholtz equation. We consider the scalar Helmholtz equation in 3D:

$$\left[-\Delta - \frac{\omega^2}{v(x)^2}\right] u(x,\omega) = f(x,\omega), \quad x \in \mathbb{R}^3; \tag{2.1}$$

where Δ is the Laplace operator, ω is the angular frequency, $v(x)$ is the longitudinal pressure wave speed, $f(x,\omega)$ is the forcing term, and $u(x,\omega)$ denotes the time-harmonic pressure wavefield we want to solve.

We discretize equation (2.1) on a 3D rectangular mesh $N_1 \times N_2 \times N_3$ in which N_1, N_2 and N_3 denote the number of grid points in each spatial direction, resorting to the finite difference method equipped with the Perfect Matched Layer (PML) boundary condition (see [14]). This leads to the following linear systems of equations:

$$\mathcal{A}(\omega)\ \mathcal{X}(\omega) = \mathcal{B}(\omega), \tag{2.2}$$

where the number of unknowns is $N_1N_2N_3 \times N_1N_2N_3$. $\mathcal{A}(\omega)$, $\mathcal{X}(\omega)$ and $\mathcal{B}(\omega)$ are discretizations of the Helmholtz operator, pressure wavefield and the forcing term, respectively. We point out that the matrix $\mathcal{A}(\omega)$ is pattern symmetric but non-Hermitian, due to the incorporation of the absorbing boundary condition. Moreover, $\mathcal{A}(\omega)$ is ill-conditioned and indefinite especially in the scenario of high frequencies.

2.2. A structured multifrontal solver. To solve the matrix system (2.2) on a large 3D domain and for a large number of forcing terms, [14] proposed a massively parallel multifrontal direct solver imbedding a scalable Hierarchically Semiseparable (HSS) matrix solver [15].

They first carry out the nested dissection reordering of the global matrix $\mathcal{A}(\omega)$ by dividing upper level domains into lower level subdomains and separators recursively. The mesh points are reordered following a post-ordering tree structure called *assembly tree*, which is denoted as $\mathbb{T}$. We denote the total level of the assembly tree as lvl.

Secondly, after the nested dissection reordering, [14] conduct local partial LU factorizations, by forming *frontal matrices* $\mathcal{F}_k$ and computing *update matrices* $\mathcal{U}_k$ locally on each node k of the $\mathbb{T}$, by taking advantage of the multifrontal method introduced by [8]. The mathematical relations can be summarized in the following concise form:

$$\mathcal{F}_k = \begin{pmatrix} \mathcal{F}_{k,11} & \mathcal{F}_{k,12} \\ \mathcal{F}_{k,21} & \mathcal{F}_{k,22} \end{pmatrix} = \begin{cases} \begin{pmatrix} \mathcal{A}_{k,11} & \mathcal{A}_{k,12} \\ \mathcal{A}_{k,21} & 0 \end{pmatrix}, & \text{if } k \text{ is a leaf node;} \\ \begin{pmatrix} \mathcal{A}_{k,11} & \mathcal{A}_{k,12} \\ \mathcal{A}_{k,21} & 0 \end{pmatrix} + \left(\mathcal{U}_{d_1} \oplus \mathcal{U}_{d_2}\right), & \text{if } k \text{ is a non-leaf node;} \end{cases} \tag{2.3}$$

$$\mathcal{U}_k = \mathcal{F}_{k,22} - \mathcal{F}_{k,21}\ \mathcal{F}_{k,11}^{-1}\ \mathcal{F}_{k,12};$$

where $\mathcal{A}_k$ denotes the portion of the global matrix $\mathcal{A}(\omega)$ associated with the node k of the assembly tree. d_1 and d_2 are two children of the node k if k is non-leaf, satisfying $d_1 < d_2 < k$. The extend-add operation is denoted by $\oplus$.

In the third place, for the sake of memory saving and numerical complexity reduction, [15] proposed a parallel HSS technique to further compress off-diagonal blocks of each frontal matrix

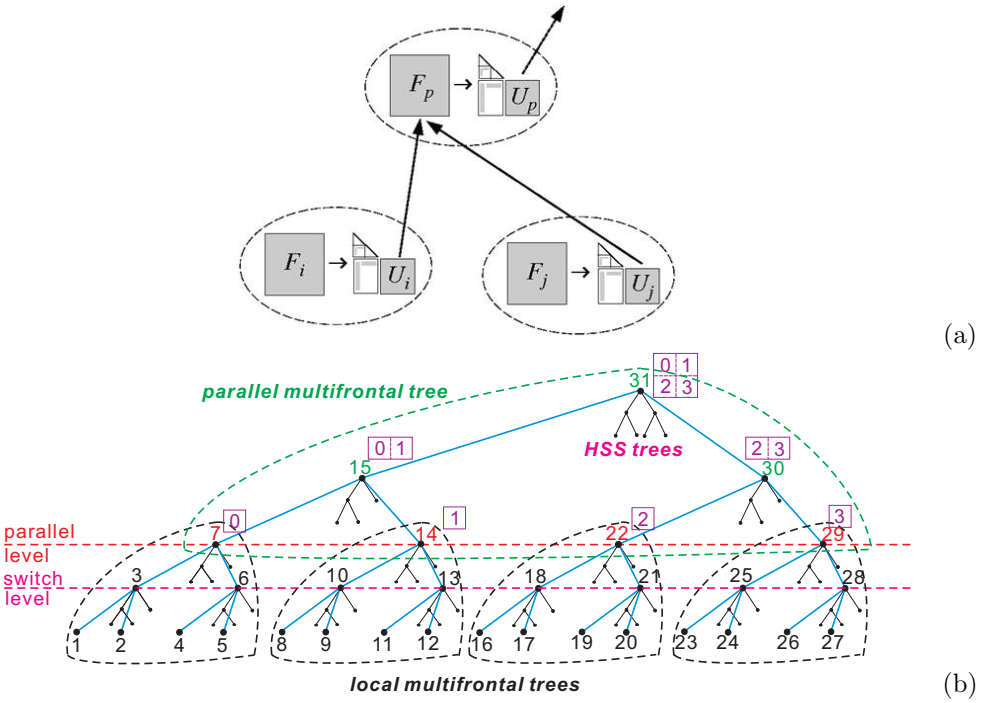

FIG. 1. *(a): the illustration of the HSS representation of local factors, without compressing update matrices; (b): the illustration of the conventional structured multifrontal direct solver proposed by [14] and [15].*

$\mathcal{F}_k$, by exploiting their low-rank properties. However, they only conduct HSS constructions and factorizations on $\mathcal{F}_{k,11}$, $\mathcal{F}_{k,12}$ and $\mathcal{F}_{k,21}$ blocks, leaving the dense Schur complement block $\mathcal{F}_{k,22}$ untouched. Thus the extend-add operation still applies to dense matrices, which is illustrated by Figure 1(a). They introduced another level called *switch level*, denoted as *slvl*, above which partial HSS constructions and factorizations are carried out. Figure 1(b) illustrates the entire classical structured multifrontal solver that incorporates a global assembly tree $\mathbb{T}$ and local *HSS trees* denoted as $\mathcal{T}_k$, which are associated with each node k. There is another level called *parallel level* that is determined by the number of processes. We note that it is introduced only in the parallel implementation, which is not the main topic of this paper.

Throughout this paper, for the sake of brevity and clarity, we use k to represent any node on the assembly tree $\mathbb{T}$, with its two children d_1 and d_2. On each local HSS tree $\mathcal{T}_k$, we use i to represent any node on it. Two children associated with i are denoted as c_1 and c_2. $\mathcal{F}$, $\mathcal{U}$ and $\mathcal{T}$ without the subscript k are to indicate a general frontal matrix, update matrix, and HSS tree, respectively.

3. An overview of HSS constructions and factorizations of dense frontal matrices via randomized sampling. In this section, following the work of [10], [20] and [17], we develop a different HSS construction technique by exploiting the randomized sampling idea. We focus on the full HSS construction of each frontal matrix $\mathcal{F}_k$ without forming its dense form, only taking advantage of its sampling matrices. Imbedding this technique into the global multifrontal solver comprises the main result of section 4 and this paper. Other recent work on randomized sampling can be found by [6], [12], [11] and [7].

3.1. The structure-preserving rank revealing factorization. We revisit the definition of the HSS representation of a general frontal matrix $\mathcal{F}$, whose associated update matrix and HSS tree are $\mathcal{U}$ and $\mathcal{T}$ respectively, by exploiting the low rank properties of its off-diagonal blocks. The size of $\mathcal{F}$ is assumed to be $n \times n$. We say $\mathcal{F}$ is in its HSS form if and only if:

1. $\mathcal{T}$ is a post-ordering tree;
2. There exists an index set $\mathbf{I}_i$ associated with each node i on $\mathcal{T}$, such that $\mathbf{I}_{c_1} \bigcap \mathbf{I}_{c_2} = \emptyset$, $\mathbf{I}_{c_1} \bigcup \mathbf{I}_{c_2} = \mathbf{I}_i$, and $\mathcal{I} = \mathbf{I}_{root} = \{1, 2, ..., n\}$, where c_1 and c_2 are two children of the non-leaf node i;
3. There exist a family of matrices D_i, U_i, R_i, B_i, W_i and V_i (called *HSS generators*) associated with each node i of $\mathcal{T}$, such that
(3.1)
$$D_i \equiv \mathcal{F}|_{\mathbf{I}_i \times \mathbf{I}_i} = \begin{pmatrix} D_{c_1} & U_{c_1} B_{c_1} V_{c_2}^T \\ U_{c_2} B_{c_2} V_{c_1}^T & D_{c_2} \end{pmatrix}, \quad U_i = \begin{pmatrix} U_{c_1} R_{c_1} \\ U_{c_2} R_{c_2} \end{pmatrix}, \quad V_i = \begin{pmatrix} V_{c_1} W_{c_1} \\ V_{c_2} W_{c_2} \end{pmatrix}.$$

The maximum row and column size of all B_i is defined as the *HSS rank* of $\mathcal{F}$. Off-diagonal blocks of D_i are called *HSS blocks* of $\mathcal{F}$, which are defined as:

$$\mathcal{F}_i^- = \mathcal{F}|_{\mathbf{I}_i \times (\mathcal{I} - \mathbf{I}_i)}, \qquad \mathcal{F}_i^| = \mathcal{F}|_{(\mathcal{I} - \mathbf{I}_i) \times \mathbf{I}_i}; \tag{3.2}$$

Here $\mathcal{F}_i^-$ and $\mathcal{F}_i^|$ are called HSS row block and HSS column block, respectively.

In order to obtain the HSS representation (3.1) of $\mathcal{F}$, we need to compress $\mathcal{F}_i^-$ and $\mathcal{F}_i^|$ exploiting their low-rank properties. [15] utilized the conventional rank revealing QR factorization with column pivoting, by forming the dense matrices prior to compressing them. Here we are seeking a certain compression without forming the dense matrices.

We follow the work of [5] and [20] to conduct the *structure-preserving rank revealing* (SPRR) factorization of a general low-rank matrix $\mathcal{G}$, seeking its most linearly independent rows and columns, whose indices are $\widehat{\mathbf{I}}$ and $\widehat{\mathbf{J}}$ respectively.

$$\mathcal{G} \approx P \begin{pmatrix} I \\ E \end{pmatrix} \mathcal{G}|_{\widehat{\mathbf{I}} \times \widehat{\mathbf{J}}} \left(I, F^T\right) Q^T, \qquad \mathcal{G} = P \begin{pmatrix} I \\ E \end{pmatrix} \mathcal{G}|_{\widehat{\mathbf{I}}}, \qquad \mathcal{G}^T = Q \begin{pmatrix} I \\ F \end{pmatrix} \mathcal{G}^T|_{\widehat{\mathbf{J}}}, \tag{3.3}$$

where $P = \mathrm{diag}\,(P_1, P_2)$ and $Q = \mathrm{diag}\,(Q_1, Q_2)$ are permutations. E and F are expansions of the rest of rows and columns on the basis $\mathcal{G}|_{\widehat{\mathbf{I}}}$ and $\mathcal{G}^T|_{\widehat{\mathbf{J}}}$, respectively.

For the sake of brevity, We impose that indices in $\widehat{\mathbf{I}}$ and $\widehat{\mathbf{J}}$ are monotonically increasing. If they are not initially monotonically increasing, we can apply two permutations $\mathcal{P}$ and $\mathcal{Q}$ in the following way:

$$\mathcal{G}|_{\widetilde{\mathbf{I}} \times \widetilde{\mathbf{J}}} = \mathcal{P}\ \mathcal{G}|_{\widehat{\mathbf{I}} \times \widehat{\mathbf{J}}}\ \mathcal{Q}^T; \tag{3.4}$$

such that indices in $\widetilde{\mathbf{I}}$ and $\widetilde{\mathbf{J}}$ are monotonically increasing. It yields that:

$$\begin{aligned} \mathcal{G} &\approx \begin{pmatrix} P_1 \\ P_2 E \end{pmatrix} \mathcal{P}^T \mathcal{P}\ \mathcal{G}|_{\widehat{\mathbf{I}} \times \widehat{\mathbf{J}}}\ \mathcal{Q}^T \mathcal{Q} \left(Q_1^T, (Q_2 F)^T\right) \\ &= \begin{pmatrix} P_1 \mathcal{P}^T \\ P_2 (E \mathcal{P}^T) \end{pmatrix} \mathcal{G}|_{\widetilde{\mathbf{I}} \times \widetilde{\mathbf{J}}} \left((Q_1 \mathcal{Q}^T)^T, (Q_2 (F \mathcal{Q}^T))^T\right) \\ &= \begin{pmatrix} \widetilde{P}_1 \\ P_2 \widetilde{E} \end{pmatrix} \mathcal{G}|_{\widetilde{\mathbf{I}} \times \widetilde{\mathbf{J}}} \left(\widetilde{Q}_1^T, (Q_2 \widetilde{F})^T\right), \end{aligned} \tag{3.5}$$

which implies that the only operation requires to be conducted is $\widetilde{E} = E\mathcal{P}^T$ and $\widetilde{F} = F\mathcal{Q}^T$, after sorting P_1 and Q_1 into $\widetilde{P}_1$ and $\widetilde{Q}_1$, respectively. We adopt the bubble sort approach to realize this step.

Rather than from the original matrix $\mathcal{G}$ itself, [10] and [9] exploited the randomized sampling approach to obtain $\widehat{\mathbf{I}}$, $\widehat{\mathbf{J}}$, P, Q, E, F from two sampling matrices Y and Z which are defined below:

$$\begin{aligned} Y \equiv \mathcal{G}X = P \begin{pmatrix} I \\ E \end{pmatrix} \mathcal{G}|_{\widehat{\mathbf{I}}} X = P \begin{pmatrix} I \\ E \end{pmatrix} Y|_{\widehat{\mathbf{I}}} \\ Z \equiv \mathcal{G}^T X = Q \begin{pmatrix} I \\ F \end{pmatrix} \mathcal{G}^T|_{\widehat{\mathbf{J}}} X = Q \begin{pmatrix} I \\ F \end{pmatrix} Z|_{\widehat{\mathbf{J}}}, \end{aligned} \tag{3.6}$$

in which X is a random matrix. And we summarize the SPRR factorization into the concise form below:

$$\mathcal{G} \approx UBV^T, \qquad U = P\begin{pmatrix} I \\ E \end{pmatrix}, \qquad B = \mathcal{G}|_{\widehat{\mathbf{I}}\times\widehat{\mathbf{J}}}, \qquad V = Q\begin{pmatrix} I \\ F \end{pmatrix}. \tag{3.7}$$

3.2. Fast HSS constructions and factorizations of dense frontal matrices via randomized sampling. In this subsection, We review the HSS construction and factorization of a dense frontal matrix $\mathcal{F}$ via randomized sampling, which is proposed by [11] and [9]. Bear in mind that $\mathcal{U} = \mathcal{F}_{22} - \mathcal{F}_{21}\mathcal{F}_{11}^{-1}\mathcal{F}_{12}$ is the update matrix of $\mathcal{F}$. The HSS tree $\mathcal{T}$ associated with $\mathcal{F}$ is illustrated in Figure 2a, for an example of a block 8×8 frontal matrix.

We start from formulating two sampling matrices Y and Z by multiplying the random matrix X to both $\mathcal{F}$ and $\mathcal{F}^T$:

$$Y_{n\times r} = \mathcal{F}_{n\times n}X_{n\times r}, \qquad Z_{n\times r} = \mathcal{F}_{n\times n}^T X_{n\times r}; \tag{3.8}$$

where n is the size of the frontal matrix $\mathcal{F}$, r is the estimated maximum numerical rank of all HSS blocks of $\mathcal{F}$. $\{X, Y, Z\}$ comprises three $n \times r$ tall skinny matrices which are illustrated in Figure 2b. Subtracting the diagonal information out of Y and Z (illustrated in Figure 2c), we obtain the HSS block sampling information of each leaf node:

$$\begin{aligned} D_i &= \mathcal{F}|_{\mathbf{I}_i\times\mathbf{J}_i}, \\ \mathcal{F}_i^- X_i &= \Phi_i = Y_i - D_i X_i, \\ (\mathcal{F}_i^|)^T X_i &= \Theta_i = Z_i - D_i^T X_i, \qquad i = \text{leaf nodes}; \end{aligned} \tag{3.9}$$

in which Φ and Θ denote sampling matrices of HSS blocks of each node on $\mathcal{T}$. We conduct the SPRR compression:

$$\begin{aligned} \Phi_i &\approx P_i\begin{pmatrix} I \\ E_i \end{pmatrix}\Phi_i|_{\widehat{\mathbf{I}}_i} = U_i\ \Phi_i|_{\widehat{\mathbf{I}}_i}, \\ \Theta_i &\approx Q_i\begin{pmatrix} I \\ F_i \end{pmatrix}\Theta_i|_{\widehat{\mathbf{J}}_i} = V_i\ \Theta_i|_{\widehat{\mathbf{J}}_i}, \qquad i = \text{leaf nodes}. \end{aligned} \tag{3.10}$$

Then we move on to non-leaf nodes. The HSS generator B_i can be obtained via:

$$B_{c_1} = \mathcal{F}|_{\widehat{\mathbf{I}}_{c_1}\times\widehat{\mathbf{J}}_{c_2}}, \qquad B_{c_2} = \mathcal{F}|_{\widehat{\mathbf{I}}_{c_2}\times\widehat{\mathbf{J}}_{c_1}}, \qquad i \neq \text{leaf nodes}. \tag{3.11}$$

The sampling matrices of non-leaf node HSS blocks are:

$$\begin{aligned} \mathcal{F}_i^- X_i &= \begin{pmatrix} \Phi_{c_1} - F_{c_1,c_2}X_{c_2} \\ \Phi_{c_2} - F_{c_2,c_1}X_{c_1} \end{pmatrix} = \begin{pmatrix} U_{c_1}\Phi_{c_1}|_{\widehat{\mathbf{I}}_{c_1}} - U_{c_1}B_{c_1}V_{c_2}^T X_{c_2} \\ U_{c_2}\Phi_{c_2}|_{\widehat{\mathbf{I}}_{c_2}} - U_{c_2}B_{c_2}V_{c_1}^T X_{c_1} \end{pmatrix} \\ &= \begin{pmatrix} U_{c_1} & 0 \\ 0 & U_{c_2} \end{pmatrix}\begin{pmatrix} \Phi_{c_1}|_{\widehat{\mathbf{I}}_{c_1}} - B_{c_1}V_{c_2}^T X_{c_2} \\ \Phi_{c_2}|_{\widehat{\mathbf{I}}_{c_2}} - B_{c_2}V_{c_1}^T X_{c_1} \end{pmatrix} = \begin{pmatrix} U_{c_1} \\ U_{c_2} \end{pmatrix}\Phi_i; \\ (\mathcal{F}_i^|)^T X_i &= \begin{pmatrix} \Theta_{c_1} - F_{c_2,c_1}^T X_{c_2} \\ \Theta_{c_2} - F_{c_1,c_2}^T X_{c_1} \end{pmatrix} = \begin{pmatrix} V_{c_1}\Theta_{c_1}|_{\widehat{\mathbf{J}}_{c_1}} - V_{c_1}B_{c_2}^T U_{c_2}^T X_{c_2} \\ V_{c_2}\Theta_{c_2}|_{\widehat{\mathbf{J}}_{c_2}} - V_{c_2}B_{c_1}^T U_{c_1}^T X_{c_1} \end{pmatrix} \\ &= \begin{pmatrix} V_{c_1} & 0 \\ 0 & V_{c_2} \end{pmatrix}\begin{pmatrix} \Theta_{c_1}|_{\widehat{\mathbf{J}}_{c_1}} - B_{c_2}^T U_{c_2}^T X_{c_2} \\ \Theta_{c_2}|_{\widehat{\mathbf{J}}_{c_2}} - B_{c_1}^T U_{c_1}^T X_{c_1} \end{pmatrix} = \begin{pmatrix} V_{c_1} \\ V_{c_2} \end{pmatrix}\Theta_i; \qquad i \neq \text{leaf nodes}. \end{aligned} \tag{3.12}$$

Then we further compress the residual blocks Φ_i and Θ_i via SPRR:

$$\begin{aligned} \Phi_i &\equiv \begin{pmatrix} \Phi_{c_1}|_{\widehat{\mathbf{I}}_{c_1}} - B_{c_1}V_{c_2}^T X_{c_2} \\ \Phi_{c_2}|_{\widehat{\mathbf{I}}_{c_2}} - B_{c_2}V_{c_1}^T X_{c_1} \end{pmatrix} \approx P_i\begin{pmatrix} I \\ E_i \end{pmatrix}\Phi_i|_{\widehat{\mathbf{I}}_i} = \begin{pmatrix} R_{c_1} \\ R_{c_2} \end{pmatrix}\Phi_i|_{\widehat{\mathbf{I}}_i}, \\ \Theta_i &\equiv \begin{pmatrix} \Theta_{c_1}|_{\widehat{\mathbf{J}}_{c_1}} - B_{c_2}^T U_{c_2}^T X_{c_2} \\ \Theta_{c_2}|_{\widehat{\mathbf{J}}_{c_2}} - B_{c_1}^T U_{c_1}^T X_{c_1} \end{pmatrix} \approx Q_i\begin{pmatrix} I \\ F_i \end{pmatrix}\Theta_i|_{\widehat{\mathbf{J}}_i} = \begin{pmatrix} W_{c_1} \\ W_{c_2} \end{pmatrix}\Theta_i|_{\widehat{\mathbf{J}}_i},\ i \neq \text{leaf nodes}. \end{aligned} \tag{3.13}$$

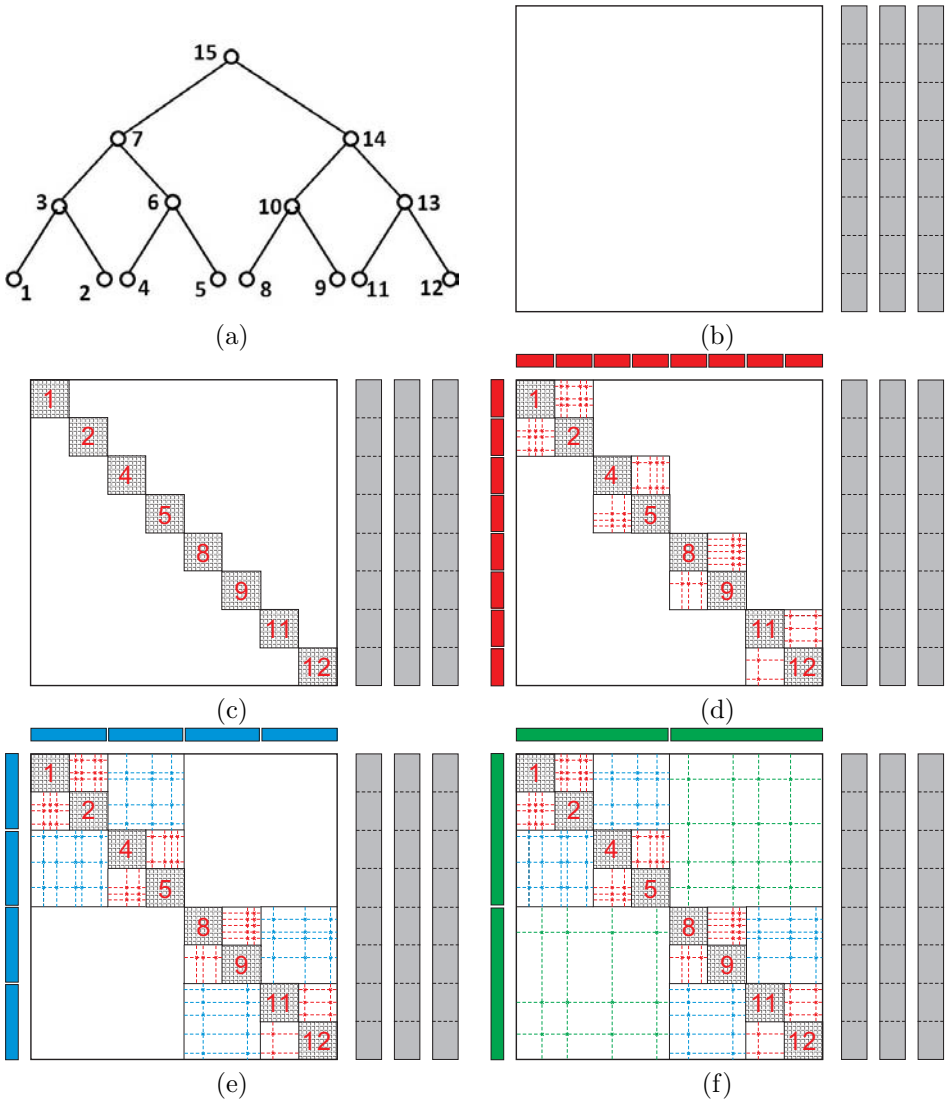

FIG. 2. *(a): the HSS tree $\mathcal{T}$ associated with a block 8×8 frontal matrix $\mathcal{F}$; (b): before HSS construction; (c): first level HSS construction; (d): second level HSS construction; (e): third level HSS construction; (f): fourth level HSS construction.*

Figure 2d - 2f illustrate the HSS construction for non-leaf nodes.

We draw a key observation that only HSS generators D_i and B_i come from the original $\mathcal{F}$, while the rest HSS generators U_i, V_i, R_i, W_i come from two sampling matrices Y and Z. Also we note that for all i it holds that $\widehat{\mathbf{I}}_i \subset \mathbf{I}_i$ and $\widehat{\mathbf{J}}_i \subset \mathbf{J}_i$. Moreover, $\widehat{\mathbf{I}}_i \subset (\widehat{\mathbf{I}}_{c_1} \cup \widehat{\mathbf{I}}_{c_2})$ and $\widehat{\mathbf{J}}_i \subset (\widehat{\mathbf{J}}_{c_1} \cup \widehat{\mathbf{J}}_{c_2})$ hold for non-leaf node i.

Following [15], we conduct the fast HSS ULV factorization only for those nodes i belonging to $\mathcal{F}_{11}$, at the same time when the HSS construction is carried out. [20] developed an efficient way to introduce zeros to U_i:

$$\begin{pmatrix} -E_i & I \\ I & 0 \end{pmatrix} P_i^T U_i = \begin{pmatrix} -E_i & I \\ I & 0 \end{pmatrix} P_i^T P_i \begin{pmatrix} I \\ E_i \end{pmatrix} = \begin{pmatrix} 0 \\ I \end{pmatrix} \tag{3.14}$$

Now we summarize the entire HSS construction and factorization via randomized sampling into the algorithm 1:

ALGORITHM 1: The HSS construction and factorization of $\mathcal{F}$ via randomized sampling.

generating $Y = \mathcal{F}X$, $Z = \mathcal{F}^T X$;
do i = 1,length($\mathcal{T}$)
  if (i is a leaf-node), then
    1. extraction: $D_i = \mathcal{F}|_{\mathbf{I}_i \times \mathbf{J}_i}$;
    2. subtraction: $\Phi_i = Y_i - D_i X_i$, $\Theta_i = Z_i - D_i^T X_i$;
    3. compression: $\Phi_i \approx U_i \Phi_i|_{\widehat{\mathbf{I}}_i}$, $\Theta_i \approx V_i \Theta_i|_{\widehat{\mathbf{J}}_i}$;
  else
    1. extraction: $B_{c_1} = \mathcal{F}|_{\widehat{\mathbf{I}}_{c_1} \times \widehat{\mathbf{J}}_{c_2}}$, $B_{c_2} = \mathcal{F}|_{\widehat{\mathbf{I}}_{c_2} \times \widehat{\mathbf{J}}_{c_1}}$;
    2. subtraction: $\Phi_i = \begin{pmatrix} \Phi_{c_1}|_{\widehat{\mathbf{I}}_{c_1}} - B_{c_1} V_{c_2}^T X_{c_2} \\ \Phi_{c_2}|_{\widehat{\mathbf{I}}_{c_2}} - B_{c_2} V_{c_1}^T X_{c_1} \end{pmatrix}$, $\Theta_i = \begin{pmatrix} \Theta_{c_1}|_{\widehat{\mathbf{J}}_{c_1}} - B_{c_2}^T U_{c_2}^T X_{c_2} \\ \Theta_{c_2}|_{\widehat{\mathbf{J}}_{c_2}} - B_{c_1}^T U_{c_1}^T X_{c_1} \end{pmatrix}$;
    3. deallocation: Φ_{c_1}, Φ_{c_2}, Θ_{c_1} and Θ_{c_2};
    4. compression: $\Phi_i \approx \begin{pmatrix} R_{c_1} \\ R_{c_2} \end{pmatrix} \Phi_i|_{\widetilde{\mathbf{I}}_i}$, $\Theta_i \approx \begin{pmatrix} W_{c_1} \\ W_{c_2} \end{pmatrix} \Theta_i|_{\widetilde{\mathbf{J}}_i}$;
  end if
  if ($i \in \mathcal{F}_{22}$) then
    continue;
  else
    HSS ULV factorization;
  end if
end do

3.3. An efficient structured Schur complement update. After the HSS construction for $\mathcal{F}$ and HSS ULV factorization for $\mathcal{F}_{11}$ are done, we need to compute the update matrix $\mathcal{U}$ by low rank updating the Schur complement block $\mathcal{F}_{22}$ based on the formula proposed by [19] below. Here we use a block 8×8 frontal matrix as an example:

$$\begin{aligned} \widetilde{D}_{14} &= D_{14} - U_{14} B_{14} \left(V_7^T D_7^{-1} U_7 \right) B_7 V_{14}^T \\ &= D_{14} - U_{14} B_{14} \left(\widetilde{V}_7^T \widetilde{D}_7^{-1} \widetilde{U}_7 \right) B_7 V_{14}^T \end{aligned} \tag{3.15}$$

where $\widetilde{D}_7$, $\widetilde{U}_7$ and $\widetilde{V}_7$ are residue blocks of D_7, U_7 and V_7 respectively, after the HSS factorization of $\mathcal{F}_{11}$ block. Being different from the classical structured solver by [15] that D_{14} block is dense, now D_{14} block is fully represented by the HSS structure.

Here we draw an important conclusion that the low-rank update only updates the D_i and B_i blocks, without changing any U_i, V_i, R_i and W_i blocks for $i \in \mathcal{F}_{22}$. For instance:

$$\widetilde{D}_8 = D_8 - U_8 R_8 R_{10} B_{14} \left(\widetilde{V}_7^T \widetilde{D}_7^{-1} \widetilde{U}_7 \right) B_7 W_{10}^T W_8^T V_8^T \tag{3.16}$$

$$\begin{aligned} \widetilde{F}_{10,13} &= U_{10} B_{10} V_{13}^T - U_{10} R_{10} B_{14} \left(\widetilde{V}_7^T \widetilde{D}_7^{-1} \widetilde{U}_7 \right) B_7 W_{13}^T V_{13}^T \\ &= U_{10} \left(B_{10} - R_{10} B_{14} \left(\widetilde{V}_7^T \widetilde{D}_7^{-1} \widetilde{U}_7 \right) B_7 W_{13}^T \right) V_{13}^T \\ &= U_{10} \widetilde{B}_{10} V_{13}^T; \end{aligned} \tag{3.17}$$

$$\begin{aligned}\widetilde{F}_{8,9} &= U_8 B_8 V_9^T - U_8 R_8 R_{10} B_{14} \left(\widetilde{V}_7^T \widetilde{D}_7^{-1} \widetilde{U}_7\right) B_7 W_{10}^T W_9^T V_9^T \\ &= U_8 \left(B_8 - R_8 R_{10} B_{14} \left(\widetilde{V}_7^T \widetilde{D}_7^{-1} \widetilde{U}_7\right) B_7 W_{10}^T W_9^T\right) V_9^T \\ &= U_8 \widetilde{B}_8 V_9^T;\end{aligned} \tag{3.18}$$

We note that when we compute $\widetilde{F}_{8,9}$, $R_{10}B_{14}\left(\widetilde{V}_7^T \widetilde{D}_7^{-1} \widetilde{U}_7\right) B_7$ is already included in the computation of $\widetilde{F}_{10,13}$. Similarly, when we compute $\widetilde{D}_8$, $R_8 R_{10} B_{14}\left(\widetilde{V}_7^T \widetilde{D}_7^{-1} \widetilde{U}_7\right) B_7 W_{10}^T$ is already included in the computation of $\widetilde{F}_{8,9}$. Thus we adopt a top-down approach to update D_i and B_i belonging to $\mathcal{F}_{22}$, by traversing the HSS tree $\mathcal{T}$ backward.

3.4. Implementation details. We now discuss implementation details that enable the HSS construction and factorization of a dense frontal matrix $\mathcal{F}$ via randomized sampling to be conducted in an efficient way, following MATLAB conventions of `cell arrays`, the counterpart of which in C or Fortran is `array of pointers`.

From the definition (equation (3.1)) of HSS structures, we note that HSS generators are associated with each node i on the HSS tree $\mathcal{T}$. Thus before the algorithm 1 starts, we first allocate cell arrays for each generator: $D = \texttt{cell}(1, \texttt{length}(\mathcal{T}))$, $B = \texttt{cell}(1, \texttt{length}(\mathcal{T}))$, $U = \texttt{cell}(1, \texttt{length}(\mathcal{T}))$, $V = \texttt{cell}(1, \texttt{length}(\mathcal{T}))$, $R = \texttt{cell}(1, \texttt{length}(\mathcal{T}))$, $W = \texttt{cell}(1, \texttt{length}(\mathcal{T}))$, in which each cell element $D\{i\}$, $B\{i\}$, $U\{i\}$, $V\{i\}$, $R\{i\}$ and $W\{i\}$ waits to be allocated to a piece of memory storing each HSS generator. Furthermore, since the size of each B_i records numerical ranks of each HSS block, we allocate two integer arrays $rkR = \texttt{zeros}(1, \texttt{length}(\mathcal{T}))$ and $rkC = \texttt{zeros}(1, \texttt{length}(\mathcal{T}))$ where rkR and rkC stand for row and column numerical ranks, respectively.

In addition, algorithm 1 indicates that during the process of HSS construction, intermediate storage is required to store sampling matrices Φ and Θ, as well as index sets $\widehat{\mathbf{I}}_i$ and $\widehat{\mathbf{J}}_i$ of each HSS block. We also note that at each loop i, $U_i^T X_i$ and $V_i^T X_i$ need to be evaluated. For a leaf-node i, this becomes direct matrix-matrix multiplications. However, for a non-leaf node i, it is inefficient to form U_i and V_i followed by the multiplication with X_i. On the contrary, we observe that by taking advantage of the intrinsic recursion properties of the HSS structure (equation (3.1)), the evaluation of $U_i^T X_i$ and $V_i^T X_i$ can be conducted in an efficient recursive way below, via defining two intermediate variables:

$$\begin{aligned}Ux_i \equiv U_i^T X_i &= \left(R_{c_1}^T U_{c_1}^T, R_{c_2}^T U_{c_2}^T\right) \begin{pmatrix} X_{c_1} \\ X_{c_2} \end{pmatrix} = R_{c_1}^T Ux_{c_1} + R_{c_2}^T Ux_{c_2}; \\ Vx_i \equiv V_i^T X_i &= \left(W_{c_1}^T V_{c_1}^T, W_{c_2}^T V_{c_2}^T\right) \begin{pmatrix} X_{c_1} \\ X_{c_2} \end{pmatrix} = W_{c_1}^T Vx_{c_1} + W_{c_2}^T Vx_{c_2}.\end{aligned} \tag{3.19}$$

Therefore, we allocate cell arrays for these intermediate variables: $\Phi = \texttt{cell}(1, \texttt{length}(\mathcal{T}))$, $\Theta = \texttt{cell}(1, \texttt{length}(\mathcal{T}))$, $\widehat{\mathbf{I}} = \texttt{cell}(1, \texttt{length}(\mathcal{T}))$, $\widehat{\mathbf{J}} = \texttt{cell}(1, \texttt{length}(\mathcal{T}))$, $Ux = \texttt{cell}(1, \texttt{length}(\mathcal{T}))$, $Vx = \texttt{cell}(1, \texttt{length}(\mathcal{T}))$. Being different from HSS generators, two children's intermediate variables are deallocated once the parent one is formulated.

To summarize, from the implementation aspect of the entire HSS algorithm via randomized sampling, we altogether have six cell arrays $\{D, B, U, V, R, W\}$ associated with generators, two integer arrays $\{rkR, rkC\}$ recording numerical ranks, and six cell arrays $\{\Phi, \Theta, \widehat{\mathbf{I}}, \widehat{\mathbf{J}}, Ux, Vx\}$ associated with intermediate variables, before the loop in algorithm 1 starts.

We combine the HSS construction and the HSS factorization together in a single loop displayed in algorithm 1. At each node i, the first step is always extracting the information out of the original matrix $\mathcal{F}$. If i is a leaf node, $D\{i\}$ is allocated, followed by the initiation of $\Phi\{i\}$ and $\Theta\{i\}$. We point out that three tall skinny matrices $\{X, Y, Z\}$ enter the loop only through leaf nodes. If i is a non-leaf node, $B\{c_1\}$ and $B\{c_2\}$ are allocated, followed by the generation of $\Phi\{i\}$ and $\Theta\{i\}$ via merging two children's counterparts. $\Phi\{c_1\}$, $\Phi\{c_2\}$, $\Theta\{c_1\}$ and $\Theta\{c_2\}$ are then deallocated.

After the SPRR compression is done, we need to collect the rank information into the $rkR(i)$ and $rkC(i)$, to store index sets $\widehat{\mathbf{I}}\{i\}$ and $\widehat{\mathbf{J}}\{i\}$, as well as to allocate $Ux\{i\}$ and $Vx\{i\}$. The HSS ULV factorization will not be carried out unless i belongs to the the Schur complement part. When the root of $\mathcal{T}$ is reached, all intermediate variables are deallocated, leaving HSS generators and two rank integer arrays containing the complete information of $\mathcal{F}$. Eventually, the Schur complement is efficiently updated via a top-down approach, which finalizes the HSS construction and factorization of $\mathcal{F}$.

4. Structured multifrontal solver via randomized sampling. In this section, we discuss the imbedding of the randomized sampling approach of the HSS construction and factorization studied in section 3 into the global multifrontal solver, which comprises the main result of this paper.

4.1. Extend-add of sampling matrices of two update matrices. We revisit one observation made in section 3 that only D and B generators come from the original frontal matrix $\mathcal{F}$, while the rest U, V, R, W generators are from the sampling information Y and Z. On the other hand, equation (2.3) implies that the information contained in each frontal matrix $\mathcal{F}_k$ comes from part of the original sparse matrix $\mathcal{A}_k$, and two children's update matrices $\mathcal{U}_{d_1}$ and $\mathcal{U}_{d_2}$ if k is a non-leaf node. Therefore, to construct the full HSS representation of each frontal matrix $\mathcal{F}_k$ above the switch level $slvl$, the complete information we need is:

1. part of the original sparse matrix $\mathcal{A}_k$ associated with the node k;
2. two update matrices in the HSS form $\mathcal{U}_{d_1}$ and $\mathcal{U}_{d_2}$ if k is a non-leaf node;
3. three tall skinny matrices $\{X_k, Y_k = \mathcal{F}_k X_k, Z_k = \mathcal{F}_k^T X_k\}$.

It is noted that a big difference between the new structured solver with randomized sampling and the conventional structured solver by [14] lies in that there is no dense frontal matrix $\mathcal{F}_k$ formulated at each node k above $slvl$. In other words, $\{X_k, Y_k, Z_k\}$ together with $\{\mathcal{A}_k, \mathcal{U}_{d_1}, \mathcal{U}_{d_2}\}$ is sufficient for the HSS construction of $\mathcal{F}_k$. Therefore, the key is how to obtain Y_k and Z_k efficiently without the explicit formulation of $\mathcal{F}_k$ and hence the explicit multiplication with X_k, as against the conventional multifrontal method.

We rewrite equation (2.3) into the following form, for a general non-leaf node k:

$$\begin{aligned}\mathcal{F}_k = \mathcal{A}_k + \Big[&\Big(\mathcal{F}_{d_1,22} - U_{d_1,2}B_{d_1,2}\left(\widetilde{V}_{d_1,1}^T\widetilde{D}_{d_1,1}^{-1}\widetilde{U}_{d_1,1}\right)B_{d_1,1}V_{d_1,2}^T\Big)\\ &\oplus\Big(\mathcal{F}_{d_2,22} - U_{d_2,2}B_{d_2,2}\left(\widetilde{V}_{d_2,1}^T\widetilde{D}_{d_2,1}^{-1}\widetilde{U}_{d_2,1}\right)B_{d_2,1}V_{d_2,2}^T\Big)\Big]\end{aligned} \tag{4.1}$$

Multiplying X_k on both sides, we obtain:

$$\begin{aligned}Y_k \equiv \mathcal{F}_k X_k &= \mathcal{A}_k X_k + \Big[\Big(\mathcal{F}_{d_1,22}X_{d_1,2} - U_{d_1,2}B_{d_1,2}\left(\widetilde{V}_{d_1,1}^T\widetilde{D}_{d_1,1}^{-1}\widetilde{U}_{d_1,1}\right)B_{d_1,1}V_{d_1,2}^T X_{d_1,2}\Big)\\ &\qquad\oplus\Big(\mathcal{F}_{d_2,22}X_{d_2,2} - U_{d_2,2}B_{d_2,2}\left(\widetilde{V}_{d_2,1}^T\widetilde{D}_{d_2,1}^{-1}\widetilde{U}_{d_2,1}\right)B_{d_2,1}V_{d_2,2}^T X_{d_2,2}\Big)\Big]\\ &= \mathcal{A}_k X_k + \Big\{\Big[Y_{d_1,2} - U_{d_1,2}B_{d_1,2}\left(V_{d_1,1}^T X_{d_1,1} + \left(\widetilde{V}_{d_1,1}^T\widetilde{D}_{d_1,1}^{-1}\widetilde{U}_{d_1,1}\right)B_{d_1,1}V_{d_1,2}^T X_{d_1,2}\right)\Big]\\ &\qquad\oplus\Big[Y_{d_2,2} - U_{d_2,2}B_{d_2,2}\left(V_{d_2,1}^T X_{d_2,1} + \left(\widetilde{V}_{d_2,1}^T\widetilde{D}_{d_2,1}^{-1}\widetilde{U}_{d_2,1}\right)B_{d_2,1}V_{d_2,2}^T X_{d_2,2}\right)\Big]\Big\}\\ &= \mathcal{A}_k X_k + \left(\widetilde{Y}_{d_1,2}\oplus\widetilde{Y}_{d_2,2}\right)\end{aligned}$$

We can obtain similar results for Z_k. To summarize, we have the following new extend-add formula for sampling matrices Y_k and Z_k, which are counterparts of equation (2.3):

$$Y_k = \begin{pmatrix} Y_{k,1} \\ Y_{k,2} \end{pmatrix} \equiv \mathcal{F}_k X_k = \begin{pmatrix} \mathcal{A}_{k,11} & \mathcal{A}_{k,12} \\ \mathcal{A}_{k,21} & 0 \end{pmatrix} \begin{pmatrix} X_{k,1} \\ X_{k,2} \end{pmatrix} + \left(\widetilde{Y}_{d_1,2} \oplus \widetilde{Y}_{d_2,2} \right) \tag{4.2}$$

$$Z_k = \begin{pmatrix} Z_{k,1} \\ Z_{k,2} \end{pmatrix} \equiv \mathcal{F}_k^T X_k = \begin{pmatrix} \mathcal{A}_{k,11}^T & \mathcal{A}_{k,21}^T \\ \mathcal{A}_{k,12}^T & 0 \end{pmatrix} \begin{pmatrix} X_{k,1} \\ X_{k,2} \end{pmatrix} + \left(\widetilde{Z}_{d_1,2} \oplus \widetilde{Z}_{d_2,2} \right) \tag{4.3}$$

$$\widetilde{Y}_{k,2} = Y_{k,2} - U_{k,2} B_{k,2} \left(V_{k,1}^T X_{k,1} + \left(\widetilde{V}_{k,1}^T \widetilde{D}_{k,1}^{-1} \widetilde{U}_{k,1} \right) B_{k,1} V_{k,2}^T X_{k,2} \right) \tag{4.4}$$

$$\widetilde{Z}_{k,2} = Z_{k,2} - V_{k,2} B_{k,1}^T \left(U_{k,1}^T X_{k,1} + \left(\widetilde{U}_{k,1}^T \widetilde{D}_{k,1}^{-T} \widetilde{V}_{k,1} \right) B_{k,2}^T U_{k,2}^T X_{k,2} \right) \tag{4.5}$$

$$X_k = X_{d_1,2} \bigcup X_{d_2,2} \tag{4.6}$$

By taking advantage of equation (3.19), we note that $U_{k,1}^T X_{k,1}$ and $V_{k,1}^T X_{k,1}$ are efficiently computed in a recursive manner. We also point out that $\mathcal{A}_k X_k$ can be computed with linear complexity due to the sparsity of $\mathcal{A}_k$. Figure 3 illustrates the new extend-add algorithm for three tall skinny matrices.

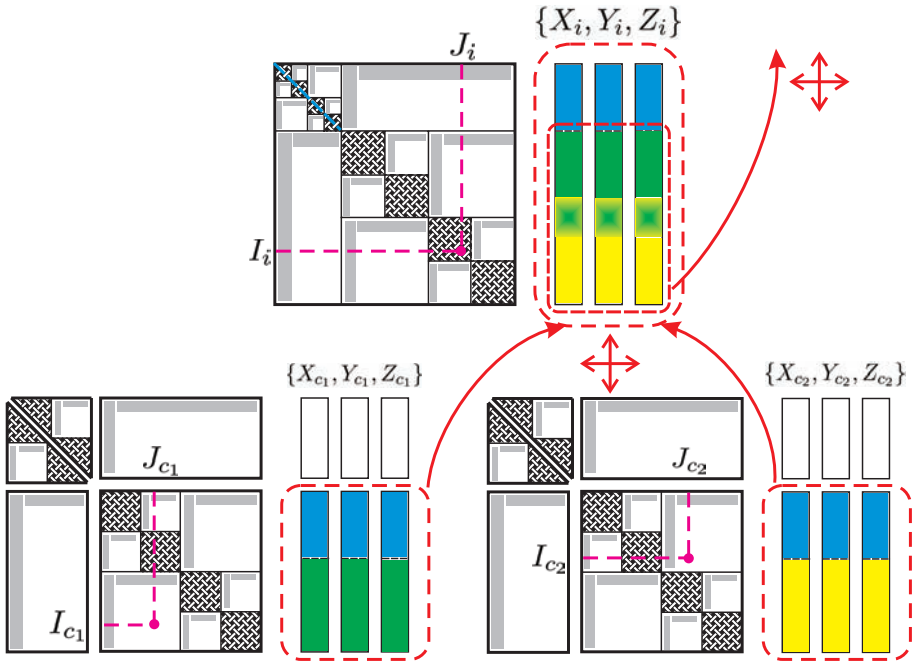

FIG. 3. *The illustration of imbedding randomized sampling HSS construction and factorization into the global multifrontal solver, via the extend-add of three tall skinny matrices.*

4.2. Implementation details of the extend-add process. From the implementation perspective, the key is to compute the extend-add for three tall skinny matrices $\{X_k, Y_k, Z_k\}$.

Equation (4.6) implies that X_k is the union rather than the extend-add of $X_{d_1,2}$ and $X_{d_2,2}$, which is different from Y_k and Z_k obtained via equation (4.2) and equation (4.3), respectively. This is equivalent to say that any $X_{d,2}$ is a subset of its parent X_k. Furthermore, we note that there is no computation on X_k during the HSS construction and factorization, which means X_k should be pre-allocated prior to the entire structured multifrontal solver is started.

We start from revisiting the conventional extend-add for dense matrices. We introduce two integer arrays $\texttt{ind}\{k\}$ and $\texttt{map}\{k\}$ associated with each node k. For the sake of clarity and convenience, we use `ind` and `map` without subscripts to represent general $\texttt{ind}\{k\}$ and $\texttt{map}\{k\}$. The parent frontal matrix and the child update matrix are denoted as $\mathcal{F}_k$ and $\mathcal{U}_d$, respectively. and

$\texttt{length}(\texttt{ind}) = N_d + 1$, $\texttt{length}(\texttt{map}) = N_d$, where N_d denotes the number of diagonal blocks, or in other words number of neighbors, in the update matrix $\mathcal{U}_d$ waiting for the extend-add.

Each element $\texttt{ind}(j)$ stores the first row or column index of each diagonal block j, such that the size of each diagonal block j is $\texttt{ind}(j+1) - \texttt{ind}(j)$, $1 \le j \le N_d$. $\texttt{map}(j)$ maps $\texttt{ind}(j)$ to the corresponding row or column index in the frontal matrix $\mathcal{F}_k$. Thus we can summarize the conventional extend-add of dense matrices into the algorithm 2.

ALGORITHM 2: Extend-add for dense matrices

```
do jj = 1,N_d
    do ii = 1,N_d
        hi = map(ii),  ti = map(ii)+ind(ii+1)-ind(ii)-1;
        hj = map(jj),  tj = map(jj)+ind(jj+1)-ind(jj)-1;
        F_k(hi:ti,hj:tj) += U_d(ind(ii):ind(ii+1)-1, ind(jj):ind(jj+1)-1);
    end do
end do
```

Rather than dense matrices, now we have tall skinny matrices to extend-add whose column size is fixed to be the estimated rank $mxrk$, which makes it easier to implement since we only have the row-wise extend-add. Algorithm 3 describes the strategy.

ALGORITHM 3: Extend-add for Y and Z

```
do jj = 1,N_d
    hj = map(jj),  tj = map(jj)+ind(jj+1)-ind(jj)-1;
    Y_k(hj:tj,:) += Y_{d,2}(ind(jj):ind(jj+1)-1,:);
    Z_k(hj:tj,:) += Z_{d,2}(ind(jj):ind(jj+1)-1,:);
end do
```

As for the pre-allocation of X, we adopt a top-down approach to realize it, which is illustrated by Figure 4. Additionally we only store the pre-allocated random matrices X_k on the switch level $slvl$. The details are studied in algorithm 4. The function we use to generate random matrices is `randn` in MATLAB, or `RANDOM_SEED` and `RANDOM_NUMBER` in Fortran.

4.3. Efficient algorithms for obtaining D and B generators via matrix blocking techniques. In algorithm 1, we note that at each node of the HSS tree, either D or B generator requires to be evaluated, given their index sets of the frontal matrix $\mathcal{F}$. If $\mathcal{F}$ is in its dense form, this is just by straightforward extraction. However, in the global structured multifrontal solver via randomized sampling, there is no dense frontal matrix $\mathcal{F}_k$ ever formulated above the switch level $slvl$, which makes such an extraction operation non-trivial and deserve to be deeply investigated.

To generalize our discussion, we use $\mathcal{D}|_{\mathbf{I}\times\mathbf{J}} \equiv \mathcal{F}_k|_{\mathbf{I}_\mathcal{F}\times\mathbf{J}_\mathcal{F}}$ to represent any D and B generators, in which $\mathbf{I} = \{1, 2, ..., N_I\}$, $\mathbf{J} = \{1, 2, ..., N_J\}$. $\mathbf{I}_\mathcal{F}$ and $\mathbf{J}_\mathcal{F}$ are row and column index sets of the frontal matrix $\mathcal{F}_k$, respectively. Remind that indices in $\mathbf{I}_\mathcal{F}$ and $\mathbf{J}_\mathcal{F}$ are monotonically increasing. Furthermore, N_I and N_J are numbers of indices in $\mathbf{I}_\mathcal{F}$ and $\mathbf{J}_\mathcal{F}$, respectively. There are two parts of contributions of $\mathcal{F}_k|_{\mathbf{I}_\mathcal{F}\times\mathbf{J}_\mathcal{F}}$: one is from $\mathcal{A}_k|_{\mathbf{I}_\mathcal{F}\times\mathbf{J}_\mathcal{F}}$, which is by straightforward summation; the other one is from the extend-add of two children's update matrices, on which our investigations are focused.

Thus we formally propose our problem to solve in the following statement: given $\mathbf{I}_\mathcal{F}$ and $\mathbf{J}_\mathcal{F}$, how to efficiently find the contribution $\mathcal{U}_d|_{\mathbf{I}_\mathcal{U}\times\mathbf{J}_\mathcal{U}}$ in which $d = d_1$ or d_2, such that

$$\begin{aligned} &\mathcal{D}|_{\mathbf{I}\times\mathbf{J}} \equiv \mathcal{F}_k|_{\mathbf{I}_\mathcal{F}\times\mathbf{J}_\mathcal{F}} \quad \longmapsto \quad \mathcal{U}_d|_{\mathbf{I}_\mathcal{U}\times\mathbf{J}_\mathcal{U}}, \\ &\mathbf{I}_\mathcal{U} = \texttt{indxF2U}\left(\mathbf{I}_\mathcal{F}\right), \qquad \mathbf{J}_\mathcal{U} = \texttt{indxF2U}\left(\mathbf{J}_\mathcal{F}\right). \end{aligned} \tag{4.7}$$

ALGORITHM 4: Top-down generation of X

```
do k = length(𝕋),1,-1
    if (level(k) < slvl), then
        continue;
    else
        1.  X_{k,1} = randn( size(F_{k,11}), mxrk );
        2.  X_k = [X_{k,1}; X_{k,2}];
        3.  if (level(k) == slvl), then
                continue;
            else
                do d = d_1 and d_2
                    do jj = 1,N_d
                        hj = map(jj),  tj = map(jj)+ind(jj+1)-ind(jj)-1;
                        X_{d,2}(ind(jj):ind(jj+1)-1,:)  = X_k(hj:tj,:);
                    end do
                end do
                4.  deallocate(X_k);
            end if
    end if
end do
```

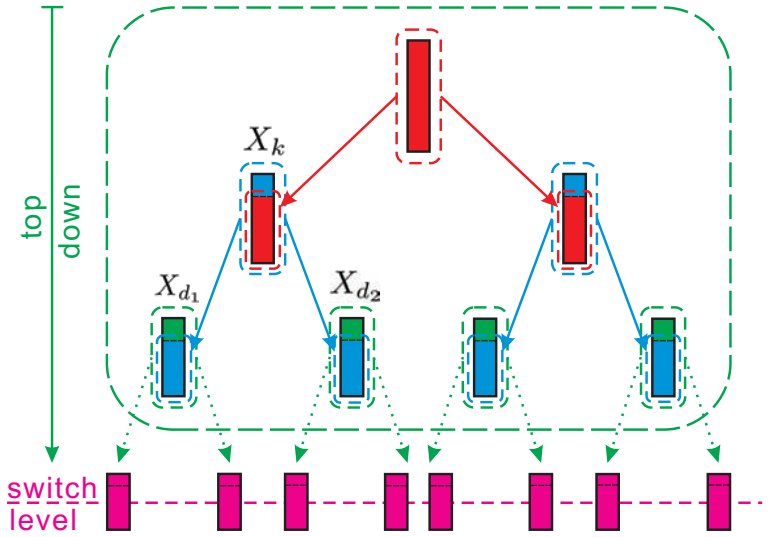

FIG. 4. *The illustration of the top-down generation process of random matrices X_k. Only X_k on the switch level slvl are stored before the structured multifrontal factorization is carried out.*

Here we define a subroutine called `indxF2U`, which utilizes the extend-add process to map the index sets $[\mathbf{I}_{\mathcal{F}}, \mathbf{J}_{\mathcal{F}}]$ of the frontal matrix $\mathcal{F}_k$ to the index sets $[\mathbf{I}_{\mathcal{U}}, \mathbf{J}_{\mathcal{U}}]$ of the update matrix $\mathcal{U}_d$. Because indices in $\mathbf{I}_{\mathcal{F}}$ and $\mathbf{J}_{\mathcal{F}}$ are monotonically increasing, correspondingly the mapped indices in $\mathbf{I}_{\mathcal{U}}$ and $\mathbf{J}_{\mathcal{U}}$ are also monotonically increasing. If the mapping of some indices of $\mathcal{F}_k$ are not in the range of $\mathcal{U}_d$, we evaluate the mapping to be -1. Figure 5 illustrates the function `indxF2U`. Implementation details can be summarized in the algorithm 5 below. $\mathbf{J}_{\mathcal{U}}$ is obtained in exactly the same way.

After obtaining $\mathbf{I}_{\mathcal{U}}$ and $\mathbf{J}_{\mathcal{U}}$, we know that the contribution of $\mathcal{U}_d$ to $\mathcal{F}_k|_{\mathbf{I}_{\mathcal{F}} \times \mathbf{J}_{\mathcal{F}}}$ is purely the

ALGORITHM 5: Function $\mathbf{I}_{\mathcal{U}} = \texttt{indxF2U}(\mathbf{I}_{\mathcal{F}})$.

```
do ii = 1,length(I_F)
    I_U(ii) = -1      % initialized to be -1
    do jj = 1,N_d
        hj = map(jj),   tj = map(jj)+ind(jj+1)-ind(jj)-1
        if (I_F(ii) >= hj .and.  I_F(ii) <= tj), then
            I_U(ii) = I_F(ii) - hj + ind(jj);
            break;
        end if
    end do
end do
```

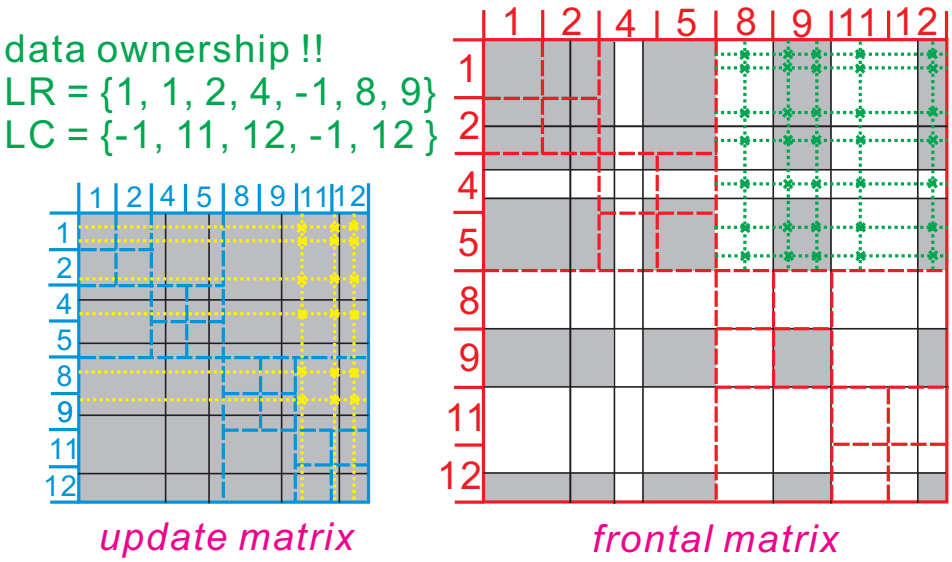

FIG. 5. *The illustration for recovering D and B generators through index mapping.*

submatrix $\mathcal{U}_d|_{\mathbf{I}_{\mathcal{U}} \times \mathbf{J}_{\mathcal{U}}}$. Therefore, the problem has been converted to how to efficiently compute the submatrix $\mathcal{U}_d|_{\mathbf{I}_{\mathcal{U}} \times \mathbf{J}_{\mathcal{U}}}$. We point out that if $\mathcal{U}_d$ is a dense matrix as appears in the conventional multifrontal solver, to evaluate $\mathcal{U}_d|_{\mathbf{I}_{\mathcal{U}} \times \mathbf{J}_{\mathcal{U}}}$ is a straightforward process by extraction. However, since $\mathcal{U}_d$ is currently in its HSS representations, to evaluate $\mathcal{U}_d|_{\mathbf{I}_{\mathcal{U}} \times \mathbf{J}_{\mathcal{U}}}$ we need to use matrix-matrix multiplications to recover its off-diagonal values.

A naive way to compute $\mathcal{U}_d|_{\mathbf{I}_{\mathcal{U}} \times \mathbf{J}_{\mathcal{U}}}$ is to loop over all points in $\mathbf{I}_{\mathcal{U}} \times \mathbf{J}_{\mathcal{U}}$ and evaluate them point-wise. Bear in mind that there is an HSS tree and layout associated with $\mathcal{U}_d$. If some points belong to a particular diagonal block D_i, the evaluation is just by extraction. Otherwise, the extraction will be a series of matrix-vector multiplications via traversing the HSS tree $\mathcal{T}$. We note that point-wise evaluation of $\mathcal{U}_d|_{\mathbf{I}_{\mathcal{U}} \times \mathbf{J}_{\mathcal{U}}}$ is a `BLAS2` operation, which is considered to be inefficient compared with `BLAS3` operations. Thus we adopt a block-wise (`BLAS3`) computation strategy, by taking advantage of the intrinsic HSS structure.

We revisit the HSS structure introduced in equation (3.1) that off-diagonal blocks are represented hierarchically by HSS generators which are associated with nodes on the HSS tree $\mathcal{T}$, and upper level generators are represented by lower level ones. Thus by resorting to the concept of data ownership, we want to seek which leaf node owns a portion of indices contained in $\mathbf{I}_{\mathcal{U}}$ and $\mathbf{J}_{\mathcal{U}}$, rather than only one point. This is where blocking technique comes into play. Thus we consider another

mapping named `indx2leaf`, which maps indices in $\mathbf{I}_{\mathcal{U}}$ and $\mathbf{J}_{\mathcal{U}}$ to the leaf node that owns it:

$$LR = \texttt{indx2leaf}\,(\mathbf{I}_{\mathcal{U}})\,, \qquad LC = \texttt{indx2leaf}\,(\mathbf{J}_{\mathcal{U}})\,. \tag{4.8}$$

where LR and LC are collections of the mapped leaf nodes that own indices in $\mathbf{I}_{\mathcal{U}}$ and $\mathbf{J}_{\mathcal{U}}$. If there is no leaf node that own a particular index, the mapped leaf node is labeled -1. An example of mapping LR and LC is illustrated by Figure 5. Algorithm 6 summarizes implementation details of `indx2leaf`, in which `range`(i) is a function which returns the first and the last index that the node i owns.

ALGORITHM 6: Function $LR = \texttt{indx2leaf}\,(\mathbf{I}_{\mathcal{U}})$.

$LR = -1$;
do i = 1,length($\mathcal{T}$)
  if (i is a leaf node), then
    do jj = 1,length($\mathbf{I}_{\mathcal{U}}$)
      if ($\mathbf{I}_{\mathcal{U}}(jj) \in$ range(i)), then
        $LR(jj) = i$;
      end if
    end do
  end if
end do

By exploiting LR and LC, we can group indices together that a particular leaf node i owns, which enables us to use `BLAS3` rather than `BLAS2`. The strategy can be summarized into the following algorithm 7.

ALGORITHM 7: Computing $\mathcal{U}_d|_{\mathbf{I}_{\mathcal{U}} \times \mathbf{J}_{\mathcal{U}}}$.

do i = 1,length($\mathcal{T}$)
  if (i is a leaf-node), then
    1. determine $[\mathbf{I}_i, \mathbf{J}_i]$ out of $[\mathbf{I}_{\mathcal{U}}, \mathbf{J}_{\mathcal{U}}]$ that i owns, by exploiting LR and LC;
    2. evaluate $\mathcal{U}_d|_{\mathbf{I}_i \times \mathbf{J}_i} = D_i|_{\widehat{\mathbf{I}}_i \times \widehat{\mathbf{J}}_i}$ ($\widehat{\mathbf{I}}_i$ and $\widehat{\mathbf{J}}_i$ are local indices within D_i);
    3. extract $\widehat{U}_i = U_i|_{\widehat{\mathbf{I}}_i}$ and $\widehat{V}_i = V_i|_{\widehat{\mathbf{J}}_i}$;
  else
    1. evaluate $\mathcal{U}_d|_{\mathbf{I}_{c_1} \times \mathbf{J}_{c_2}} = \widehat{U}_{c_1} B_{c_1} \widehat{V}_{c_2}^T$ and $\mathcal{U}_d|_{\mathbf{I}_{c_2} \times \mathbf{J}_{c_1}} = \widehat{U}_{c_2} B_{c_2} \widehat{V}_{c_1}^T$;
    2. merge $\widehat{U}_i = \begin{pmatrix} \widehat{U}_{c_1} R_{c_1} \\ \widehat{U}_{c_2} R_{c_2} \end{pmatrix}$ and $\widehat{V}_i = \begin{pmatrix} \widehat{V}_{c_1} W_{c_1} \\ \widehat{V}_{c_2} W_{c_2} \end{pmatrix}$;
  end if
end do

5. Conclusions. We present the framework of the structured multifrontal solver via randomized sampling, which successfully avoids the extend-add operation on dense update matrices. Instead, we only conduct the extend-add on three tall skinny matrices that consist of one random matrix and two sampling matrices. By resorting to the randomized sampling approach, we do not form dense frontal matrices at each stage to carry out the HSS construction, which plays a key role in the memory reduction. We develop an efficient algorithm by retrieving D and B generators from update matrices that are in HSS forms, via matrix blocking techniques. Data ownership is the key concept for developing this new algorithm.

Acknowledgements. The authors thank the members, ConocoPhillips, ExxonMobil, Total, BGP, PGS and Statoil, of the Geo-Mathematical Imaging Group (GMIG) for partial financial support. J. Xia was supported in part by NSF grants DMS-1115572 and CHE-0957024, M.V. de Hoop was supported in part by NSF CMG grant DMS-1025318, and X.S. Li was supported in part by the Director, Office of Science, Office of Advanced Scientific Computing Research of the U.S. Department of Energy under the contract DE-AC02-05CH11231. The authors thank the National Energy Research Scientific Computing Center (NERSC) at Lawrence Berkeley National Laboratory (LBNL) for providing the computing resources.

REFERENCES

[1] S. Börm, L. Grasedyck, and W. Hackbusch, *Introduction to hierarchical matrices with applications*, Eng. Anal. Bound. Elem, 27 (2003), pp. 405–422.
[2] S. Chandrasekaran, M. Gu, and T. Pals, *A fast ULV decomposition solver for hierarchically semiseparable representations*, SIAM J. Matrix Anal. Appl., 28 (2006), pp. 603–622.
[3] I.S. Duff and J.K. Reid, *The multifrontal solution of indefinite sparse symmetric linear equations*, ACM Trans. Math. Software, 9 (1983), pp. 302–325.
[4] J.A. George, *Nested dissection of a regular finite element mesh*, SIAM J. Numer. Anal., 10 (1973), pp. 345–363.
[5] M. Gu and S.C. Eisenstat, *Efficient algorithms for computing a strong rank-revealing qr factorization*, SIAM J. Sci. Comput., 17 (1996), pp. 848–869.
[6] S. Li, M. Gu, C. Wu, and J. Xia, *New efficient and robust HSS Cholesky factorization of SPD matrices*, submitted to SIAM J. Matrix Anal. Appl., (2011).
[7] L. Lin, J. Lu, and L. Ying, *Fast construction of hierarchical matrix representation from matrix-vector multiplication*, Journal of Computational Physics, to appear (2012).
[8] J.W.H. Liu, *The multifrontal method for sparse matrix solution: Theory and practice*, SIAM Review, 34 (1992), pp. 82–109.
[9] P.G. Martinsson, *A fast randomized algorithm for computing a hierarchically semi-separable representation of a matrix*, submitted, (2011).
[10] P.G. Martinsson, V. Rokhlin, and M. Tygert, *A randomized algorithm for the approximation of matrices*, Tech. Report 1361, Department of Computer Science, Yale University, New Haven, CT, 2006.
[11] ———, *A randomized algorithm for the decomposition of matrices*, Applied and Computational Harmonic Analysis, 30 (2011), pp. 47–68.
[12] V.Y. Pan and G. Qian, *Randomized preprocessing of homogeneous linear systems of equations*, Linear Algebra Appl., 432 (2010), pp. 3272–3318.
[13] L. Grasedyck W. Hackbusch and S. Börm, *An introduction to hierarchical matrices*, Math. Bohem., 127 (2002), pp. 229–241.
[14] S. Wang, M.V. de Hoop, and J. Xia, *On 3D modeling of seismic wave propagation via a structured parallel multifrontal direct Helmholtz solver*, (2011).
[15] S. Wang, X.S. Li, J. Xia, Y.C. Situ, and M.V. de Hoop, *Efficient scalable algorithms for hierarchically semiseparable matrices*, submitted to SIAM Journal of Scientific Computing, (2011).
[16] J. Xia, *Efficient structured multifrontal factorization for large sparse matrices*, preprint, `http://www.math.purdue.edu/~xiaj/work/mfhss.pdf`, (2010).
[17] ———, *Randomized sparse direct solvers*, preprint, to be submitted to SIAM J. Matrix Anal. Appl., (2011).
[18] 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.
[19] ———, *Fast algorithms for hierarchically semiseparable matrices*, Numer. Linear Algebra Appl., 2010 (2010), pp. 953–976.
[20] J. Xia, Y. Xi, and M. Gu, *A superfast structured solver for Toeplitz linear systems via randomized sampling*, submitted to SIAM J. Matrix Anal. Appl., (2011).