NEW EFFICIENT AND ROBUST HSS CHOLESKY FACTORIZATION OF SPD MATRICES

SHENGGUO LI*, MING GU†, CINNA JULIE WU†, AND JIANLIN XIA‡

Abstract. In this paper, we propose a robust Cholesky factorization method for symmetric positive definite (SPD), hierarchically semiseparable (HSS) matrices. Classical Cholesky factorizations and some semiseparable methods need to sequentially compute Schur complements. In contrast, we develop a strategy involving orthogonal transformations and approximations which avoids the explicit computation of the Schur complement in each factorization step. The overall factorization requires fewer floating point operations and has better data locality when compared to the recent HSS method in [SIAM J. Matrix Anal. Appl., 31(2010), 2899-2920]. Our strategy utilizes a robustness technique so that an approximate generalized Cholesky factorization is guaranteed to exist.

We test three different methods for compressing the off-diagonal blocks in each iteration, i.e., rank-revealing QR, SVD, and SVD with random sampling. In our comparisons, we find that, with high probability, using SVD with random sampling is fast and stable. The complexity of the methods proposed in this paper is analyzed and shown to be $O(N^2k)$, where N is the dimension of matrix and k is the maximum off-diagonal (numerical) rank. Numerical results from applications show the efficiency of our method and its effectiveness as a preconditioner. Moreover, our techniques are helpful in improving the scalability and robustness of other rank structured methods.

Key words. robust preconditioner, hierarchical semiseparable matrices, random algorithm, RRQR, SVD

AMS subject classifications. 15A12, 65F05, 65F30

1. Introduction.

Efficient and reliable structured matrix computations have been an intensive focus of recent research. Numerically stable fast and superfast algorithms have been developed for structured matrices such as Toeplitz matrices, Vandermonde matrices, and various forms of semi-separable matrices. In this paper, we are concerned with the rapid computation of effective preconditioners for *symmetric positive definite* (SPD) matrices via structured matrix techniques. Given an SPD matrix A, we are interested in the rapid computation of an SPD, *hierarchical semi-separable* (HSS) matrix S such that

$$A = S + O(\tau), \tag{1.1}$$

where τ is a user-prescribed tolerance. (See Section 3 for a precise definition of an HSS matrix.)

The HSS matrix structure was first discussed in [5, 6] and arose from an algebraic abstraction of the fast integral equation solver developed in [30]. More broadly, the HSS matrix is closely related to other rank-structured matrices such as the $\mathscr{H}$ [20, 22], $\mathscr{H}^2$ [23, 21], quasiseparable [11, 31, 32], and sequentially semiseparable (SSS) [5, 6] matrices. Among other things, some of these matrix structures, such as the HSS, $\mathscr{H}$, and $\mathscr{H}^2$ matrices, have proven to be invaluable tools in the fast numerical solutions of integral equations. More recently, they have been shown to play central roles in the superfast direct factorization and preconditioning of certain classes of large,

*College of Science, National University of Defense Technology, Changsha, China (nudtlsg@gmail.com).

†Department of Mathematics, University of California, Berkeley, CA 47920, USA (mgu@math.berkeley.edu, cinnawu@math.berkeley.edu).

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

sparse matrices [14, 15, 20, 21, 34]. These developments are the main driving force for developing a fast, reliable method to solve (1.1).

The semi-separable matrix structures share the common feature that all their off-diagonal blocks have rapidly decaying singular values. Thus, the numerical ranks of off-diagonal blocks are significantly smaller than the matrix dimensions [8, 9, 12, 27, 29, 1, 2, 5]. Recently, Xia and Gu have proposed an efficient algorithm that computes the approximation

$$A = R^\top R + O(\tau), \tag{1.2}$$

where $R^\top R$ is an HSS matrix and R is upper triangular [36]. This algorithm costs $O(N^2 k)$ floating operations (*flops*), where N is the order of A and k is the HSS rank. The main attractiveness over the algorithms in [5, 6, 9, 26] is that all Schur complements of A are kept SPD throughout the computation, thereby ensuring the existence of R for any given positive τ value. This robustness characteristic, often referred to as *Schur-monotonic*, is achieved by an approximation process, whereby the difference between the approximated and true Schur complement is a small, non-negative definite matrix. This technique is referred to as *Schur compensation* in [36].

In this paper, we propose a new algorithm for computing the approximation S in (1.1), where $S = PP^\top$ is a generalized Cholesky factorization with P an HSS matrix, computed through a sequence of Householder transformations and Cholesky factorizations. Our algorithm is designed to be Schur-monotonic and free of any *direct* Schur complement computations, resulting in faster computation and better data locality.

Recent work has suggested the efficiency and effectiveness of utilizing randomized algorithms [24, 37, 17, 18] for low-rank matrix compression during the HSS matrix construction. As pointed out in [17, 18], some of these randomized algorithms are equivalent to subspace iteration methods with an excellent start matrix, and the feature that the off-diagonal blocks of A have rapidly decaying singular values allows such algorithms to compute low-rank approximations quickly. Our algorithm adopts a reliable version of the randomized algorithm that maintains Schur-monotonicity and yet allows fast low-rank compression. When the matrix is large, our algorithm is much faster than that of [36]. (See Section 4 for details.) Note that Martinsson [26] has developed an efficient algorithm to approximately construct HSS matrices using random sampling techniques. However, this algorithm does not appear to maintain Schur-monotonicity during the factorization process, and can produce an indefinite HSS approximation even when the original matrix is SPD.

Depending on the tolerance level, the matrix S in (1.1) can either be used as a matrix factorization for a rapid linear system solver or as a preconditioner in the context of preconditioned conjugate gradient iterations. Compared with that of [36], our HSS matrix S, is typically a better preconditioner than the matrix $R^\top R$ in (1.2), requiring much fewer iterations.

1.1. Organization of paper. The paper is organized as follows. In Section 2, we briefly review the HSS structure and describe some standard matrix factorization methods, including the random sampling method, for low-rank matrix approximation. In Section 3, we present our generalized HSS Cholesky factorization method, related algorithms, and complexity analysis. We present numerical results in Section 4 and conclusions and future work in Section 5.

2. Preliminaries. In this section, we introduce some notation and give a brief introduction to the key concepts of HSS structures. We also describe some standard

low-rank matrix approximation methods, including random sampling methods, which will be used to compress off-diagonal blocks.

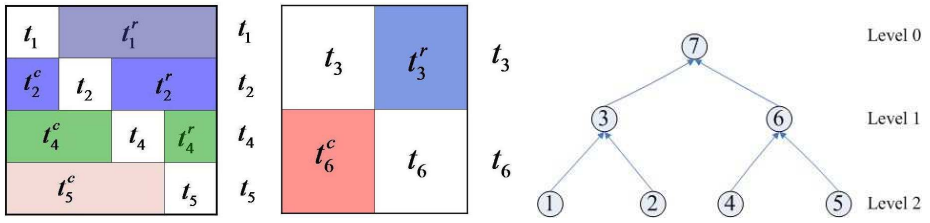
2.1. Notation and terminology. In this paper, $\mathcal{T}$ is a full binary tree. The root of $\mathcal{T}$ is denoted by $\mathrm{root}(\mathcal{T})$, and for each node i, $\mathrm{sib}(i)$ and $\mathrm{par}(i)$ denote the sibling and parent of i. If i is a non-leaf node, we represent the left and right child of i with i_1 and i_2, respectively. For our purposes, $\mathcal{T}$ is assumed to be *postordered*. That is, the nodes are ordered so that non-leaf nodes i satisfy the ordering $i_1 < i_2 < i$. We assume the levels of $\mathcal{T}$ are ordered *top-down*. In other words, $\mathrm{root}(\mathcal{T})$ is at level 0 and the leaves of $\mathcal{T}$ are at the largest level; see Figure 2.1(c).

Let $A \in \mathbb{R}^{N\times N}$ be a symmetric matrix with indexing set $\mathcal{I} := \{1, \ldots, N\}$. For a subset t_i of $\mathcal{I}$, let t_i^c be the set of all indices less than those of t_i and t_i^r be the set of all indices greater than those of t_i; then, $\mathcal{I} = t_i^c \cup t_i \cup t_i^r$. Allow $A_{t_i t_j}$ to represent the submatrix of A with row index set t_i and column index set t_j.

2.2. Introduction to symmetric HSS matrices. We introduce the postordered HSS form of a *symmetric* matrix A; see [36] for the general case. The HSS representation of A depends on a recursive partitioning of the rows and columns. Since A is symmetric, we assume the rows and columns have the same partitioning, and it is understood that the ith partition of A refers to both the ith row and column partition. As in [36], the partitioning is organized via a full, postordered binary tree $\mathcal{T}$; i.e., the ith node of $\mathcal{T}$ corresponds to the ith partition of A. The indices of the ith partition of A are contiguous and satisfy the following:

- $t_i \cup t_{\mathrm{sib}(i)} = t_{\mathrm{par}(i)}$,
- $t_{\mathrm{root}(\mathcal{T})} = \mathcal{I}$.

Figure 2.1 illustrates these sets.



(a) Bottom level (b) Level 1 (c) HSS postordering tree

FIG. 2.1. *Matrix partition and the corresponding index sets and binary tree $\mathcal{T}$.*

Each node i of $\mathcal{T}$ is associated with a set of matrices D_i, U_i, R_i, B_i called *generators*, where B_i is empty if i is a right child. The generators satisfy the recursive relationships

$$D_i = \begin{pmatrix} D_{i_1} & U_{i_1} B_{i_1} U_{i_2}^\top \\ U_{i_2} B_{i_1}^\top U_{i_1}^\top & D_{i_2} \end{pmatrix}, \; U_i = \begin{pmatrix} U_{i_1} R_{i_1} \\ U_{i_2} R_{i_2} \end{pmatrix}, \tag{2.1}$$

where $D_i \equiv A_{t_i t_i}$. For example, a 4×4 block HSS form looks like

$$\begin{pmatrix} D_1 & U_1 B_1 U_2^\top & U_1 R_1 B_3 R_4^\top U_4^\top & U_1 R_1 B_3 R_5^\top U_5^\top \\ U_2 B_1^\top U_1^\top & D_2 & U_2 R_2 B_3 R_4^\top U_4^\top & U_2 R_2 B_3 R_5^\top U_5^\top \\ U_4 R_4 B_3^\top R_1^\top U_1^\top & U_4 R_4 B_3^\top R_2^\top U_2^\top & D_4 & U_4 B_4 U_5^\top \\ U_5 R_5 B_3^\top R_1^\top U_1^\top & U_5 R_5 B_3^\top R_2^\top U_2^\top & U_5 B_4^\top U_4^\top & D_5 \end{pmatrix}, \tag{2.2}$$

and the corresponding HSS tree is shown in Figure 2.1(c). Following the notation of [36], a block row (column) of A excluding the diagonal block is called an *HSS block row (column)*, or simply *HSS block*. For instance, the ith HSS block row and block column are

$$H_i^{row} = \begin{pmatrix} A_{t_i t_i^c} & A_{t_i t_i^r} \end{pmatrix} \quad \text{and} \quad H_i^{col} = \begin{pmatrix} A_{t_i^c t_i} \\ A_{t_i^r t_i} \end{pmatrix},$$

respectively. In this paper, our algorithm is introduced using HSS block rows; the discussions for HSS block columns are similar. We call the maximum (numerical) rank of all HSS blocks the *HSS rank* of the matrix.

Many efficient algorithms have been developed for working with matrices represented or approximated by HSS structures. As shown in [9], there exist $O(N)$ algorithms for solving an HSS linear system. To clearly describe such HSS algorithms, we review the definition of a *visited set* [36].

DEFINITION 2.1. *The visited set associated with a node i of a postordered binary tree $\mathcal{T}$ is*

$$\mathcal{V}_i := \{j | j \text{ is a left node and } \mathrm{sib}(j) \in \mathrm{pred}(i)\}, \tag{2.3}$$

where $\mathrm{pred}(i)$ *is the set of predecessors associated with node i, i.e.,*

$$\mathrm{pred}(i) = \begin{cases} \{i\}, & \text{if } i = \mathrm{root}(\mathcal{T}), \\ \{i\} \cup \mathrm{pred}(\mathrm{par}(i)), & \text{otherwise.} \end{cases}$$

The set $\mathcal{V}_i$ can be interpreted as the stack before the visit of i in the postordering traversal of $\mathcal{T}$ [36]. For example, we have

$$\mathcal{V}_4 = \mathcal{V}_6 = \{3\},\ \mathcal{V}_5 = \{3,4\},\ \mathcal{V}_{11} = \mathcal{V}_{13} = \{7,10\},\ \mathcal{V}_{12} = \{7,10,11\};$$

see Figure 2.2.

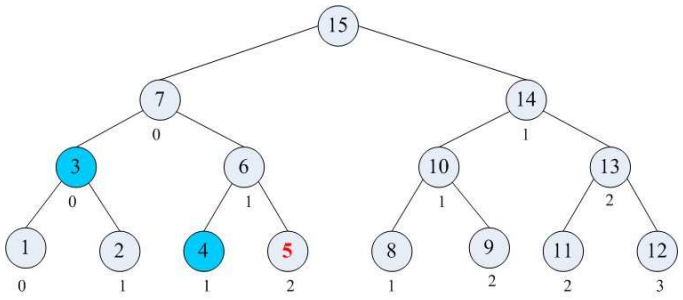


FIG. 2.2. *The visited set $\mathcal{V}_5$. (The number under each node i in Figure 2.2 denotes the cardinality s_i of $\mathcal{V}_i$.)*

We later use the following theorem involving $\mathcal{V}_i$ to describe and analyze the complexity of the HSS construction algorithm.

THEOREM 2.2. *[33] Let $\mathcal{T}$ be a perfect binary tree with levels ordered top-down and s_i be the cardinality of $\mathcal{V}_i$. Then for any node i at level l, $1 \le s_i \le l$. Moreover, there are exactly $f_j^l := \binom{l}{j}$ nodes i at level l with $s_i \equiv j$ for some $1 \le j \le l$.*

2.3. Low-rank matrix approximation. In our experiments we use three different methods for computing low-rank approximations to a matrix $B \in \mathbb{R}^{m\times n}$. Each method can be computed by setting a tolerance τ or an explicit rank k. That is, we have the following two types of approximations:

- *Fixed-precision approximation:* Seek matrices $U \in \mathbb{R}^{m\times r_\tau}$ and $T \in \mathbb{R}^{r_\tau\times n}$ such that

$$\|B - UT\|_2 \le \tau, \tag{2.4}$$

  where r_τ is determined by τ.
- *Fixed-rank approximation:* Seek matrices $U \in \mathbb{R}^{m\times r}$ and $T \in \mathbb{R}^{r\times n}$ such that

$$\|B - UT\|_2 = \min_{\mathrm{rank}(X)\le k} \|B - X\|_2. \tag{2.5}$$

The first method we use is a rank-revealing QR (RRQR) factorization. It is well known that RRQR can be used to compute low-rank approximations [3, 16] since RRQR allows one to (approximately) factor B as

$$BP \approx QR,$$

where $Q \in \mathbb{R}^{m\times k}$ has orthonormal columns, $R \in \mathbb{R}^{k\times n}$ is upper triangular, and $P \in \mathbb{R}^{n\times n}$ is a permutation matrix. The second method we use is the commonly used truncated singular value decomposition (SVD) [13] where

$$B \approx U\Sigma V^\top$$

with $U \in \mathbb{R}^{m\times k}, V \in \mathbb{R}^{n\times k}$, and $\Sigma \in \mathbb{R}^{k\times k}$.

Lastly, we use a randomized algorithm to compute the low-rank approximations. In general, such randomized algorithms are divided into two stages [25, 24]. First, a low-dimensional subspace approximately spanning the range of B is constructed. Then, the desired matrix decomposition is computed on a reduced matrix.

Stage A: Compute an approximate low-rank basis $Q \in \mathbb{R}^{m\times k}$ of the range of B such that Q has orthonormal columns and

$$B \approx QQ^*B.$$

Stage B: Compute the desired matrix decomposition on the smaller matrix $C := Q^*B \in \mathbb{R}^{k\times n}$.

The HSS construction requires computing low-rank approximations of H_i^{row} (or H_i^{col}) satisfying (2.4) or (2.5). This can be achieved by approximating the orthonormal row (or column) bases. Thus, it is enough to compute the basis Q in **Stage A**. The following algorithm, equivalent to the random SVD algorithm proposed in Section 5.2 of [28], is used to quickly find Q.

ALGORITHM 1. (Random low-rank approximation) Choose an l such that $k \le l < \min\{m, n\}$ where k is an approximation for the rank of B.

1. Draw an $n \times l$ random matrix Ω whose entries are Gaussian random variables with zero mean and unit variance. Compute the sample matrix

$$Y = B\Omega.$$

2. Let $Q \in \mathbb{R}^{m\times k}$ consist of the left singular vectors correspond to the k largest singular values of Y. This can be computed using an SVD where

$$Y = B\Omega = U\Sigma V^\top = [Q \mid P]\Sigma V^\top.$$

Here, $U \in \mathbb{R}^{m\times l}$ and $V \in \mathbb{R}^{l\times l}$ have orthonormal columns, Σ is an $l \times l$ nonnegative diagonal matrix, and $P \in \mathbb{R}^{m\times(l-k)}$.
3. Let $U = Q$ and $T = U^*B$. Then, UT is a low-rank approximation of B.

The following theorem, summarized from [28], says that QQ^*B closely approximates B with very high probability for small values of p as long as the $(k+1)$st singular value of B is small. For instance, we can choose $p = 8, 10$.

THEOREM 2.3. *Let $B \in \mathbb{R}^{m\times n}$, k and p be positive integers such that $1 \le k \le k+p \le \min\{m,n\}$, and $\Omega \in \mathbb{R}^{n\times(k+p)}$ be a Gaussian random matrix with zero mean and unit variance. Let Q be the $m \times k$ matrix computed from Algorithm 1 and σ_{k+1} be the $(k+1)$st largest singular value of B. Then*

$$\|B - QQ^*B\|_2 \le 10\sigma_{k+1}\sqrt{(k+p)n},$$

with probability at least $1-\phi(p)$ for a decreasing function ϕ.

REMARK 1.

1. The function ϕ decreases rapidly. For example, $\phi(8) < 10^{-5}$ and $\phi(20) < 10^{-17}$.
2. While faster algorithms, such as the subsampled random Fourier Transform, could in theory be used to reduce the cost of low-rank approximation (see [24]), we have not found such algorithms to be more efficient in our experiments.
3. In our experiments, Algorithm 1 is usually faster than the deterministic algorithms RRQR and SVD.

3. Generalized HSS Cholesky Factorization for SPD matrices. In this section, we discuss our new algorithm for computing a generalized HSS Cholesky factorization. We begin with a simple 2×2 block partitioning of an $N \times N$ SPD matrix A where

$$A = \begin{pmatrix} A_{11} & A_{12} \\ A_{12}^\top & A_{22} \end{pmatrix}, \tag{3.1}$$

with $A_{12} \in R^{m\times(N-m)}$ and $m \le \frac{N}{2}$. We will assume the off-diagonal submatrix A_{12} has rapidly decaying singular values. Thus, A_{12} is a *low-rank matrix* up to a given tolerance $\tau > 0$. Our approach exploits this low-rank property.

To motivate this approach, we introduce the scheme developed in [36]. First compute the Cholesky factorization of A_{11} as $L_{11}L_{11}^\top$, and let $L_{21} = A_{12}^\top L_{11}^{-\top}$. Then A can be factored as

$$A = \begin{pmatrix} L_{11} & \\ L_{21} & I \end{pmatrix} \begin{pmatrix} L_{11}^\top & L_{21}^\top \\ & S \end{pmatrix},$$

where $S = A_{22} - L_{21}L_{21}^\top$ is the Schur complement. The computation of this factorization can be sped up by taking the truncated SVD of $L_{21}^\top$. We have

$$L_{21}^\top = (U \quad \hat{U}) \begin{pmatrix} \Sigma & \\ & \hat{\Sigma} \end{pmatrix} \begin{pmatrix} V^\top \\ \hat{V}^\top \end{pmatrix} = U\Sigma V^\top + \hat{U}\hat{\Sigma}\hat{V}^\top = U\Sigma V^\top + O(\tau), \tag{3.2}$$

where $\Sigma = \mathrm{diag}(\sigma_1, \ldots, \sigma_k)$, $\hat{\Sigma} = \mathrm{diag}(\sigma_{k+1}, \ldots, \sigma_m)$, and $\sigma_k \geq \tau \geq \sigma_{k+1}$. Then $U\Sigma V^\top$ is the τ-truncated SVD of $L_{21}^\top$ and can be used to approximate the Schur complement S by

$$\tilde{S} = A_{22} - V\Sigma^2 V^\top. \tag{3.3}$$

Since $\tilde{S} = A_{22} - L_{21}L_{21}^\top + \hat{V}\hat{\Sigma}^2\hat{V}^\top = S + \hat{V}\hat{\Sigma}^2\hat{V}^\top$, $\tilde{S}$ is always SPD for any tolerance τ. Thus, one can continue the Cholesky factorization on $\tilde{S}$ in the same fashion, and an approximate HSS factorization of A is guaranteed for any $\tau > 0$. See [36] for more details.

In this paper, we take a different approach. Instead of keeping track of the approximate Schur complement $\tilde{S}$ throughout the computation, we completely avoid any explicit computation of the Schur complements throughout the generalized Cholesky factorization.

We again use the matrix (3.1) to illustrate the main idea of our new algorithm. To this end, we only factorize part of the first block row. There are two phases in this algorithm: *compression* and *merging*. The main idea is to find an orthonormal matrix $\mathscr{U}$ such that the Cholesky factorization of $\mathscr{U}^\top A \mathscr{U}$ be approximately computed without calculating the Schur complement.

Compute the Cholesky factorization $A_{11} = L_1 L_1^\top$ and an orthogonal decomposition $L_1^{-1}A_{12} = Q_1 W_1 + Q_2 W_2$, where $Q = [Q_1 \quad Q_2]$ is an orthonormal matrix with $Q_2 \in R^{m\times k}$ and $\|W_1\|_2 = O(\tau)$. Now further compute a QL factorization $U^{(1)}\hat{L} = L_1 Q$ which leads to

$$U^{(1)}\hat{L}\left(U^{(1)}\hat{L}\right)^\top = L_1 Q \left(L_1 Q\right)^\top = L_1 L_1^\top = A_{11},$$

and

$$A_{12} = L_1 Q \begin{pmatrix} W_1 \\ W_2 \end{pmatrix} = U^{(1)}\hat{L}\begin{pmatrix} W_1 \\ W_2 \end{pmatrix}.$$

Defining

$$\mathscr{U}_1 := \begin{pmatrix} U^{(1)} & \\ & I \end{pmatrix} \tag{3.4}$$

leads to

$$\mathscr{U}_1^\top \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} \mathscr{U}_1 = \begin{pmatrix} \hat{L}\hat{L}^\top & \hat{L}\begin{pmatrix} W_1 \\ W_2 \end{pmatrix} \\ \begin{pmatrix} W_1^\top & W_2^\top \end{pmatrix}\hat{L}^\top & A_{22} \end{pmatrix},$$

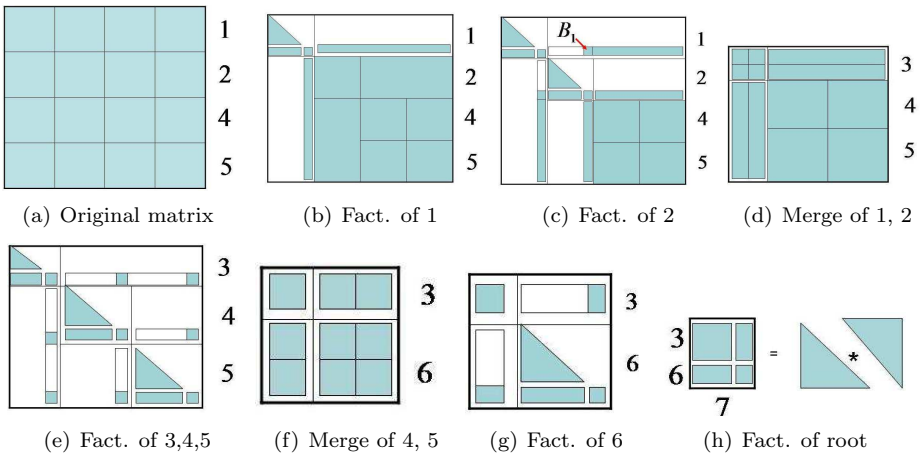
and the partitioning

$$\hat{L} = \begin{pmatrix} \hat{L}_{11} & \\ \hat{L}_{21} & \hat{L}_{22} \end{pmatrix}$$

yields

$$
\begin{aligned}
&\mathscr{U}_1^\top \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} \mathscr{U}_1 \\
&= \begin{pmatrix} \hat{L}_{11} & & \\ \hat{L}_{21} & I & \\ W_1^\top & & I \end{pmatrix} \begin{pmatrix} I & & \\ & \hat{L}_{22}\hat{L}_{22}^\top & \hat{L}_{22}W_2 \\ & W_2^\top \hat{L}_{22}^\top & A_{22} - W_1^\top W_1 \end{pmatrix} \begin{pmatrix} \hat{L}_{11} & & \\ \hat{L}_{21} & I & \\ W_1^\top & & I \end{pmatrix}^\top \\
&\approx \begin{pmatrix} \hat{L}_{11} & & \\ \hat{L}_{21} & I & \\ & & I \end{pmatrix} \begin{pmatrix} I & & \\ & \hat{L}_{22}\hat{L}_{22}^\top & \hat{L}_{22}W_2 \\ & W_2^\top \hat{L}_{22}^\top & A_{22} \end{pmatrix} \begin{pmatrix} \hat{L}_{11} & & \\ \hat{L}_{21} & I & \\ & & I \end{pmatrix}^\top .
\end{aligned}
$$

In the last equation, we have set W_1 to zero in each of the matrices, resulting in an error of $O(\tau)$ in the first and last matrices and an error of $O(\tau^2)$ in the center matrix. Since A is SPD, the center matrix in the last equation is also SPD for any $\tau > 0$. In the following context, we denote the compressed off-diagonal block of node i as $A^{(i)}_{\hat{t}_i t_i^r}$. For example, after the compression of node 1, we may use $A^{(1)}_{\hat{t}_1 t_2}$ to denote $\hat{L}_{22}W_2$. $A^{(1)}_{\hat{t}_1 t_2}$ is a matrix with fewer rows than $A_{t_1 t_2} (= A_{12})$.



(a) Original matrix (b) Fact. of 1 (c) Fact. of 2 (d) Merge of 1, 2

(e) Fact. of 3,4,5 (f) Merge of 4, 5 (g) Fact. of 6 (h) Fact. of root

FIG. 3.1. *The factorization process.*

This summarizes how to compress the $(1, 2)$ block in a 2×2 block partition setting. In general, the compression process is similar to that of [36]. The main difference of our algorithm is in the need to determine the HSS block row

$$
H_i^{row} := \left[A^{(j_1)T}_{\hat{t}_{j_1} t_i}, \ldots, A^{(j_{s_i})T}_{\hat{t}_{j_{s_i}} t_i} \mid A_{t_i t_i^r} \right],
$$

where $j_1, j_2, \ldots, j_{s_i}$ are the elements of the visited set $\mathcal{V}_i$ for a node i. Figure 3.2 illustrates how to obtain the HSS block row for $i = 2$. To compress H_i^{row}, we apply the above compression procedure to node i. Below, we summarize the general procedure for leaf nodes i, following the ordering of the postordered tree. Note that since A is symmetric, it is enough to work on the block rows in the upper triangular section of A.

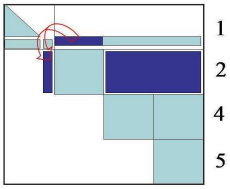


FIG. 3.2. *The second HSS block row*

ALGORITHM 2. (Compressing off-diagonal block rows) Suppose the SPD matrix A has been partitioned into n^2 blocks; i.e., there are n leaf nodes in the HSS tree $\mathcal{T}$. Assume the off-diagonal block corresponding to node i has m_i rows and rank k_i.

`for` $i = 1, 2, \ldots,$ root($\mathcal{T}$)
  `if` i is a leaf node
    1. Identify the ith diagonal block $A_{t_i t_i}$ and ith HSS block row

$$H_i^{row} = \left[A_{\hat{t}_{j_1} t_i}^{(j_1)T}, \ldots, A_{\hat{t}_{j_{s_i}} t_i}^{(j_{s_i})T} \mid A_{t_i t_i^r} \right],$$

    where $\mathcal{V}_i = \{j_1, j_2, \ldots, j_{s_i}\}$ is the visited set of node i.
    2. Compute the Cholesky factorization of $A_{t_i t_i} = L_i L_i^\top$.
    3. Compute the orthogonal decomposition $L_i^{-1} H_i^{row} = Q_1 W_1 + Q_2 W_2$, where $Q = [Q_1 \quad Q_2]$ is an orthonormal matrix with $Q_2 \in R^{m_i \times k_i}$, to obtain the *compression* $L_i^{-1} H_i^{row} \approx Q_2 W_2$.
    4. Compute $U^{(i)}$ from the QL decomposition $L_i Q = U^{(i)} \hat{L}^{(i)}$, and write

$$\hat{L}^{(i)} = \begin{pmatrix} \hat{L}_{11}^{(i)} & \\ \hat{L}_{21}^{(i)} & \hat{L}_{22}^{(i)} \end{pmatrix},$$

    where $\hat{L}_{22}^{(i)} \in R^{k_i \times k_i}$.
    5. Compute the HSS block $\widehat{H}_i = \hat{L}_{22}^{(i)} W_2$ and $D_i = \hat{L}_{22}^{(i)} \hat{L}_{22}^{(i)T}$.
  `end if`
`end for`

REMARK 2.

1. The matrices $U^{(i)}$, $\hat{L}_{11}^{(i)}$, $\hat{L}_{21}^{(i)}$, and the scalar k_i are the *generators* of node i and are stored to later reconstruct the preconditioner. The matrix D_i and remaining HSS block $\widehat{H}_i$ are passed to par(i).
2. In Step 3 of Algorithm 2, computing Q is a classical low-rank matrix approximation problem with many possible algorithms (RRQR, τ-truncated SVD, SVDR).
3. The HSS block H_i^{row} can be formed with the aid of the visited set $\mathcal{V}_i$: If node i is a left node, push i onto a stack $\mathcal{S}_v$; otherwise, pop an element from the stack. The elements of the stack $\mathcal{S}_v$ are exactly the nodes in $\mathcal{V}_i$ right before i is visited.
4. In Algorithm 2, we compress the off-diagonal block row H_i^{row}. We can similarly compress the off-diagonal block column H_i^{col}.

After compression, the first and second block rows, $A^{(1)}_{\hat{t}_1 t_4}$ and $A^{(2)}_{\hat{t}_2 t_4}$, are of full rank. However, when the two block rows are merged to form the matrix

$$H_3 = \begin{bmatrix} A^{(1)}_{\hat{t}_1 t_4} \\ A^{(2)}_{\hat{t}_2 t_4} \end{bmatrix},$$

H_3 may be again be of low-rank. Thus, our algorithm hierarchically compresses the off-diagonal blocks. The process is outlined in the next section.

3.1. Merging child blocks. For each parent node, we merge the appropriate blocks of its children together and compress it again. Take for example node 3, where the resulting blocks from nodes 1 and 2 are merged to form

$$\mathcal{A}_3 = \begin{bmatrix} D_1 & B_1 & A^{(1)}_{\hat{t}_1 t_4} \\ B_1^\top & D_2 & A^{(2)}_{\hat{t}_2 t_4} \\ A^{(1)\top}_{\hat{t}_1 t_4} & A^{(2)\top}_{\hat{t}_2 t_4} & A_{t_4 t_4} \end{bmatrix}, \tag{3.5}$$

where B_1 is obtained when compressing the second HSS block, see Figure 3.1(c). The size of the original matrix is then reduced. In general, for parent nodes we need to first determine the ith diagonal block D_i and ith HSS block H_i. In the case of node 3,

$$D_3 = \begin{bmatrix} D_1 & B_1 \\ B_1^\top & D_2 \end{bmatrix}, \; H_3 = \begin{bmatrix} A^{(1)}_{\hat{t}_1 t_4} \\ A^{(2)}_{\hat{t}_2 t_4} \end{bmatrix} \tag{3.6}$$

are formed by merging the appropriate blocks of the children, node 1 and node 2. For a general parent node i, the diagonal block D_i and its off-diagonal block H_i are of the form

$$D_i = \begin{bmatrix} D_{i_1} & B_{i_1} \\ B_{i_1}^\top & D_{i_2} \end{bmatrix}, \; H_i = \left[\begin{array}{ccc|c} [A^{(j_1)\top}_{\hat{t}_{j_1}\hat{t}_{i_1}} & \cdots & A^{(j_{s_i})\top}_{\hat{t}_{j_{s_i}}\hat{t}_{i_1}} & A^{(i_1)}_{\hat{t}_{i_1} t_i^r}] \\ \hline [A^{(j_1)\top}_{\hat{t}_{j_1}\hat{t}_{i_2}} & \cdots & A^{(j_{s_i})\top}_{\hat{t}_{j_{s_i}}\hat{t}_{i_2}} & A^{(i_2)}_{\hat{t}_{i_2} t_i^r}] \end{array}\right], \tag{3.7}$$

where i_1 and i_2 are the children of node i. The blocks $\left[A^{(j_1)\top}_{\hat{t}_{j_1}\hat{t}_{i_1}}, \ldots, A^{(j_{s_i})\top}_{\hat{t}_{j_{s_i}}\hat{t}_{i_1}}\right]$ and $\left[A^{(j_1)\top}_{\hat{t}_{j_1}\hat{t}_{i_2}}, \ldots, A^{(j_{s_i})\top}_{\hat{t}_{j_{s_i}}\hat{t}_{i_2}}\right]$ make up the leftmost block of HSS block row H_i which makes up the portion in front of D_i in $\mathcal{A}$.

The computation is continued in the manner of the leaf nodes; that is, we Cholesky factorize $D_i = L_i L_i^\top$ and compute the compression $L_i^{-1} H_i \approx Q_2 W_2 + O(\tau)$, where $Q = [Q_1\ Q_2]$ is orthonormal and $Q_2 \in R^{m_i \times k_i}$. The generators, $U^{(i)}$, $\tilde{L}^{(i)}_{11}$, $\tilde{L}^{(i)}_{21}$ and k_i are computed from $L_i Q = U^{(i)} \tilde{L}^{(i)}$. Traversing the HSS tree $\mathcal{T}$ in postorder, our algorithm alternates between *compressions* and *merges* until arriving at root($\mathcal{T}$). The complete *Generalized HSS Cholesky factorization* algorithm is summarized in Algorithm 3.

ALGORITHM 3. (Generalized HSS Cholesky factorization) Suppose A is an SPD matrix, and the HSS tree $\mathcal{T}$ has $N_\mathcal{T}$ nodes.

`for` $i = 1, \ldots, N_\mathcal{T} - 1$
    `if` i is a leaf node

1. Cholesky factorize $A_{t_i t_i} = L_i L_i^\top$, and form the ith HSS block row,

$$H_i^{row} = \left[A_{\hat{t}_{j_1} t_i}^{(j_1)\top}, \ldots, A_{\hat{t}_{j_{s_i}} t_i}^{(j_{s_i})\top} \mid A_{t_i t_i^r} \right],$$

where $\mathcal{V}_i = \{j_1, j_2, \ldots, j_{s_i}\}$ is the visited set of node i.
2. *Compress* $L_i^{-1} H_i^{row} \approx Q_2 W_2$, where $Q = [Q_1 \quad Q_2]$ is an orthonormal matrix.
3. Compute $U^{(i)}$ from $L_i Q = U^{(i)} \hat{L}^{(i)}$.
4. Factorize node i, and compute $\widehat{H}_i = \hat{L}_{22}^{(i)} W_2$, $D_i = \hat{L}_{22}^{(i)} \hat{L}_{22}^{(i)\top}$ (see Algorithm 2).
5. If node i is a right node, construct $B_{\mathrm{sib}\{i\}}$ from $\widehat{H}_i$.

`else`

1. *Merge* D_{i_1}, D_{i_2}, B_{i_1}, $\widehat{H}_{i_1}$, and $\widehat{H}_{i_2}$ to form

$$D_i = \begin{bmatrix} D_{i_1} & B_{i_1} \\ B_{i_1}^\top & D_{i_2} \end{bmatrix}, \; H_i = \left[\begin{array}{ccc|c} [A_{\hat{t}_{j_1} \hat{t}_{i_1}}^{(j_1)\top} & \cdots & A_{\hat{t}_{j_{s_i}} \hat{t}_{i_1}}^{(j_{s_i})\top} & A_{\hat{t}_{i_1} t_i^r}^{(i_1)}] \\ \hline [A_{\hat{t}_{j_1} \hat{t}_{i_2}}^{(j_1)\top} & \cdots & A_{\hat{t}_{j_{s_i}} \hat{t}_{i_2}}^{(j_{s_i})\top} & A_{\hat{t}_{i_2} t_i^r}^{(i_2)}] \end{array} \right].$$

2. Compute $D_i = L_i L_i^\top$ and compress $L_i^{-1} H_i$ using Algorithm 1, to obtain $U^{(i)}$, $\hat{L}^{(i)}$, k_i, and $\widehat{H}_i$.
3. If i is a right node, construct $B_{\mathrm{sib}\{i\}}$ from $\widehat{H}_i$.

`end if`

`end for`

Merge $D_{N_{\mathcal{T}} 1}$, $D_{N_{\mathcal{T}} 2}$, $B_{N_{\mathcal{T}} 1}$ to form $D_{N_{\mathcal{T}}}$.

Compute the Cholesky factorization $D_{N_{\mathcal{T}}} = L_{N_{\mathcal{T}}} L_{N_{\mathcal{T}}}^\top$.

See Figure 3.1 for an illustration of the entire process. As seen in Figure 3.1(a), the original matrix A is partitioned into 16 blocks; i.e., there are four leaf nodes in the HSS tree. Figure 3.1(b) represents the factorization of node 1 after compression; note that the first off-diagonal block row has been approximated by a low-rank matrix. The factorization of node 2 is represented in Figure 3.1(c), and the appropriate blocks of node 1 and node 2 are merged to form a smaller matrix (Figure 3.1(d)). Continuing the process in Figure 3.1(e), nodes 3, 4 and 5 are factorized. Nodes 4 and 5 are then merged to form node 6 as seen in Figure 3.1(f). Finally, node 6 is factorized and merged with node 3, which is then in turn factorized (Figure 3.1(h)).

3.2. HSS solver with generalized Cholesky factors. We briefly describe the HSS solver proposed in [35] for solving $Ax = b$ where A has a generalized Cholesky factorization organized by an HSS tree $\mathcal{T}$. As in a classical LU decomposition, the HSS solver involves a forward substitution and a backward substitution. Each node i of $\mathcal{T}$ has generators $U^{(i)}$, $\hat{L}_{11}^{(i)}$, $\hat{L}_{21}^{(i)}$ and k_i, where k_i is the approximate rank of node i. To solve the linear system, we traverse the HSS tree $\mathcal{T}$ in postorder to implement forward and backward substitution. We first partition b according to the bottom level (leaf) nodes, and denote the partition corresponding to leaf node i with b_i. Assume there are $N_{\mathcal{T}}$ nodes.

ALGORITHM 4. (Forward substitution)

`for` node $i = 1, \ldots, N_{\mathcal{T}} - 1$

`if` i is a leaf node

Compute

$$\hat{b}_i = U^{(i)\top} \cdot b_i = \begin{bmatrix} \hat{b}_{i;1} \\ \hat{b}_{i;2} \end{bmatrix} \begin{matrix} m_i - k_i \\ k_i \end{matrix}, \; b_i = \begin{bmatrix} \hat{L}_{11}^{(i)} & \\ \hat{L}_{21}^{(i)} & I \end{bmatrix}^{-1} \cdot \hat{b}_i = \begin{bmatrix} b_{i;1} \\ b_{i;2} \end{bmatrix} \begin{matrix} m_i - k_i \\ k_i \end{matrix}.$$

`else`

Form b_i from the lower sections of its children, i.e., $b_i = \begin{bmatrix} b_{i_1;2} \\ b_{i_2;2} \end{bmatrix}$, where i_1, i_2 are the left and right child of node i, respectively. Then compute

$$b_i = \begin{bmatrix} \hat{L}_{11}^{(i)} & \\ \hat{L}_{11}^{(i)} & I \end{bmatrix}^{-1} \cdot U^{(i)\top} \cdot b_i = \begin{bmatrix} b_{i;1} \\ b_{i;2} \end{bmatrix} \begin{matrix} m_i - k_i \\ k_i \end{matrix}.$$

`end if`

`end for`

Compute

$$b_{N_\mathcal{T}} = L_{N_\mathcal{T}}^{-1} \cdot b_{N_\mathcal{T}} \equiv \begin{bmatrix} b_{N_{\mathcal{T}1};2} \\ b_{N_{\mathcal{T}2};2} \end{bmatrix},$$

where $N_{\mathcal{T}1}, N_{\mathcal{T}2}$ are the left and right child of node $N_\mathcal{T}$, respectively.

After the forward substitution, each node has updated an b_i. Then, the solution x can be computed from b_i using backward substitution. The procedure is very similar to forward substitution with similar operation counts. We omit the details.

TABLE 3.1
Flops counts of some matrix operations.

Operation	Flops
Cholesky factorization of an $n \times n$ matrix	$\frac{n^3}{3}$
Inverse of an $n \times n$ lower triangular matrix times an $n \times k$ matrix	$n^2 k$
Product of a general $m \times n$ matrix and an $n \times k$ matrix	$2mnk$
QR factorization of an $m \times k$ tall matrix ($m > k$)	$2k^2(m - \frac{k}{3})$
QL factorization for an $n \times n$ matrix	$\frac{4}{3}n^3$
SVD of a general $m \times n$ matrix $A = U\Sigma V^*, m > n$, computing U, Σ	$4m^2 n$
Product of an $n \times n$ lower triangular matrix and an $n \times n$ upper triangular matrix	$\frac{2n^3}{3}$

3.3. Complexity of construction. We have the following complexity result for our algorithms: Assume A is an $N \times N$ SPD matrix and has been assigned a full HSS tree $\mathcal{T}$. Furthermore, assume the HSS rank of A is k, and at the bottom level, each leaf node has m rows, where m is of $O(k)$. Then the *generalized HSS Cholesky factorization* method has complexity of $O(N^2 k)$.

In the following discussion, assume $\mathcal{T}$ is ordered top-down with L levels so that the bottom level is at $L - 1$ and the root is at level 0. Since $\mathcal{T}$ is a full binary tree, $\mathcal{T}$ has $n := 2^{L-1}$ leaf nodes and a total of $2n - 1$ nodes. Moreover, assume

all off-diagonal blocks of A have rank k, and each leaf node contains m rows (that is, $N = mn$). Denote the set of leaf nodes by $LN := \{i|$ node i is a leaf node$\}$. We compute the cost level by level. Let N_i be the number of columns in $A_{t_i t_i^r}$. According to Theorem 2.2 in [33],

$$\sum_{i \text{ at level } l} s_i = \sum_{j=1}^{l} j \binom{l}{j} = \frac{1}{2} l 2^l, \tag{3.8}$$

$$\sum_{i \text{ at level } l} N_i = \sum_{j=1}^{2^l} \left(n - j\frac{n}{2^l}\right) m = \frac{1}{2} mn \left(2^l - 1\right). \tag{3.9}$$

The following illustrates the computation when using SVD to compress the HSS block rows. The major operations of our generalized HSS Cholesky factorization are as follows.

For each leaf node i (bottom level nodes):

- Cholesky factorization of $A_{ii} = L_i L_i^\top$ requires $\frac{m^3}{3}$ flops.
- Compressing $H_i^\top L_i^{-\top} = Q_i R_i L_i^{-\top}$ requires $2m^2(N_i - \frac{m}{3}) + m^3 + 2m^2 k s_i$ flops, where $H_i \in \mathbb{R}^{m \times (k \times s_i + N_i)}$ and s_i is the cardinality of $\mathcal{V}_i$.
- Computing $W_2^\top = Q_i U_1 \Sigma_1$, where $R_i L_i^{-\top} = [U_1 \quad U_2] \begin{bmatrix} \Sigma_1 & \\ & \Sigma_2 \end{bmatrix} V^\top$ and $V = [V_1 \quad V_2]$ with $V_1 \in R^{m \times k}$, requires $2N_i mk + m^3 + mk + 2mk^2 s_i$ flops.
- Computing $U^{(i)}$ from $L_i Q = U^{(i)} \hat{L}^{(i)}$ where $Q = [V_2 \quad V_1]$ requires $m^3 + \frac{4m^3}{3}$ flops.
- Computing $D_i = \hat{L}_{22}^{(i)} \hat{L}_{22}^{(i)\top}$ and $\widehat{H}_i = \hat{L}_{22}(i) W_2$ requires $\frac{2k^3}{3} + 2N_i k^2 + 2k^3 s_i$ flops.

Therefore, the total cost of all the leaf nodes is approximately

$$\begin{aligned}
C_f &= \sum_{i \in LN} \{2(m^2 + k^2 + mk)N_i + 4m^3 + \frac{2}{3}k^3 + 2(m^2 k + mk^2 + k^3)s_i\} \\
&= 2(m^2 + k^2 + mk) \sum_{i \in LN} N_i + 4m^3 \frac{N}{m} + \frac{2N}{3m} k^3 + 2(m^2 k + mk^2 + k^3) \sum_{i \in LN} s_i \\
&\approx (m^2 + k^2 + mk) n(n-1)m + 4Nm^2 + \frac{2}{3} Nk^2 + (m^2 k + mk^2 + k^3) nL \\
&\approx N^2 k + O(Nk^2) \approx O(N^2 k)
\end{aligned}$$

where $N = mn$ and $m = O(k)$. At level l, there are 2^l parent nodes, and there are a total of $n - 1$ non-leaf nodes. The analysis for non-leaf nodes is the same as for leaf nodes. The difference is that the main diagonal block of each parent node is a $2k \times 2k$ matrix; thus, $m = 2k$ in the above flop counts.

The complexity of each non-leaf node (except the root) is $14k^2 N_i + \frac{98}{3} k^3 + 14k^3 s_i$.

Summing over the levels between 0 and $L-1$,

$$\begin{aligned} C_p &= \sum_{l=1}^{L-2} \sum_{i \in \text{level } l} \{14k^2 N_i + \frac{98}{3}k^3 + 14k^3 s_i\} \\ &= 14k^2 \sum_{l=1}^{L-2} \frac{1}{2} N(2^l - 1) + \frac{98}{3}k^3(n-1) + 7k^3 \sum_{l=1}^{L-2} l2^l \\ &= 7k^2 N 2^{L-1} + \frac{98}{3}k^3(n-1) + 7k^3(n(L-3)+2) \\ &\approx 7N^2 k + O(Nk^2) \approx O(N^2 k), \end{aligned} \tag{3.10}$$

where $n = 2^{L-1}$, $N = mn$, and $m = O(k)$. The complexity of the root node is $C_r = \frac{(2k)^3}{3} < N^2 k$. Thus, the total complexity is $C = C_f + C_p + C_r = 8N^2 k + O(Nk^2)$.

Remark 3.

1. The complexity of the algorithm in [36] is also $O(N^2 k)$. However, in our numerical results, our algorithm requires fewer flops when using the same low-rank matrix approximation method for compression.
2. With modern computer architectures, floating-point operations are no longer the dominant factor in execution speed. Although the randomized algorithm SVDR requires more flops than RRQR and SVD, in our experience, SVDR is much faster.

4. Numerical results. As in [36], [35], we test the HSS preconditioner on the dense fill-in arising from the factorization of some sparse discretized PDE problems. We run our tests on a dense intermediate matrix instead of the entire sparse discretized matrix. Our algorithms were implemented in Matlab, and the following tests were ran on a server with 32GB memory, 8 Intel(R) X5460, and 3.16GHZ processors.

In the following, we refer to the structured Cholesky factorization proposed by Xia and Gu in [36] as XG's factorization. The factorization proposed in Algorithm 3 will be referred to as GHCF, short for *Generalized HSS Cholesky Factorization*. We first consider a linear elasticity equation.

Example 1.

$$\begin{aligned} -(\mu \Delta \mathbf{u} + (\lambda + \mu) \bigtriangledown \bigtriangledown \cdot \mathbf{u}) = \mathbf{f} \text{ in } \Omega = (0,1) \times (0,1) \\ \mathbf{u} = 0 \text{ on } \partial\Omega, \end{aligned}$$

where $\mathbf{u}$ is the displacement vector field, and λ and μ are the Lamé constants. If λ/μ is large, this PDE can be very ill-conditioned, as illustrated by the results in Table 4.1. We use nested dissection on a regular mesh, and consider the last Schur complement A corresponding to the top level separator in nested dissection during the factorization of the stiffness matrix. The dimension of A is $n = 2002$. The diagonal block size at the bottom level is $m \approx 60$. We use fixed-rank approximation methods to compress the off-diagonal blocks with a preset rank of $k = 15$. Table 4.1 gives the conditions numbers of A without preconditioning and preconditioned by the diagonal block preconditioner, XG's preconditioner [36], and our GHCF preconditioner. In this table, we use the following notation:

$\kappa(A)$ represents the condition number of A without preconditioning.
$\kappa(A_0)$ represents the condition number of A with the block diagonal preconditioner.
$\kappa_1(A_{15})$ represents the condition number of A with XG's preconditioner [36] and rank $k = 15$.

$\kappa_2(A_{15})$ represents the condition number of A with the GHCF preconditioner and rank $k = 15$.

From the results, we can see that the preconditioned matrix using our GHCF preconditioner becomes well-conditioned, with condition number always close to one.

TABLE 4.1
Example 1: The original condition number of A, and the condition numbers after preconditioning with the block diagonal, XG's, and GHCF preconditioners.

λ/μ	10	10^3	10^6	10^9	10^{12}
$\kappa(A)$	3.50e03	2.03e05	2.02e08	2.01e11	1.39e14
$\kappa(A_0)$	7.63e01	3.15e01	3.55e02	1.73e03	6.93e10
$\kappa_1(A_{15})$	1.16	4.43	1.79	1.80	2.57
$\kappa_2(A_{15})$	1.03	1.02	1.04	1.04	1.14

For this example, we compare the construction time of the three different low-rank approximation methods (RRQR, SVD, and SVDR) in Table 4.2 with a preset rank of $k = 10$. We use the matrix A corresponding to $\lambda/\mu = 10^{12}$. With different choices of the bottom level block sizes, we compare the construction time of using RRQR for compression and using SVD with and without randomized algorithm for compression. From the results in Table 4.2, we can see that the randomized algorithm can provide up to a three times speedup for this matrix A.

TABLE 4.2
Example 1: Comparison of the time (in seconds) when using different low-rank approximation methods for compression.

m	RRQR	SVD	SVDR
15	2.02	0.63	0.49
20	1.78	0.58	0.42
25	1.27	0.69	0.39
40	1.15	1.02	0.33
60	1.02	1.42	0.31
80	0.98	1.92	0.33

Example 2. In this example we consider the following problem defined on the unit square:

$$a(x,y)\frac{\partial^2 u}{\partial x^2} + 2b(x,y)\frac{\partial^2 u}{\partial x \partial y} + c(x,y)\frac{\partial^2 u}{\partial y^2} = f(x,y)$$

where $\begin{pmatrix} a(x,y) & b(x,y) \\ b(x,y) & c(x,y) \end{pmatrix} = \alpha I + dd^\top$ for $\alpha > 0$ and a unit vector d. We assume a mixture of Dirichlet and Neumann boundary conditions. This problem is discretized on an $n \times n$ regular mesh with a nested dissection ordering of the mesh points. The matrix A we consider is again the last Schur complement corresponding to the top level separator of the nested dissection.

In this example, we choose $n = 200$ and $\alpha = 10^{-p}$ with $p = 2, 4, 6, 8$. The block sizes of the leaf nodes in the bottom level of the HSS tree are chosen to be $m \approx 5$. The HSS block ranks are preset to $k = 2, 3, 4, 5$. For each preset rank k, we compare $\kappa_1(A_k)$, the condition number of A using XG's preconditioner, with

$\kappa_2(A_k)$, the condition number of A using the GHCF preconditioner. In Table 4.3, $\kappa_2(A_k)$ is consistently smaller than $\kappa_1(A_k)$: for instance, when $\alpha = 10^{-6}$ and $k = 2$, $\kappa_1(A_6) = 7.6 \times 10^4$ while $\kappa_2(A_6) = 1.76$. This suggests that the GHCF preconditioner performs better than XG's preconditioner. Table 4.4 shows that the preconditioned conjugate gradient (PCG) method in Matlab using the GHCF preconditioner requires fewer iterations than using PCG with XG's preconditioner.

TABLE 4.3
Example 2: The original condition number of A, and the condition numbers using XG's and GHCF preconditioners where the diagonal block size at the bottom level of the HSS tree is $m \approx 5$.

α	10^{-2}	10^{-4}	10^{-6}	10^{-8}
$\kappa(A)$	1.04e02	2.50e05	4.65e05	4.70e05
$\kappa_1(A_2)$	2.80	2.93e04	7.60e04	7.74e04
$\kappa_2(A_2)$	2.62	88.3	1.76	1.70e02
$\kappa_1(A_3)$	2.06	2.32e02	7.19e02	7.33e02
$\kappa_2(A_3)$	1.31	3.89	2.11	5.83
$\kappa_1(A_4)$	1.08	5.76	10.5	10.6
$\kappa_2(A_4)$	1.05	2.19	1.78	3.01
$\kappa_1(A_5)$	1.03	2.64	3.72	3.75
$\kappa_2(A_5)$	1.01	1.07	1.10	1.10

TABLE 4.4
Example 2: The numbers of PCG iterations with XG's and GHCF preconditioners.

α	10^{-2}	10^{-4}	10^{-6}	10^{-8}
$\kappa_1(A_2)$	11	20	21	20
$\kappa_2(A_2)$	11	16	17	16
$\kappa_1(A_3)$	10	15	16	16
$\kappa_2(A_3)$	9	12	12	12
$\kappa_1(A_4)$	7	10	11	11
$\kappa_2(A_4)$	6	8	9	8

Example 3. In this example, we consider the following matrix

$$A = \alpha I + B. \tag{4.1}$$

Here, I is the identity matrix, $B = \left(\sqrt{|x_i - x_j|}\right)_{n\times n}$ where $x_i = \cos\left((2i+1)\pi/2n\right)$ are the zeros of the nth Chebyshev polynomial, and $\alpha > 0$ is chosen so that A is positive definite. It is well known that B has low HSS rank [4, 33]. In the following results, we let $\alpha = \frac{n}{2}$. The block sizes at bottom level are $m = 25$, and we fix the precision parameter to be $\tau = \frac{1}{n}$.

We compare the total floating-point operations of constructing GHCF and XG's factorizations [36] in Table 4.5 using RRQR and the same parameter for compression as suggested in [36]. In our results, GHCF requires fewer flops than XG's algorithm and can save up to 50% or more operations for larger matrices. We also compare the Matlab run time in Table 4.6. The results in Table 4.6 are the CPU times of XG's algorithm with RRQR for compression over those of GHCF with RRQR, SVD, or

TABLE 4.5
Example 3: The complexity (flops) of GHCF and XG's algorithm.

n	512	1024	2048	4096
XG's	9.07e06	3.86e07	1.59e08	6.60e08
GHCF	7.45e06	2.92e07	1.14e08	4.69e08
$\frac{XG's-GHCF}{GHCF}$	21.8%	31.7%	39.1%	40.7%
n	8192	16384	32768	65536
XG's	2.75e09	1.12e10	4.53e10	1.87e11
GHCF	1.84e09	7.46e09	3.00e10	1.21e11
$\frac{XG's-GHCF}{GHCF}$	49.2%	50.1%	51.0%	54.7%

TABLE 4.6
Example 3: The speedup of GHCF over XG's algorithm with different compression methods.

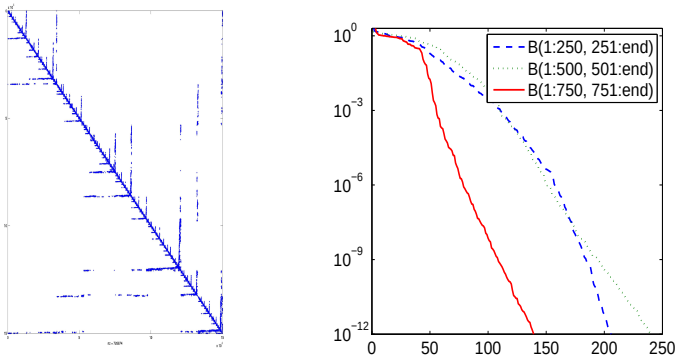
n	512	1024	2048	4096
$RRQR$	1.49	1.27	1.23	1.34
SVD	1.72	1.41	1.35	1.10
$SVDR$	1.56	1.40	1.85	2.51
n	8192	16384	32768	65536
$RRQR$	1.42	1.23	1.41	1.47
SVD	1.56	1.67	2.76	3.15
$SVDR$	3.34	3.84	4.97	6.44

SVDR for compression for different matrix sizes. In all cases, our new algorithm is faster, especially for large matrices.

Example 4. Lastly, we show that our algorithm can also be used to develop fast solvers for sparse matrices. Combining HSS structures with the multifrontal method can provide fast structured algorithms for sparse matrices; see, for example, [34]. Davis [10] has collected many sparse SPD matrices, most of which can be explored using HSS matrices. For instance, we take the SPD matrix **G2_circuit** from Tim Davis's homepage. The structure of A after permutating with the Matlab command SYMAMD is presented in Figure 4.1(a). This command computes a symmetric approximate minimum degree permutation of A which helps to reduce fill-in when factorizing A.

We consider the bottom dense triangular block C_b of the Cholesky factor of $A(P,P)$, where $P = \text{SYMAMD}(A)$, and C_b corresponds to the last separator of the nested dissection ordering. Figure 4.1(b) shows the rank-deficient property of some off-diagonal blocks of B. The vertical axis of Figure 4.1(b) gives the singular values of $B(1:100, 101:\text{end})$, $B(1:500, 501:\text{end})$ and $B(1:700, 701:\text{end})$ on a logarithmic scale. The condition number of B is 3.77×10^4.

Even when we choose a relatively small off-diagonal rank r, the factor computed from Algorithm 3 can still act as a good preconditioner for B. These results are illustrated in Table 4.7. In Figure 4.1(b), we see cases where the off-diagonal block of B is rank-deficient since the singular values decay rapidly but its numerical rank is still quite big. If we choose artificially small HSS ranks $k = 5, 8, 10$, our preconditioner can still make the matrix well-conditioned.



(a) $A(P, P)$

(b) The singular values of some off-diagonal block rows of B

FIG. 4.1. *The structure of this sparse matrix*

TABLE 4.7
The condition number of B preconditioned with the GHCF preconditioner

m	60	100	100	150	150	200
k	5	5	8	8	10	10
cond	68.3	55.2	46.4	32.7	33.9	21.1

5. Conclusion. We propose a generalized HSS Cholesky factorization method for symmetric positive definite matrices. This method is robust and Schur-monotonic since symmetric positive semidefinite matrices are automatically added to Schur complements during the factorization, preserving the positive definiteness property. Our factorization does not compute the Schur complement, and therefore requires fewer floating point operations than the method in [36]. We compare three low-rank matrix approximation methods for compression, i.e., RRQR, SVD, and SVD with random sampling (SVDR), and find that SVDR is fast and stable with high probability. Numerical results are given to show that our factorization can be used as an effective preconditioner or a direct solver with reasonable accuracy.

Acknowledgments. The authors are grateful to the anonymous referees for their invaluable suggestions. The research of Shengguo Li was supported by CSC (No. 2010611043) and in part by the Graduate School of NUDT, Fund of Innovation (B100201) and the innovation for postgraduate of Hunan province (grant No. CX2010B006). The research of Ming Gu was supported by NSF Award CCF-0830764. The research of Jianlin Xia was supported in part by NSF grants DMS-1115572 and CHE-0957024.

REFERENCES

[1] M. BEBENDORF, *Efficient inversion of Galerkin matrices of general second-order elliptic differential operators with nonsmooth coefficients*, Math. Comp., 74 (2005), pp. 1179–1199.
[2] D. BINI, L. GEMIGNANI, AND V. Y. PAN, *Fast and stable QR eigenvalue algorithms for generalized companion matrices and secular equations*, Numer. Math. 100 (2005), pp. 373–408.
[3] T. CHAN, *Rank revealing QR factorizations*, Linear Algebra Appl., 88/89 (1987), pp. 67–82.

[4] 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.

[5] S. Chandrasekaran, P. Dewilde, M. Gu, T. Pals, X. Sun, A. J. van der Veen, and D. White, *Fast stable solvers for sequentially semi-separable linear systems of equations and least squares problems*, Technical report, University of California, Berkeley, CA, 2003.

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

[7] 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.

[8] S. Chandrasekaran, M. Gu, and T. Pals, *Fast and stable algorithms for hierarchically semi-separable representations*, Technical report, Department of Mathematics, University of California, Berkeley, 2004.

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

[10] T. Davis, *University of Florida Sparse Matrix Collection*, NA Digest, Vol, 92, No, October 16, 1994.

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

[12] I. Gohberg, T. Kailath, and I. Koltracht, *Linear complexity algorithms for semiseparable matrices*, Integral Equations and Operator Theory, 8 (1985), pp. 780–804.

[13] G. H. Golub and C. F. Van Loan, *Matrix Computations*, Third ed., The Johns Hopkins University Press, Baltimore, MD, 1996.

[14] L. Grasedyck, R. Kriemann, and S. Le Borne, *Parallel black box domain decomposition based $\mathcal{H}$-LU preconditioning*, Technical Report 115, Max Planck Institute for Mathematics in the Sciences, Leipzig, 2005.

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

[16] 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.

[17] M. Gu, *Randomized sampling I: Low-rank matrix approximations*, SIAM J. Matrix Anal. Appl., submitted, 2011.

[18] M. Gu, *Randomized sampling II: Subspace iterations*, SIAM J. Matrix Anal. Appl., submitted, 2011.

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

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

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

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

[23] W. Hackbusch, B. Khoromskij, and S. Sauter, *On $\mathscr{H}^2$-matrices*, In Lecture on Applied Mathematics, Bungartz H, Hoppe RHW, Zenger C (eds). Springer: Berlin, (2000), pp. 9–29.

[24] N. Halko, P. G. Martinsson, and J. A. Tropp, *Finding structure with randomness probabilistic algorithms for constructing approximate matrix decompositions*, SIAM Review, 53 (2011), pp. 217–288.

[25] 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.

[26] 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.

[27] P. G. Martinsson and V. Rokhlin, *A fast direct solver for boundary integral equations in two dimensions*. J. Comput.Phys., 205 (2005), pp. 1–23.

[28] P. G. Martinsson, V. Rokhlin, and M. Tygert, *A randomized algorithm for the approximation of matrices*, Appl. Comput. Harmon. Anal., 30 (2011), pp. 47–68.

[29] V. Rokhlin, *Rapid solution of integral equations of scattering theory in two dimensions*, J. Comput.Phys., 86 (1990), pp. 414–439.

[30] H. P. Starr, Jr., *On the Numerical Solution of One-Dimensional Integral and Differential Equations*, Ph.D. thesis, Department of Computer Science, Yale University, May, 1992.
[31] R. Vandebril, M. Van Barel, G. Golub, and N. Mastronardi, *A bibliography on semiseparable matrices*, Calcolo, 42 (2005), pp. 249–270.
[32] R. Vandebril, M. Van Barel, and N. Mastronardi, *Matrix Computations and Semiseparable Matrices, Volume I: Linear Systems*, Johns Hopkins University Press, 2008.
[33] J. Xia, *On the complexity of some hierarchical structured matrices*, SIAM J. Matrix Anal. Appl., to appear, 2012.
[34] 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.
[35] J. Xia, S. Chandrasekaran, M. Gu, and X. S. Li, *Fast algorithm for hierarchically semiseparable matrices*, Numer. Linear Algebra Appl. 17 (2010), pp. 953–976.
[36] J. Xia and M. Gu, *Robust approximate Cholesky factorization of rank-structured symmetric positive definite matrices*, SIAM J. Matrix Anal. Appl., 31 (2010), pp. 2899–2920.
[37] J. Xia, Y. Xi, and M. Gu, *A superfast structured solver for Toeplitz linear systems via randomized sampling*, SIAM J. Matrix Anal. Appl., submitted, 2011.