AN ALGEBRAIC MULTILEVEL PRECONDITIONER WITH LOW-RANK CORRECTIONS FOR GENERAL SPARSE SYMMETRIC MATRICES*

YUANZHE XI†, RUIPENG LI†, AND YOUSEF SAAD†

Abstract. This paper describes a multilevel preconditioning technique for solving linear systems with general sparse symmetric coefficient matrices. This "multilevel Schur low rank" (MSLR) preconditioner first builds a tree structure $\mathcal{T}$ based on a hierarchical decomposition of the matrix and then computes an approximate inverse of the original matrix level by level. Unlike classical direct solvers, the construction of the MSLR preconditioner follows a top-down traversal of $\mathcal{T}$ and exploits a low-rank property that is satisfied by the difference between the inverses of the local Schur complements and specific blocks of the original matrix. A few steps of the generalized Lanczos tridiagonalization procedure are applied to capture most of this difference. Numerical results are reported to illustrate the efficiency and robustness of the MSLR preconditioner with both two- and three-dimensional discretized PDE problems as well as some publicly available test problems.

Key words. Low-rank approximation, Schur complements, multilevel preconditioner, domain decomposition, incomplete factorization, Krylov subspace methods, nested dissection ordering.

AMS subject classifications. 65F08, 65N22, 65N55, 65Y05, 65Y20

1. Introduction.

In this paper, we consider iterative methods for solving large symmetric sparse linear systems

$$Ax = b, \tag{1.1}$$

where $A \in \mathbb{R}^{n\times n}$ and $b \in \mathbb{R}^n$. Krylov subspace methods belong to a class of general-purpose techniques that are usually combined with a preconditioner to solve the above system. Preconditioning consists of modifying the original system into, for example, the left-preconditioned system $M^{-1}Ax = M^{-1}b$. The preconditioner M is an approximation to A such that solving linear systems with it is relatively inexpensive. Often, the preconditioning matrix M is built from an Incomplete LU (ILU) factorization extracted from a form of approximate Gaussian elimination process in which "fill-ins", i.e., nonzero entries appearing in zero locations in the original matrix, are removed. ILU preconditioners are fairly easy to implement and robust for a large class of problems.

The success of these ILU methods for solving certain systems arising from Partial Differential Equations (PDEs), has been widely illustrated in the literature. At the same time it has also become clear in recent years that they fall short of providing robust and scalable solvers for handling harder problems. The situation has been accentuated by the increased complexity of linear systems encountered on the one hand and the parallelization requirements on the other. The preconditioners developed so far in the literature, often face one, or both, of the following issues. First, they fail for highly indefinite systems because in these situations they will either encounter some small/zero pivots during the ILU factorization process or produce unstable triangular factors [4, 28]. Second, their performance in the new breed of highly parallel architectures, such as computers equipped with GPUs [20] or Intel Xeon Phi processors,

*This work was supported by NSF under grant NSF/DMS-1216366 and by the Minnesota Supercomputing Institute

†Address: Department of Computer Science & Engineering, University of Minnesota, Twin Cities. {yxi,rli,saad} @cs.umn.edu

is unacceptably poor due to the sequential nature of the construction and the application of the preconditioner. Approximate inverse preconditioners were developed in the 1990s as an alternative to ILUs [3, 5, 6] to address these issues but their success has been limited because they rely on the inverse of A being well approximated by a sparse matrix. In the end, when considering construction cost, storage requirement, and number of iterations needed for convergence by these techniques, users found them non-competitive for solving systems with general matrices.

Both of the issues just mentioned can be addressed by low-rank approximation techniques. To begin with, several rank structured representations of matrices have been developed for the fast solutions of structured systems, PDEs, integral equations, eigenvalue problems and for building preconditioners [7, 17, 18, 23, 34, 35, 39, 40]. They include the work on $\mathcal{H}$-matrices [11, 12] and on the variant of hierarchically semiseperable (HSS) matrices [34, 38]. These matrix representations partition the given matrix into appropriate blocks and approximate certain off-diagonal blocks with low-rank matrices. They have also been integrated into sparse matrix techniques to speed up the intermediate dense matrix operations and provided a new class of sparse direct solvers [14, 37] and preconditioners [9, 36]. Although these methods have lower asymptotic complexity bounds than standard methods, the prefactors in these bounds are large, and the methods generally require much more sophisticated implementations. They often also require some information on the underlying physical structure [9].

More recently, a few articles focused on a new class of approximate inverse preconditioners. Within the domain decomposition framework, these preconditioners seek data sparsity rather than the standard non-zero sparsity in the inverse matrices, and approximate these inverses with various low-rank correction techniques. The starting point is the Multilevel Low-Rank (MLR) preconditioner proposed in [19]. The Domain Decomposition based Low-Rank (DD-LR) preconditioner [21] is a variant geared towards distributed sparse matrices on massively distributed memory computers. The Schur complement based Low-Rank (SLR) preconditioner [22] further extends this framework to provide preconditioners designed for classical Schur complement methods. In general, these low-rank correction based approximate inverse preconditioners reach a good balance between robustness and efficiency and thus offer promising alternatives to the ILU-type preconditioners on modern high performance computers.

1.1. Contributions. In this paper, we generalize the SLR preconditioner introduced in [22] and propose a multilevel-level Schur complement based Low-Rank (MSLR) preconditioner. Its main advantages are outlined below.

(1) *HID ordering for general sparse matrices.* In contrast to the SLR preconditioner where the adjacency graph of the matrix is only partitioned into subdomains and interface points, the MSLR preconditioner applies the Hierarchical Interface Decomposition (HID) ordering [15] to exploit the hierarchy among the interface points such that a block independent set structure [30] appears at each level of the HID tree $\mathcal{T}$. The MSLR preconditioner construction algorithm can then fully take advantages of this structure from two aspects. First, all the diagonal blocks in the reordered matrix can be factored simultaneously. Second, the rank used in the low-rank corrections at each level can be estimated for some model problems. The HID ordering can be obtained from graph partitioning tools and is applicable to both 2D and 3D discretizated PDEs as well as general sparse matrices.

(2) *Enhanced efficiency and robustness to indefiniteness.* After the HID ordering is

obtained and applied, the MSLR preconditioner follows a top-down traversal of $\mathcal{T}$ and constructs an approximate inverse by exploiting a low-rank property in the Schur complement inverse at each level of $\mathcal{T}$. This construction procedure traverses the tree $\mathcal{T}$ in an order that is the reverse of that used in direct methods and significantly reduces fill-ins, especially for large 3D problems. We prove that, for some 2D/3D model problems, the storage for the MSLR preconditioner is nearly $O(n)$. In addition, the MSLR preconditioner gains in efficiency in the application phase since irregular computations, such as the backward and forward substitutions, are limited to the diagonal blocks where these operations can be performed independently for the blocks at the same level of $\mathcal{T}$. The remaining operations only involve the sparse/dense matrix-vector products. Extensive numerical experiments further show that the MSLR preconditioner is not as sensitive to indefinite systems as the classical ILU-type preconditioners.

(3) *Easy update.* The quality of the MSLR preconditioner is highly dependent on the rank used in the low-rank corrections. These low-rank corrections are computed by the generalized Lanczos algorithm. When the current preconditioner does not achieve good performance, it can be quickly updated by simply computing more eigenvectors and adding them into the constructed low-rank correction terms.

1.2. Outline. The remaining sections are organized as follows. Section 2 reviews the SLR preconditioner proposed in [22]. A hierarchical interface decomposition ordering is introduced in Section 3. We study the low-rank property associated with a sequence of local Schur complement inverses in Section 4 and propose the MSLR preconditioner in Section 5. The complexity of the MSLR preconditioner is analyzed in Section 6. Numerical examples with some 2D and 3D PDEs as well as general sparse matrices are reported in section 7 and a conclusion is drawn in Section 8. In the presentation, we use the following notation:

- $[W, \Sigma] = \text{eigs}(A, k)$ denotes the application of the Lanczos algorithm to compute k largest eigenpairs of A, where the columns of W contain the desired eigenvectors and the diagonal entries of Σ are the k largest eigenvalues of A;
- $\text{diag}(\sigma_i)$ represents a diagonal matrix with σ_i along its diagonal;
- $|A|$ denotes the dimension of A for a square matrix.

2. Review of the SLR preconditioner. The MSLR preconditioner and the SLR preconditioner both rely on a low-rank property associated with the Schur complement inverse. The goal of this section is to briefly review this property and give some details that were not provided in [19, 22]. Assume A is an SPD matrix and let it be partitioned in 2×2 block form as

$$A = \begin{pmatrix} B & E \\ E^T & C \end{pmatrix}, \tag{2.1}$$

where $C \in \mathbb{R}^{s \times s}$. We define the Schur complement matrix $S = C - E^T B^{-1} E$. Since A is SPD, C is also SPD and admits a Cholesky factorization

$$C = LL^T.$$

Define the matrix

$$G = L^{-1} E^T B^{-1} E L^{-T} = L^{-1} (C - S) L^{-T}, \tag{2.2}$$

and call $\sigma_i, i = 1, \cdots, s$, its eigenvalues labeled decreasingly. It can be easily shown that [22]

$$0 \leq \sigma_s \leq \sigma_{s-1} \leq \cdots \leq \sigma_1 < 1.$$

We are interested in the separation of the eigenvalues of $S^{-1} - C^{-1}$ or those of $X \equiv L^T \left(S^{-1} - C^{-1}\right) L$. The second is easier to analyze. The following result explains the observed 'decay' property of the eigenvalues of X.

PROPOSITION 2.1. *The eigenvalues $\theta_s \leq \theta_{s-1} \leq \cdots \leq \theta_1$ of X are related to those of $G = L^{-1}(C - S)L^{-T}$ by*

$$\theta_i = \frac{\sigma_i}{1 - \sigma_i}, \quad i = 1, \cdots, s. \tag{2.3}$$

In particular the gaps $\theta_i - \theta_{i+1}$ between two consecutive eigenvalues of X are given by

$$\theta_i - \theta_{i+1} = \frac{\sigma_i - \sigma_{i+1}}{(1 - \sigma_{i+1})(1 - \sigma_i)}, \quad i = 1, \cdots, s-1. \tag{2.4}$$

These gaps are bigger than the gaps $\sigma_i - \sigma_{i+1}$ associated with the matrix G, and they are much bigger for those eigenvalues σ_i close to one, i.e., for the largest ones.

Proof. We have

$$S = L(I - L^{-1}E^T B^{-1} E L^{-T})L^T = L(I - G)L^T.$$

From the above expression and the Cholesky factorization of C we have:

$$S^{-1} - C^{-1} = L^{-T}\left[(I - G)^{-1} - I\right] L^{-1} = L^{-T}\left[G(I - G)^{-1}\right] L^{-1}.$$

Therefore, $X = L^T(S^{-1} - C^{-1})L = G(I - G)^{-1}$ and so the eigenvalues of X and those of G are related by (2.3). The gap formula (2.4) results from a straightforward calculation. ∎

Proposition 2.1 shows the relation between the eigenvalues of $L^{-1}\left(C - S\right) L^{-T}$ and $L^T \left(S^{-1} - C^{-1}\right) L$. It shows that as long as the largest σ_i's do not cluster around 1, the largest θ_i's will be very well separated in general.

We were initially interested in the difference between $S^{-1} - C^{-1}$ which is equal to $L^{-T} X L^{-1}$. First we observe that X has the same eigenvalues as the matrix $S^{-1}C - I$ which is equal to $(S^{-1} - C^{-1})C$. In many applications the matrix A originates from a discretized PDE and in these applications the matrix C is strongly diagonally dominant.

Based on the eigenvalue inequality (Lidskiis product inequalities and their further developments [24]), we know that

$$\theta_i \lambda_s(C^{-1}) \leq \lambda_i(S^{-1} - C^{-1}) \leq \theta_i \lambda_1(C^{-1}).$$

Assume that

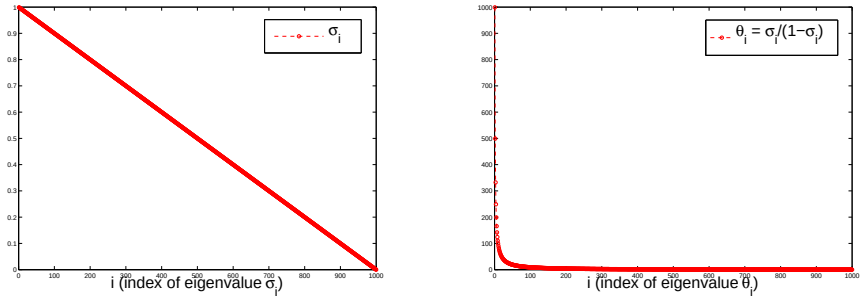
$$\theta_{k+1} \leq \tau \theta_1 \quad \text{and} \quad \lambda_1(C^{-1}) = c\lambda_s(C^{-1}) \tag{2.5}$$

for some constants τ and c, then the following estimation holds

$$\lambda_{k+1}(S^{-1} - C^{-1}) \leq \tau\theta_1\lambda_1(C^{-1}) = c\tau\theta_1\lambda_s(C^{-1}) \leq c\tau\lambda_1(S^{-1} - C^{-1}).$$

Therefore, if the gap between θ_1 and θ_{k+1} is relatively big which means τ in (2.5) is small, then the gap between $\lambda_1(S^{-1} - C^{-1})$ and $\lambda_{k+1}(S^{-1} - C^{-1})$ will be preserved except that it will be damped by the condition number of C to a certain degree.

It is not easy to measure the decay rate of $S^{-1} - C^{-1}$ for a general matrix. It can be explicitly analyzed for a 2-D Laplacian on a regular grid when 2 domains are used [22].



(i) σ_i decays linearly between $[0, 0.999]$. (ii) $\theta_i = \frac{\sigma_i}{1-\sigma_i}$ decays between $[0, 1000]$.

FIG. 2.1. *Example: Eigenvalue decay rate comparison between* σ_i *and* $\sigma_i/(1-\sigma_i)$. *Here we assume* σ_i*s decay linearly from* 0.999 *to* 0 *for* 1000 *points.*

The decay property was fully illustrated in [22] for 2-D Laplacian matrices discretized on a regular grid. Here we illustrate Proposition 2.1, by considering a simpler example where 1000 eigenvalues values σ_i are assumed to decay linearly between (0, 0.999), see Figure 2.1 (i). The values of $\theta_i = \sigma_i/(1-\sigma_i)$ are also plotted in Figure 2.1 (ii) for a comparison.

We see that the values of θ_i decay much faster than those of σ_i in this special example. When θ_{11} equals only 9% of θ_1, σ_{11} is still 99% as large as σ_1. This means S^{-1} can be well approximated by C^{-1} plus a rank-10 matrix in this case. In contrast, $C - S$ cannot be approximated by a low-rank matrix. The usefulness of this property for developing preconditioners of general matrices was also demonstrated through numerical examples in [22].

The SLR preconditioner proposed in [22] incorporates the result of Proposition 2.1 into the domain decomposition framework for general sparse matrices. The idea is to first partition the adjacency graph of the input matrix into p disjoint subdomains and interface points and label the unknowns associated with each subdomain before the unknowns associated with interface points such that B in (2.1) corresponds to the points of subdomains and is in a block diagonal form. The matrices B and C are factored using a form of incomplete Cholesky factorization, so we have $B \approx L_B L_B^T$, and $C \approx L_C L_C^T$. The SLR preconditioner M then has the following form

$$M = \begin{pmatrix} L_B & \\ E^T L_B^{-T} & I \end{pmatrix} \begin{pmatrix} I & \\ & \tilde{S} \end{pmatrix} \begin{pmatrix} L_B^T & L_B^{-1}E \\ & I \end{pmatrix},$$

where $\tilde{S}^{-1} = L_C^{-T}\left(I + WTW^T\right)L_C^{-1}$, and $L_C^{-T}WTW^TL_C^{-1}$ is a low-rank approximation for $S^{-1} - C^{-1}$ obtained from approximating $L_C^{-1}E^TB^{-1}EL_C^{-T}$ using the Lanczos algorithm.

The numerical tests reported in [22] indicated that the SLR preconditioner is quite robust, making it an attractive alternative to ILU-type preconditioners for certain types of problems. However, its efficiency deteriorates for large 3D problems. This is due to the fact that the number of the interface points can be relatively big for those problems and a direct factorization of the matrix C in (2.1) becomes costly. This observation provides a motivation for exploring a multilevel scheme to improve performance.

3. Hierarchical interface decomposition ordering. An interesting class of domain decomposition methods, that exploits a hierarchy of "interfaces", is based on so-called *wirebasket* orderings [32, 33]. These techniques take advantages of cross-points in the partitioned mesh to derive preconditioners with good convergence properties. In [15] a similar idea is introduced for general sparse matrices not necessarily originating from partial differential equations. This paper defines a so-called "*Hierarchical Interface Decomposition (HID)*" ordering for a general graph through "*connectors*". In an HID ordering with L levels, connectors of level l are a set of subsets of vertices that are disjoint with other connectors on the same level and separated in the original graph by connectors of higher levels. When labeled by levels the resulting matrix will have a block-diagonal structure for each level and very efficient parallel preconditioners based on parallel Gaussian elimination can be developed. However, we will not seek to develop ILU-type preconditioners, but rather to incorporate low-rank approximations in this context as is explained in the following sections.

An HID ordering can be obtained in a number of ways, see [15] for an example from a standard graph partitioning. They can also be obtained from the standard Nested Dissection (ND) algorithm [10] . Let $\mathcal{G} = (V, E)$ be the adjacency graph associated with a symmetric sparse matrix A. The main idea of the ND ordering is to recursively bi-partition the graph using *vertex separators*. Here, a vertex separator is a small set of nodes in V whose removal separates the graph into two disjoint subgraphs. Suppose we have L levels. The first bisection produces two subsets and a separator. At this level ($L = 3$) we will have one connector and it is just the separator. This set is the line of vertices labeled 15 in Figure 3.1 (i). Descending to the next level ($L - 1 = 2$) we now have two connectors consisting of the two separators of each of the resulting subsets. These are labeled 13 and 14 in Figure 3.1 (i). The separators produced at step i of ND ordering ($i = 1, \ldots, L - 1$) are the connectors of level $L - i + 1$. The zero-th level consists of the subsets of points interior to each subdomain. The level information of these connectors can be represented by an HID tree $\mathcal{T}$. See Figure 3.1 (ii) for an example. The matrix is ordered by level, starting with nodes of level 0 and ending with nodes of level L. Figure 3.1 (iii) shows the non-zero pattern for the mesh in Figure 3.1 (i) when this ordering is used.

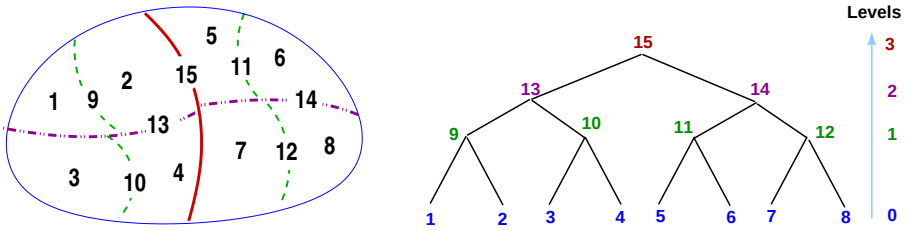
The reordered matrix with the HID ordering has the following multilevel recursive form:

$$A_l = \begin{pmatrix} B_l & E_l \\ E_l^T & C_l \end{pmatrix} \quad \text{and} \quad C_l = A_{l+1} \quad \text{for} \quad l = 0 : L - 1, \tag{3.1}$$

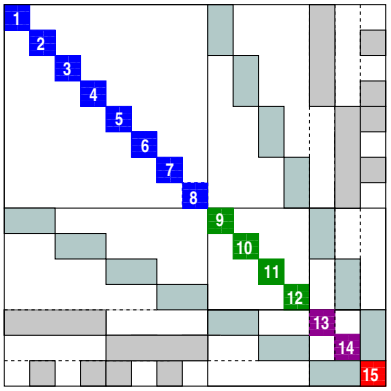
where A_0 is the reordered matrix of A from the HID ordering and A_L represents the submatrix associated with the top-level connector. Each leading block B_l in A_l has the desired block-diagonal structure resulting from the block independent set [30] nature of this ordering. In the example of Figure 3.1 (iii), B_0, B_1 and B_2 correspond to the matrices containing the blocks labeled 1-8, 9-12 and 13-14, respectively.

The HID ordering forms the block independent sets in a preprocessing stage. In contrast, the methods in [27, 30, 31] determine these block independent sets dynamically since each A_{l+1}, which is an approximate Schur complement matrix at level l, is only available after the factorization of A_l. Once the block independent set at each level l is determined, classical multilevel algorithms [27, 30, 31] explore different strategies to approximate the following factorization of A_l:

$$A_l = \begin{pmatrix} I & \\ E_l^T B_l^{-1} & I \end{pmatrix} \begin{pmatrix} B_l & \\ & S_l \end{pmatrix} \begin{pmatrix} I & B_l^{-1} E_l \\ & I \end{pmatrix}, \tag{3.2}$$



(i) 3-level partition of a 2-D domain. (ii) An HID tree with connector level information.



(iii) Non-zero pattern of the reordered matrix.

FIG. 3.1. *An illustration of the hierarchical domain decomposition ordering for a 2D problem.*

where $S_l = C_l - E_l^T B_l^{-1} E_l$ is the Schur complement. The next section discusses the low rank property of $S_l^{-1} - C_l^{-1}$ for each l and derives an efficient multilevel preconditioner based on this property.

4. Low-rank property of $\mathbf{S_l^{-1} - C_l^{-1}}$. In this section, we examine the rank of the matrix $S_l^{-1} - C_l^{-1}$ obtained from the HID ordering for some model problems. We start our analysis from the exact rank of $S_l^{-1} - C_l^{-1}$. First we need to define the *cross points* in the HID ordering since these points are closely related to the rank of $S_l^{-1} - C_l^{-1}$. Similar definitions can also be found in [15, 33].

DEFINITION 4.1. Cross points *at level l are subsets of connectors of level l that intersect with connectors of higher levels.*

In Figure 3.1 (i) for example, the cross points of level 2 are those points at the intersection of the lines labeled 13 and 14 with the line labeled 15. Similarly, cross points of level 1 are those points at the intersection of the lines labeled $9, 10, 11, 12$ with the lines labeled $13, 14$. The relation between the cross points and the rank of $S_l^{-1} - C_l^{-1}$ is shown in the following lemma.

LEMMA 4.2. *The exact rank of $S_l^{-1} - C_l^{-1}$ is bounded by the number of the cross points at level l.*

Proof. Since E_l represents the couplings between the connectors at level l and upper levels k for $k > l$, the rank of E_l should equal the number of cross points at level l which is generally smaller than the dimension of C_l. This means E_l admits a low-rank representation

$$E_l = U_l V_l^T,$$

where the column size of V_l equals the number of cross points at level l. Thus,

$$S_l = C_l - V_l \underbrace{U_l^T B_l^{-1} U_l}_{G_l} V_l^T.$$

Applying the Sherman-Morrison-Woodbury (SMW) formula to S_l,

$$S_l^{-1} = C_l^{-1} + C_l^{-1} V_l (G_l^{-1} - V_l^T C_l^{-1} V_l)^{-1} V_l^T C_l^{-1},$$

shows clearly that the rank of $S_l^{-1} - C_l^{-1}$ is bounded by the rank of V_l, which is equal to the rank of E_l. □

Lemma 4.2 shows that we can bound the rank of $S_l^{-1} - C_l^{-1}$ based on the rank of E_l. For the 2D and 3D PDE problems discretized on the regular grids ($N \times N$ for 2D or $N \times N \times N$ for 3D), the rank of E_l can be analytically derived.

LEMMA 4.3. *Suppose an L level HID ordering is applied to a 2D/3D regular grid. Then the rank of $S_l^{-1} - C_l^{-1}$ has order $O(2^{L-l})$ for 2D problems and $O(2^{L-l}N)$ for 3D problems when $l > 0$.*

Proof. Based on Lemma 4.2, we know that we only need to count the number of the cross points at each level. For the 2D case, each connector at level l connects to upper level connectors through at most 2 points. There are 2^{L-l} connectors in all at level l, so the rank of $S_l^{-1} - C_l^{-1}$ is bounded by $O(2^{L-l})$. The ranks in the 3D case are much larger since the connectors are all 2D planes. Each connector at level l connects to upper level ones through at most four lines. The length of those lines is bounded by N. Thus, the rank of $S_l^{-1} - C_l^{-1}$ has order $O(2^{L-l}N)$. □

Lemma 4.3 shows that the rank of $S_l^{-1} - C_l^{-1}$ can be considered small when l is near L. When $l = 0$, the rank of $S_l^{-1} - C_l^{-1}$ no longer satisfies the rank bound estimation in Lemma 4.2 and is proportional to the number of boundary points of all subdomains on level 0, which is $O(2^{\frac{L}{2}}N)$ for the 2D case and $O(2^{\frac{L}{3}}N^2)$ for the 3D case. See Figures 4.1, 4.2, 4.3 and 4.4 for a Laplacian example. The eigenvalues are computed by the matlab built in function EIG and the resulting ranks are slightly different from the estimation in Lemma 4.3. The reason is that we use the graph partitioning tool METIS [16] to achieve the HID ordering so that the connectors may have irregular connectivity and do not strictly follow the assumptions in the theoretical analysis.

Figures 4.2 and 4.4 numerically confirm the claim of Lemma 4.3 and show that the exact rank of $S_l^{-1} - C_l^{-1}$ almost doubles as we go down one level on $\mathcal{T}$ until level 1. But a more interesting property is in regard to the numerical rank. Based on Proposition 2.1, we can expect that the numerical rank r_l of $S_l^{-1} - C_l^{-1}$ should be much smaller. We report these numerical ranks with respect to different tolerances τ in Tables 4.1 and 4.2 for the Laplacian examples in Figures 4.1 and 4.3. Here the numerical rank $r_l = k$ with respect to τ, or relative to τ, just means that the kth singular value is larger than $\tau \times$ the 1st singular value while the $(k+1)$st singular value is less than or equal to $\tau \times$ the 1st singular value. (In these Laplacian examples, the eigenvalues and the singular values are the same.)

From Table 4.1, we find that the numerical ranks in the 2D Laplacian example are all less than 7 for $\tau = 2 \times 10^{-1}$. From Table 4.2 we also observe that the ranks for the 3D problem are slightly larger than the 2D problem. Their ranks are all less than 38 when $\tau = 2 \times 10^{-1}$.

From the above analysis, we see that $S_l^{-1} - C_l^{-1}$ can be well approximated with a low-rank matrix. Next, we will derive an efficient method to compute such an approximation without having to form S_l^{-1} explicitly. This low-rank approximation method

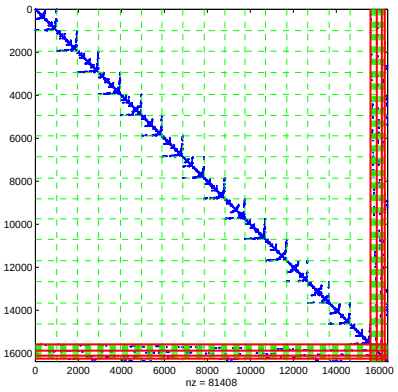


FIG. 4.1. *Nonzero pattern of a reordered discretized negative 2D Laplacian with a 4–level HID ordering. The original matrix is discretized on a* 128×128 *regular grid with 5-point stencil.*

TABLE 4.1
Numerical rank r_l *of* $S_l^{-1} - C_l^{-1}$ *for the 2D Laplacian example in Figure 4.1 for varying tolerances* τ.

Matrix	$S_0^{-1} - C_0^{-1}$	$S_1^{-1} - C_1^{-1}$	$S_2^{-1} - C_2^{-1}$	$S_3^{-1} - C_3^{-1}$
dim	797	504	269	135
τ	r_0	r_1	r_2	r_3
2×10^{-1}	6	7	2	1
10^{-1}	12	7	2	1
10^{-2}	126	7	2	1
10^{-3}	371	7	2	1

will be used extensively in the construction of the multilevel low-rank preconditioner introduced in the next section. Based on the proof in Proposition 2.1, we know that

$$S_l^{-1} - C_l^{-1} = L_l^{-T} U_l \left(\Sigma_l (I - \Sigma_l)^{-1}\right) U_l^T L_l^{-1}, \tag{4.1}$$

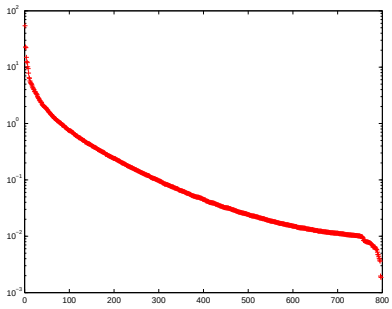
where L_l is the Cholesky factor of C_l and $U_l \Sigma_l U_l^T$ is the eigen-decomposition of $L_l^{-1} E_l^T B^{-1} E_l L_l^{-T}$. Notice that if we simply follow the approximation method used in the SLR preconditioner, we would have to compute the incomplete Cholesky factorization of each C_l. Obviously, this would prove highly inefficient because it is not possible to reuse the factorization information among these factors even though C_l is a submatrix of C_{l-k} for $k < l$.

It is easy to see that $C_l^{-1} E_l^T B_l^{-1} E_l$ has the same eigenvalues as $L_l^{-1} E_l^T B^{-1} E_l L_l^{-T}$ and the columns of $L_l^{-T} U_l$ in (4.1) are the eigenvectors of $C_l^{-1} E_l^T B_l^{-1} E_l$ due to the following relation

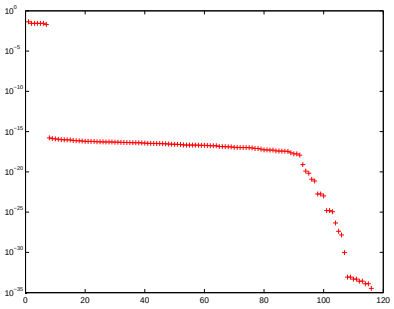
$$L_l^{-1} E_l^T B^{-1} E_l L_l^{-T} U_l = U_l \Sigma_l \Longleftrightarrow C_l^{-1} E_l^T B_l^{-1} E_l L_l^{-T} U_l = L_l^{-T} U_l \Sigma_l.$$

Thus, the computation of a low-rank approximation to (4.1) can be obtained through the following generalized eigenvalue problem

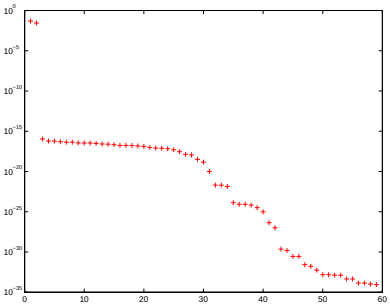
$$E_l^T B_l^{-1} E_l x = \lambda C_l x.$$



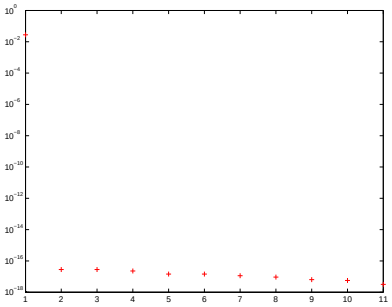
(i) Eigenvalues of $S_0^{-1} - C_0^{-1}$.



(ii) Eigenvalues of $S_1^{-1} - C_1^{-1}$.



(iii) Eigenvalues of $S_2^{-1} - C_2^{-1}$.



(ii) Eigenvalues of $S_3^{-1} - C_3^{-1}$.

FIG. 4.2. *The eigenvalues of $S_l^{-1} - C_l^{-1}$ at different levels l for the 2D example in Figure 4.1.*

TABLE 4.2
Numerical rank r_l of $S_l^{-1} - C_l^{-1}$ for the 3D Laplacian example in Figure 4.3 relative to different tolerance τ.

Matrix	$S_0^{-1} - C_0^{-1}$	$S_1^{-1} - C_1^{-1}$	$S_2^{-1} - C_2^{-1}$	$S_3^{-1} - C_3^{-1}$
dim	1790	1137	780	400
τ	r_0	r_1	r_2	r_3
2×10^{-1}	10	38	17	13
10^{-1}	28	63	32	21
10^{-2}	275	103	63	30
10^{-3}	1268	103	63	30

We compute the k largest eigenpairs of the above problem and denote this computation as

$$[W_l, \Sigma_l] = \text{eigs}(C_l^{-1} E_l^T B_l^{-1} E_l, k). \tag{4.2}$$

Then, a rank k approximation to $S_l^{-1} - C_l^{-1}$ can be chosen as

$$S_l^{-1} - C_l^{-1} \approx W_l H_l W_l^T, \tag{4.3}$$

where the diagonal matrix H_l is computed by

$$H_l = \Sigma_l (I - \Sigma_l)^{-1}.$$

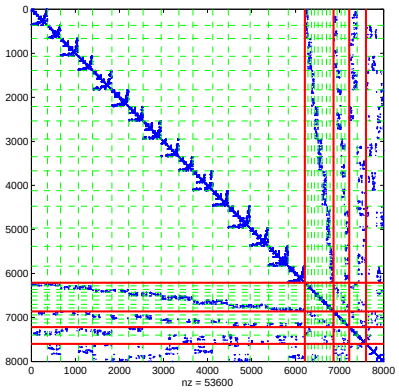


FIG. 4.3. *Nonzero pattern of a reordered discretized negative 3D Laplacian with a* 4*–level HID ordering. The original matrix is discretized on a* $20 \times 20 \times 20$ *regular grid with* 7*-point stencil.*

This low-rank approximation method has the advantage that it only requires a system solution with the matrix C_l in the eigenvalue computation. In the next section, we will introduce a recursive formula to solve systems with C_l based on the approximation of C_k for $k > l$.

5. Multilevel low-rank correction preconditioner construction. In this section, we will show how to utilize the low-rank property discussed in Section 4 to develop an efficient multilevel Schur complement based Low-Rank (MSLR) preconditioner for general symmetric sparse matrices. Even though the above analysis is derived from symmetric positive definite (SPD) systems, the proposed method can also be extended to the indefinite systems where only B_0 is indefinite and the remaining B_l's for $l > 0$ are all SPD. This situation often occurs in discretized PDE problems [22].

5.1. Basic idea of the 3 **level scheme.** The basic idea of the MSLR preconditioner can be illustrated with a 3 level algorithm. The major steps are as follows:

1. Apply a 3 level HID ordering to reorder the matrix A such that the top level connector is at level 3.
2. At level 0, factorize A_0 as follows

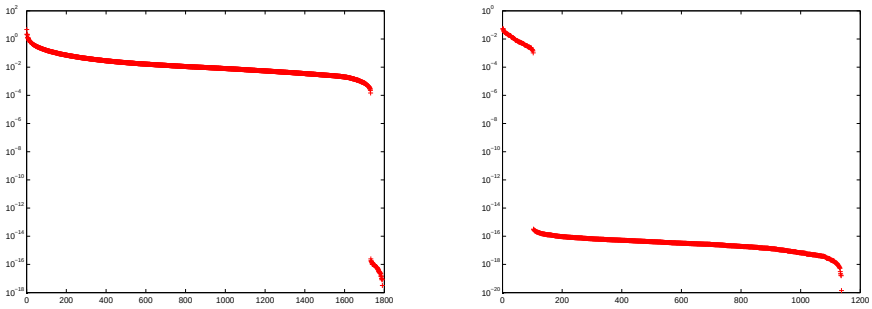
$$A_0 = \begin{pmatrix} I & \\ E_0^T B_0^{-1} & I \end{pmatrix} \begin{pmatrix} B_0 & \\ & S_0 \end{pmatrix} \begin{pmatrix} I & B_0^{-1} E_0 \\ & I \end{pmatrix}.$$

Thus,

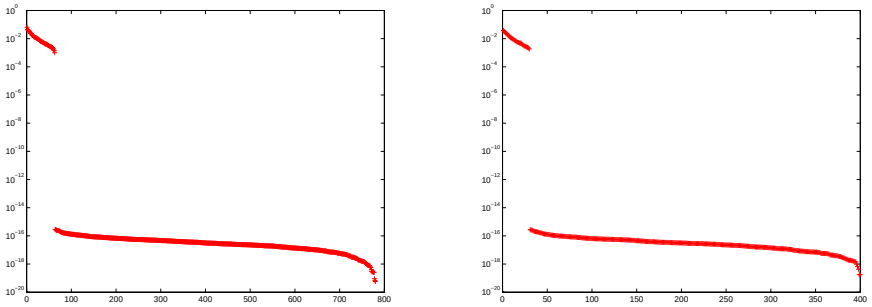
$$A_0^{-1} = \begin{pmatrix} I & -B_0^{-1} E_0 \\ & I \end{pmatrix} \begin{pmatrix} B_0^{-1} & \\ & S_0^{-1} \end{pmatrix} \begin{pmatrix} I & \\ -E_0^T B_0^{-1} & I \end{pmatrix}.$$

This shows that the original system A can be easily solved if S_0^{-1} is available. Based on the discussion in Section 4, we know that S_0^{-1} can be approximated by C_0^{-1} plus a low-rank correction term as follows

$$S_0^{-1} \approx C_0^{-1} + W_0 H_0 W_0^T.$$



(i) Eigenvalues of $S_0^{-1} - C_0^{-1}$. (ii) Eigenvalues of $S_1^{-1} - C_1^{-1}$.



(iii) Eigenvalues of $S_2^{-1} - C_2^{-1}$. (ii) Eigenvalues of $S_3^{-1} - C_3^{-1}$.

FIG. 4.4. *The eigenvalues of $S_l^{-1} - C_l^{-1}$ at different levels l for the 3D example in Figure 4.3.*

Notice that this low-rank term also involves the information from C_0^{-1}. Thus the problem reduces to finding C_0^{-1}. A direct factorization of C_0 is too costly if its dimension is large, so we go upward to the next level (level 1) and try to find an approximation to C_0^{-1}.

3. At level 1, we have

$$C_0^{-1} \equiv A_1^{-1} = \begin{pmatrix} I & -B_1^{-1}E_1 \\ & I \end{pmatrix} \begin{pmatrix} B_1^{-1} & \\ & S_1^{-1} \end{pmatrix} \begin{pmatrix} I & \\ -E_1^T B_1^{-1} & I \end{pmatrix}.$$

Following the same reasoning as in level 0, the matrix of S_1^{-1} can be approximated by C_1^{-1} plus a correction

$$S_1^{-1} \approx C_1^{-1} + W_1 H_1 W_1^T.$$

We then move to level 2 to compute C_1^{-1}.

4. At level 2, we have

$$C_1^{-1} \equiv A_2^{-1} = \begin{pmatrix} I & -B_2^{-1}E_2 \\ & I \end{pmatrix} \begin{pmatrix} B_2^{-1} & \\ & S_2^{-1} \end{pmatrix} \begin{pmatrix} I & \\ -E_2^T B_2^{-1} & I \end{pmatrix}.$$

and similarly to previous levels

$$S_2^{-1} \approx C_2^{-1} + W_2 H_2 W_2^T.$$

We assume the size of C_2 is small enough to admit a direct incomplete Cholesky factorization

$$C_2 \approx L_2 L_2^T.$$

5. Therefore, a 3-level scheme to compute an approximation to A^{-1} follows the following dependencies

$$A^{-1} \to A_0^{-1} \to S_0^{-1} \to A_1^{-1} \to S_1^{-1} \to A_2^{-1} \to S_2^{-1} \to C_2^{-1} \to L_2.$$

See Figure 5.1 for a pictorial illustration of this dependence.

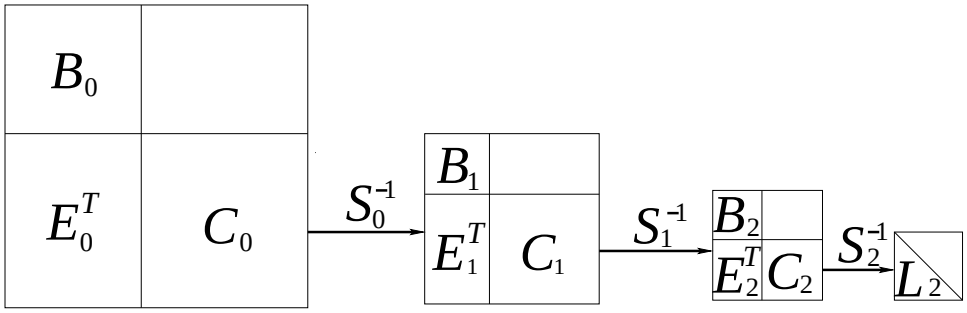


FIG. 5.1. *An illustration of the dependencies in the* 3 *level algorithm.*

ALGORITHM 1
Multilevel Schur complement based Low-Rank (MSLR) preconditioner construction

---

procedure `MSLR`
  Apply an L-level HID ordering to A and denote the reordered matrix as A_0.
  Factor C_{L-1} by an incomplete Cholesky factorization

$$C_{L-1} \approx M_{L-1} M_{L-1}^T.$$

  for level l from $L-1$ to 0 **do**
    Factor B_l by an incomplete Cholesky/LDL factorization

$$B_l \approx L_l D_l L_l^T.$$

    Compute the k largest eigenpairs ▷ *Call Algorithm 2 to apply* C_l^{-1} *except when* $l = L-1$

$$[W_l, \Sigma_l] = \text{eigs}(C_l^{-1} E_l^T B_l^{-1} E_l, k)$$

    by the generalized Lanczos algorithm.
    Compute $H_l = \Sigma_l (I - \Sigma_l)^{-1}$.
  end for
end procedure

Once an approximation to C_l^{-1} is available, the low-rank correction matrices W_l and H_l can be computed with the method in (4.2).

5.2. The multilevel scheme. The 3-level algorithm introduced in the previous section illustrates the basic idea of a multilevel algorithm. In a real implementation, the MSLR preconditioner construction follows a reverse order and proceeds from level $L-1$ to level 0. In this section we provide more details on this construction process.

First we apply an L-level HID ordering to reorder the matrix A and compute an incomplete Cholesky factorization of C_{L-1}:

$$C_{L-1} \approx M_{L-1}M_{L-1}^T.$$

We then traverse the HID tree from level $L-1$. We compute an incomplete Cholesky factorization of B_{L-1}:

$$B_{L-1} \approx L_{L-1}L_{L-1}^T,$$

and derive the low-rank correction by computing the largest k eigenpairs of the matrix $C_{L-1}^{-1}E_{L-1}^T B_{L-1}^{-1}E_{L-1}$,

$$[W_{L-1}, \Sigma_{L-1}] = \text{eigs}(C_{L-1}^{-1}E_{L-1}^T B_{L-1}^{-1}E_{L-1}, k).$$

A diagonal matrix H_{L-1} is then modified from Σ_{L-1} given by

$$H_{L-1} = \Sigma_{L-1}(I - \Sigma_{L-1})^{-1}.$$

Based on the analysis in Section 4, we know that

$$S_{L-1}^{-1} \approx C_{L-1}^{-1} + W_{L-1}H_{L-1}W_{L-1}^T.$$

We then go downward the HID tree and repeat the same operations performed on level $L-1$ for the remaining levels. At level 0, if B_0 is still an SPD matrix, we apply the incomplete Cholesky factorization to it, otherwise we apply the incomplete LDL factorization.

When applying the generalized Lanczos algorithm to compute W_l and Σ_l at each level l, we need to compute the matrix-vector products of $C_l^{-1}E_l^T B_l^{-1}E_l$ with arbitrary vectors. Since we have the factors of B_l, the matrix-vector product associated B_l^{-1} can be carried out by one forward and one backward substitutions. However, the same strategy cannot be applied to C_l^{-1} since we do not have the incomplete factors of C_l except at level $L-1$. Recall that $C_l^{-1} \equiv A_{l+1}^{-1}$ for $l < L-1$ and an approximate factorization of A_{l+1} has already been available after the computation at level $l+1$. Thus, in order to reduce computational cost, we can use this approximation to apply C_l^{-1} to an arbitrary vector. More specifically, we take the following approximation form

$$C_l \equiv A_{l+1} \approx \begin{pmatrix} L_{l+1} & \\ Y_{l+1}^T & I \end{pmatrix} \begin{pmatrix} D_{l+1} & \\ & T_{l+1} \end{pmatrix} \begin{pmatrix} L_{l+1}^T & Y_{l+1} \\ & I \end{pmatrix},$$

where $Y_{l+1} = D_{l+1}^{-1}L_{l+1}^{-1}E_{l+1}$ and $T_{l+1}^{-1} = C_{l+1}^{-1} + W_{l+1}H_{l+1}W_{l+1}^T \approx S_{l+1}^{-1}$ for $l < L-1$, and $C_{L-1} \approx M_{L-1}M_{L-1}^T$. Note that D_{l+1} equals to the identity matrix if B_{l+1} is SPD. Algorithm 1 provides a detailed illustration of the multilevel construction scheme. A recursive scheme for computing the product of C_l^{-1} with a vector b is described in Algorithm 2.

Note that the block independent set structure from the HID ordering can greatly improve the efficiency of the MSLR preconditioner in a number of ways. First, in the

ALGORITHM 2
A recursive formula for the approximation of $y = C_l^{-1} b$

procedure RSC(l,b)
  if $l = L - 1$ **then**
    return $y = M_{L-1}^{-T} M_{L-1}^{-1} b$.
  else
    Split $b = (b_1, b_2)^T$.
    Compute $z_1 = L_{l+1}^{-1} b_1$.
    Compute $z_2 = b_2 - E_{l+1}^T L_{l+1}^{-T} D_{l+1}^{-T} z_1$.
    Compute $z_1 = D_{l+1}^{-1} z_1$.
    Compute $y_2 = \mathtt{RSC}(l+1, z_2)$.
    Compute $y_2 = y_2 + W_{l+1} H_{l+1} W_{l+1}^T z_2$.
    Compute $y_1 = L_{l+1}^{-T} \left(z_1 - D_{l+1}^{-1} L_{l+1}^{-1} E_{l+1} y_2 \right)$.
    return $y = (y_1, y_2)^T$.
  end if
end procedure

TABLE 6.1
Application cost ξ_{appl} *and storage cost* σ_{mem} *of the MSLR preconditioner for the matrix A discretized on a regular grid, where r is the maximal rank used in the low-rank corrections.*

Grid	ξ_{appl}	σ_{mem}
2D ($N \times N$, $n = N^2$)	$O(\sqrt{r}n)$	$O(\sqrt{r}n)$
3D ($N \times N \times N$, $n = N^3$)	$O((r^{\frac{2}{3}} + \log n)n)$	$O((r^{\frac{2}{3}} + \log n)n)$

MSLR construction algorithm, all the diagonal blocks in B_l can be factored simultaneously. Second, in the application of the MSLR preconditioner to a vector v, which corresponds to applying Algorithm 2 by setting $l = -1$, i.e, $\mathtt{RSC}(-1, v)$, the triangular solves associated with B_l can be performed independently for each diagonal block in B_l. The computation of the matrix vector products in the generalized Lanczos algorithm can also benefit from this structure.

6. Complexity analysis. In this section, we show the computational complexity and the storage cost of the MSLR preconditioner for some model problems. For simplicity, we consider the case where the maximal rank used in the approximation of $S_l^{-1} - C_l^{-1}$ is bounded by a constant r. We also want to emphasize that the prefactors in these bounds can be large.

THEOREM 6.1. *Suppose the matrix A is discretized on a 2D* $N \times N$ *grid* ($n = N^2$) *or a 3D* $N \times N \times N$ *grid* ($n = N^3$), *and the maximal rank used in the low-rank approximation of* $S_l^{-1} - C_l^{-1}$ *is bounded by a constant r. Then the optimal application cost* ξ_{appl} *and storage cost* σ_{mem} *for the MSLR preconditioner on this matrix A satisfy the estimates listed in Table 6.1.*

Proof. The MSLR preconditioner requires to store the triangular factors and the low-rank correction matrices at each level. For the 2D case, each diagonal block in B_l has the size of $O(\frac{N}{2^{\lfloor \frac{L-l+1}{2} \rfloor}})$ and there are 2^{L-l} blocks in total, thus $|B_l|$ is $O(2^{\lfloor \frac{L-l}{2} \rfloor} N)$. The row size of W_l is equal to the row size of C_l, which is N when $l = L - 1$ and $\sum_{k=l+1}^{L-1} |B_k| + N$ for $l < L - 1$. Therefore, the storage for the low-rank correction

σ_{lrkmem} can be estimated as follows

$$\sigma_{\text{lrkmem}} = \sum_{l=0}^{L-2} \left(\sum_{k=l+1}^{L-1} |B_k| + N \right) r + Nr + Lr = O(2^{\frac{L}{2}} Nr),$$

which is an increasing function of L. On the other hand, the storage σ_{lumem} for the triangular factors can be estimated in the following way

$$\begin{aligned} \sigma_{\text{lumem}} &= \sum_{\mathbf{l}=1}^{L} 2^{L-l} O\left(\frac{N}{2^{\lfloor \frac{L-l+1}{2} \rfloor}} \right) + 2^L O\left((\frac{N}{2^{\lfloor \frac{L+1}{2} \rfloor}})^3 \right) \\ &= O(2^{\frac{L}{2}} N) + O(\frac{N^3}{2^{\frac{L}{2}}}). \end{aligned}$$

Here, we assume each diagonal block in B_l for $l > 0$ is a tridiagonal matrix and each diagonal block in B_0 is a banded matrix with bandwidth $O(\frac{N}{2^{\lfloor \frac{L+1}{2} \rfloor}})$. Therefore, the optimal $\sigma_{\text{men}} = O(\sqrt{r}n)$ is obtained when $2^L = O(\frac{N^2}{r})$ for the 2D case.

Similarly, the complexity for the 3D case can be analyzed as follows.

$$\sigma_{\text{lrkmem}} = \sum_{l=0}^{L-2} \left(\sum_{k=l+1}^{L-1} |B_k| + N^2 \right) r + N^2 r + Lr = O(2^{\frac{L}{3}} N^2 r),$$

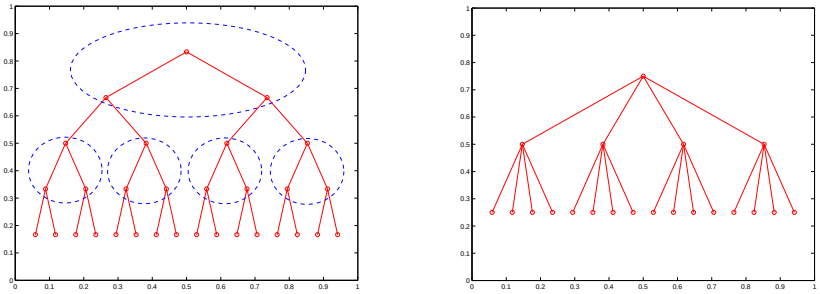
and

$$\begin{aligned} \sigma_{\text{lumem}} &= \sum_{l=1}^{L} 2^{L-l} O\left((\frac{N}{2^{\lfloor \frac{L-l+1}{3} \rfloor}})^3 \right) + 2^L O\left((\frac{N}{2^{\lfloor \frac{L+1}{3} \rfloor}})^5 \right) \\ &= O(N^3 L) + O(\frac{N^5}{2^{\frac{2L}{3}}}). \end{aligned}$$

The optimal $\sigma_{\text{men}} = O((r^{\frac{2}{3}} + \log n)n)$ is obtained when $2^L = O(\frac{N^3}{r})$. The application cost ξ_{appl} of the MSLR preconditioner is the same as in the memory storage. □

Note that in practice, we need to merge a few nodes on the binary HID tree into a new node to reduce the height of the HID tree and thus reduce the length of the recursion in the MSLR construction algorithm for better performance. More specifically, for those non-leaf nodes, we merge one node and its two children nodes into a new node for the 2D problems and merge one node, its two children nodes and four children nodes of those two children nodes into one new node for the 3D problems. See Figure 6.1 for a 2D example. In fact, this modification corresponds a quad/octa-partitioning of the mesh for 2D/3D cases, which will create fewer but larger connectors than the original binary HID ordering.

7. Numerical experiments. In this section, we illustrate the efficiency and robustness of the MSLR preconditioner with some sparse matrices from both 2D and 3D simulations. The testing platform consists of two Intel Xeon X5560 processors (8 MB cache, 4 cores each at 2.8 GHz) and 24 GB of memory. The MSLR preconditioner was implemented in C/C++, and the code was compiled by the Intel C compiler using the -O2 optimization level. We also used BLAS and LAPACK routines from Intel Math Kernel Library to boost the performance on multiple cores. The computation of the MSLR preconditioner was parallelized using OpenMP [25].



(i) A binary HID tree. (ii) A merged HID quadtree.

FIG. 6.1. *An illustration of the merged HID tree from a binary HID tree. Here, every three non-leaf nodes in the same blue dotted ellipse in the left binary tree are merged into a new node in the right quadtree.*

The HID ordering in the MSLR preconditioner was implemented through the `PartGraphRecursive` from METIS [16]. Each diagonal block in a MSLR preconditioner was reordered by the approximate minimum degree (AMD) ordering [1, 2] to reduce the fill-ins in the incomplete Cholesky or LDL factorizations. We consider those reordering procedures as a preprocessing step and do not include these in the MSLR preconditioner construction time. In the generalized Lanczos algorithm, we used the full reorthogonalization algorithm and set the maximal number of Lanczos steps to not exceed five times the number of required eigenvalues.

We compared the MSLR preconditioner with the incomplete Cholesky factorization with threshold dropping (ICT), the incomplete LDL factorization with threshold dropping (ILDLT) and the restricted additive Schwarz (RAS) method with one-level overlapping in the following tests. The accelerators we used included the conjugate gradient (CG) method [13] for the SPD cases, and the generalized minimal residual (GMRES) [26, 29] method with a restart dimension of 40, denoted by GMRES(40) for the indefinite cases. Since the RAS preconditioner is nonsymmetric even for the symmetric matrices, we used it along with GMRES(40) for all the numerical tests. For the ICT and the ILDLT preconditioners, the test matrices were first reordered by the AMD ordering. The right-hand side in all the tests was formed by taking $b = Ae$, where e is the vector of all ones.

The following notation is used throughout the section:

- fill: ratio of the number of non-zeros in the preconditioners over the number of non-zeros in the original matrix;
- p-t: wall clock time to build the preconditioners in seconds;
- its: number of iterations using CG or GMRES(40) with preconditioners to reduce the initial residual by a factor of 10^{-6}. We use "`F`" to indicate non-convergence within 300 iterations;
- i-t: wall clock time required in the iteration phase;
- rk: maximal rank used in the low-rank corrections;
- lev: number of levels in the binary tree from the HID ordering;
- mg: number of levels merged into one level in the binary HID tree.

TABLE 7.1
The fill-factor and iteration counts for solving a discretized negative Laplacian on a 50^3 grid along with the CG-MSLR *method. Here, the rank used in the low-rank correction is* 16 *and the threshold used in the incomplete Cholesky factorization is* 10^{-3}.

lev	fill from ICT	fill from low-rank	fill	its
5	11.6	.629	12.2	16
6	7.77	1.01	8.79	18
7	5.74	1.50	7.24	22
8	3.50	2.16	5.66	23
9	2.32	3.00	5.32	27
10	1.65	4.01	5.66	30
11	1.12	5.26	6.38	31

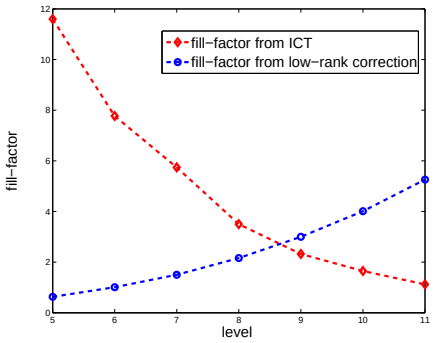


FIG. 7.1. *Illustration of fill-factors from ICT and low-rank correction with respect to different levels in Table 7.1.*

7.1. Model problems. We begin our tests of the MSLR preconditioner with the following model problem

$$\begin{aligned} -\Delta u - cu &= f \ \text{ in } \Omega, \\ u &= \phi(x) \text{ on } \partial\Omega, \end{aligned} \tag{7.1}$$

where the PDEs are defined over $\Omega = (0,1)^d$ with $d = 2$ or 3 and with Dirichlet boundary conditions. These PDEs are discretized by the 5-point stencil in 2D and 7-point stencil in 3D.

7.1.1. Effect of the number of levels. One important consideration when using the MSLR preconditioner is the number of levels. We studied its effect by solving (7.1) with $c = 0$ and discretizing it on a 50^3 grid. We fixed the rank to 16 and the threshold to 10^{-3} and gradually increased the number of levels from 5 to 11. We did not merge the levels of the binary HID tree in this example. As can be seen from Table 7.1, the total fill-factor first decreases from 12.2 to 5.32 as the number of levels increases from 5 to 9 and then increases to 6.38 as the number of levels further increases to 11. This shows that the fill-factor from the incomplete factorization is a decreasing function with *lev* while the fill-factor from the low-rank corrections is an increasing function with *lev*. The optimal fill-factor was obtained when the fill-factors from these two parts were almost the same which is when $lev = 9$ in this example, see Figure 7.1. The results in Table 7.1 and Figure 7.1 are consistent with our complexity analysis in Section 6.

7.1.2. Effect of the rank in the low-rank corrections. We next varied the rank used in the MSLR preconditioner and compared the iteration counts and the corresponding fill-factors to solve a discretized negative Laplacian on a 50^3 grid with the CG method. We fixed the threshold to 10^{-3} for the incomplete Cholesky factorization, *lev* to 7 and *mg* to 3 for this test matrix. The convergence results with respect to different ranks can be found in Table 7.2.

As the rank increases from 10 to 50, one can observe a steady improvement in the convergence in Table 7.2. However, larger ranks also lead to larger fill-factors and more computational time to compute the low-rank corrections. In the following numerical examples, we chose a moderate rank for the SPD problems and a relatively larger rank for the indefinite problems.

TABLE 7.2
Ranks and iteration counts for solving a discretized negative Laplacian on a 50^3 grid with the CG-MSLR *method. Here we fix the grid size to 50^3, lev to* 7 *and the threshold used in the incomplete Cholesky factorization to 10^{-3}.*

rk	fill from ICT	fill from low-rank	fill	its
10	3.56	.826	4.38	24
20	3.56	1.65	5.21	21
30	3.56	2.48	6.03	18
40	3.56	3.30	6.56	18
50	3.56	4.13	7.69	16

7.1.3. Preconditioning model problems. We first solve (7.1) with $c = 0$ and various grid sizes. We compared the convergence behavior between the MSLR, the ICT and the RAS preconditioners and reported the results in Table 7.3. As the grid size increased, we increased *lev* accordingly to control the fill-ins from the incomplete Cholesky factorization. For the 2D problem, we set *lev* to 5, 7 and 9 for the grid size 256^2, 512^2 and 1024^2, respectively. For the 3D problem, we chose *lev* to be 7, 10 and 13 for the grid size 32^3, 64^3 and 128^3, respectively. Here, the rank in the low-rank corrections was fixed to 16.

TABLE 7.3
Comparison of the ICT*, the* RAS *and the* MSLR *preconditioners for solving SPD linear systems from the 2-D/3-D PDE in (7.1) with the* CG *or the* GMRES *methods when $c = 0$.*

Grid	ICT-CG				RAS-GMRES				MSLR-CG						
	fill	p-t	its	i-t	fill	p-t	its	i-t	lev	mg	rk	fill	p-t	its	i-t
256^2	4.86	0.08	34	0.17	4.96	0.19	83	4.96	5	2	16	4.80	0.05	49	0.11
512^2	4.55	0.31	67	1.33	4.74	0.86	211	1.87	7	2	16	4.72	0.27	78	0.74
1024^2	4.58	1.20	128	11.1	5.21	4.38	F	–	9	2	16	4.74	1.16	159	6.49
32^3	4.27	0.06	16	0.05	4.31	0.17	26	0.03	7	3	16	4.13	0.10	17	0.03
64^3	4.14	0.46	30	0.76	4.12	1.63	49	0.53	10	3	16	4.07	0.85	35	0.47
128^3	4.29	3.67	57	12.40	4.20	21.90	121	19.90	13	3	16	4.16	10.18	66	8.21

Table 7.3 shows that with almost the same fill-factors, the CPU time to construct a MSLR preconditioner is the smallest among these three for 2D problems and becomes more expensive than the ICT preconditioner for 3D problems. In fact, the construction time of the MSLR preconditioner is dominated by the generalized Lanczos algorithm since the computation of each Lanczos vector needs a sequence of triangular solves and sparse/dense matrix-vector products. But the MSLR preconditioner can greatly reduce the CPU time in the iteration phases in both cases. We find that even though the iteration number of the MSLR-CG method is slightly larger than that of the ICT-CG method in Table 7.3, the MSLR-CG method only costs about 60% of the iteration time used by the ICT-CG method to achieve the same accuracy. This can be explained from two factors. First, all the triangular solves associated with the diagonal blocks in the same B_l can be performed independently. We have exploited this property with the simple thread-level parallelism by OpenMP. Second, a large portion of the fill-factor in the MSLR preconditioner is from the low-rank corrections. These dense matrix vector multiplication operations are quite efficient on modern

computer architectures. We expect to see a more substantial performance improvement when implementing the MSLR preconditioner on SIMD-type parallel machines such as those equipped with GPUs or Intel Xeon Phi processors in the future.

It is also interesting to compare the MSLR and the SLR [22] preconditioners on these model problems. We controlled *lev*, *rk* and the threshold such that the SLR preconditioner had roughly the same fill-factors as the MSLR and reported its performance in Table 7.4.

TABLE 7.4
Performance of the SLR-CG on the 3D model problems in Table 7.3.

Grid	fill	p-t	its	i-t
32^3	4.18	0.15	16	0.03
64^3	4.14	1.05	37	0.61
128^3	4.17	10.80	78	10.9

We find that the when the grid size is 32^3, SLR-CG method requires almost the same number of iterations and iteration times to converge as does the MSLR-CG method. However, as the grid size increases, the SLR-CG method becomes less efficient. The MSLR-CG method not only requires less CPU time to construct but also requires slightly fewer iterations and less iteration time to converge.

We then consider solving indefinite systems by setting $c > 0$ in (7.1). Here, we shifted the discretized negative Laplacian by sI. We fixed $s = 0.01$ for the 2D problems and $s = 0.04$ for the 3D problems. The convergence results are tabulated in Table 7.5. We can see that ILDLT-GMRES and RAS-GMRES methods failed on 5 out of the 6 test matrices while the MSLR-GMRES method only failed on the largest 3D test matrix. Compared with the SPD test matrices in Table 7.3, we observe that the MSLR preconditioner requires larger fill-factors to converge for these indefinite systems. The reason is twofold. First, for the indefinite matrices, we need to apply a smaller threshold in the incomplete factorization. We also noticed that the MSLR-GMRES method failed to converge when we chose the same number of levels as the SPD matrices in Table 7.3. This forced us to reduce the number of levels. Thus, the fill-factors from the incomplete factorization are larger than those in the SPD cases. Second, the MSLR preconditioner also required larger ranks in the low-rank corrections for indefinite systems. This factor not only contributed to a larger fill-factor but also significantly increased the CPU time in the construction of the MSLR preconditioner.

7.2. Preconditioning general matrices. To show that the MSLR preconditioner is generally applicable, we further tested it on 9 symmetric matrices from the University of Florida sparse matrix collection [8]. These matrices arise from different backgrounds, see Table 7.6 for a short description.

The convergence results of the three preconditioned methods are reported in Table 7.7. For the MSLR preconditioner, we chose *lev* and *mg* according to the matrix order, and nnz such that most fill-factors are around 4.5 and the largest one is 6.02 for the matrix `thermal2`. As can be seen, with almost the same fill-factors, the Krylov methods with the MSLR preconditioner achieved convergence for all the test matrices whereas they failed to converge with the other preconditioners on many cases, especially for the indefinite matrices. Although the MSLR preconditioner requires slightly more CPU time to construct than the ILDLT preconditioner for the matrices

TABLE 7.5
Comparison of the ILDLT*, the* RAS *and the* MSLR *preconditioners for solving symmetric indefinite linear systems from the 2-D/3-D PDE in (7.1) with the* GMRES *methods when* $c > 0$.

Grid	ILDLT-GMRES				RAS-GMRES				MSLR-GMRES						
	fill	p-t	its	i-t	fill	p-t	its	i-t	lev	mg	rk	fill	p-t	its	i-t
256^2	8.18	0.18	F	–	7.56	0.26	F	–	4	2	64	6.58	0.29	20	0.07
512^2	8.39	0.71	F	–	7.84	1.19	F	–	5	2	80	7.68	3.17	36	0.60
1024^2	12.6	5.34	F	–	19.40	22.90	F	–	6	2	180	9.13	41.09	76	6.30
32^3	5.89	0.11	21	0.11	5.78	0.09	40	0.06	7	3	32	5.60	0.25	17	0.04
64^3	7.05	1.03	F	–	11.40	3.01	F	–	10	3	64	7.06	7.44	187	3.97
128^3	9.35	12.20	F	–	10.2	31.20	F	–	13	3	64	8.07	80.20	F	–

TABLE 7.6
Example 2: Test matrices from the University of Florida Sparse Matrix Collection, where nnz stands for the number of non-zeros in the matrix.

Matrix	order	nnz	SPD	Origin
`cfd1`	70,656	1,825,580	yes	CFD problem
`cfd2`	123,440	3,085,406	yes	CFD problem
`Dubcova3`	146,689	3,636,643	yes	2-D/3-D PDE problem
`thermal1`	82,654	574,458	yes	thermal problem
`thermal2`	1,228,045	8,580,313	yes	thermal problem
`F2`	71,505	5,294,285	no	structural problem
`Lin`	256,000	1,766,400	no	structural problem
`qa8fk`	66,127	1,660,579	no	3-D acoustics problem
`vibrobox`	12,328	301,700	no	vibroacoustic problem

`Lin` and `qa8fk`, the Krylov methods with the MSLR preconditioner required the smallest iteration time to converge for all cases.

8. Conclusion. The MSLR preconditioner presented in this paper exploits a hierarchical graph decomposition called HID, that reorders the matrix into a multilevel block form where at each level the (1,1) block is block-diagonal. One such structure can be obtained from adapting the nested dissection ordering for example and this is what was used in the paper. The preconditioner is built by obtaining approximate inverses of certain Schur complements, exploiting in this process a low-rank property associated with these Schur complements at each level of the the HID tree. The proposed MSLR preconditioner appears to be more efficient and robust than ILU-type preconditioners, especially for indefinite systems. In our future work, we plan on extending these techniques to nonsymmetric and to complex (non-Hermitian) systems. We also plan to explore other low-rank approximation methods to reduce the pre-processing costs. Finally, another topic that is worth investigating further concerns the obtention of effective HID orderings, other than those extracted from Nested Dissection.

REFERENCES

TABLE 7.7
Comparison of the ICT *or the* ILDLT*, the* RAS *and the* MSLR *preconditioners for solving general symmetric linear systems with the* CG *or* GMRES(40) *methods.*

Matrix	ICT/ILDLT				RAS				MSLR						
	fill	p-t	its	i-t	fill	p-t	its	i-t	lev	mg	rk	fill	p-t	its	i-t
cfd1	5.81	3.40	298	11.7	5.61	1.83	F	–	7	1	64	5.00	1.42	85	0.96
cfd2	4.47	3.20	271	13.60	4.52	2.65	F	–	8	2	50	4.47	1.53	155	2.58
Dubcova3	1.19	0.47	45	0.93	1.18	1.17	54	0.45	5	1	64	1.17	0.39	20	0.18
thermal1	4.57	0.20	52	0.50	4.55	0.38	235	0.77	6	3	50	4.50	0.18	42	0.16
thermal2	5.98	4.21	90	17.80	5.87	8.11	F	–	8	3	64	6.02	3.29	87	6.02
F2	2.82	3.43	F	–	2.90	3.46	F	–	6	2	64	2.49	1.24	79	1.51
Lin	4.61	0.53	F	–	5.36	4.63	F	–	10	3	22	4.55	0.88	185	2.96
qa8fk	1.90	0.26	17	0.20	4.56	1.16	25	0.20	7	2	32	1.89	0.44	21	0.14
vibrobox	4.13	0.43	F	–	4.35	0.36	F	–	5	1	16	3.86	0.18	57	0.16

[1] P. R. AMESTOY, T. A. DAVIS, AND I. S. DUFF, *An approximate minimum degree ordering algorithm*, SIAM J. Matrix Anal. Appl., 17 (1996), pp. 886–905.

[2] ———, *Algorithm 837: An approximate minimum degree ordering algorithm*, ACM Trans. Math. Software, 30 (2004), pp. 381–388.

[3] M. BENZI AND M. TUMA, *A sparse approximate inverse preconditioner for nonsymmetric linear systems*, SIAM J. Sci. Comput., 19 (1998), pp. 968–994.

[4] E. CHOW AND Y. SAAD, *Experimental study of ilu preconditioners for indefinite matrices*, J. Comput. Appl. Math., 86 (1997), pp. 387 – 414.

[5] ———, *Approximate inverse preconditioners via sparse-sparse iterations*, SIAM J. Sci. Comput., 19 (1998), pp. 995–1023.

[6] ———, *Preconditioned krylov subspace methods for sampling multivariate gaussian distributions*, SIAM J. Sci. Comput., 36 (2014), pp. A588–A608.

[7] E. CORONA, P.-G. MARTINSSON, AND D. ZORIN, *An $O(N)$ direct solver for integral equations on the plane*, Appl. Comput. Harmon. Anal., (2014), pp. –.

[8] T. A. DAVIS AND Y. HU, *The University of Florida Sparse Matrix Collection*, ACM Trans. Math. Software, 38 (2011), pp. 1:1–1:25.

[9] B. ENGQUIST AND L. YING, *Sweeping preconditioner for the helmholtz equation: Hierarchical matrix representation*, Commun. Pure Appl. Math., 64 (2011), pp. 697–735.

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

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

[12] W. HACKBUSCH AND B. N. KHOROMSKIJ, *A sparse $\mathcal{H}$-matrix arithmetic. Part II: Application to multi-dimensional problems*, Computing, 64 (2000), pp. 21–47.

[13] M. R. HESTENES AND E. L. STIEFEL, *Methods of conjugate gradients for solving linear systems*, J. Res. Natl. Bur. Stand., 49 (1952), pp. 409–436.

[14] K. L. HO AND L. YING, *Hierarchical interpolative factorization for elliptic operators: differential equations*. Preprint, 2013, arXiv:1307.2895 [math.NA].

[15] P. HNON AND Y. SAAD, *A parallel multistage ilu factorization based on a hierarchical graph decomposition*, SIAM J. Sci. Comput., 28 (2006), pp. 2266–2293.

[16] G. KARYPIS AND V. KUMAR, *A fast and high quality multilevel scheme for partitioning irregular graphs*, SIAM J. Sci. Comput, 20 (1998), pp. 359–392.

[17] S. LE BORNE, *$\mathcal{H}$-matrices for convection-diffusion problems with constant convection*, Computing, 70 (2003), pp. 261–274.

[18] S. LE BORNE AND L. GRASEDYCK, *$\mathcal{H}$-matrix preconditioners in convection-dominated problems*, SIAM J. Matrix Anal. Appl., 27 (2006), pp. 1172–1183.

[19] R. LI AND Y. SAAD, *Divide and conquer low-rank preconditioners for symmetric matrices*, SIAM J. Sci. Comput., 35 (2013), pp. A2069–A2095.

[20] ———, *GPU-accelerated preconditioned iterative linear solvers*, Journal Supercomput., 63 (2013), pp. 443–466.
[21] ———, *Low-rank correction methods for algebraic domain decomposition preconditioners*. Submitted, 2014.
[22] R. Li, Y. Xi, and Y. Saad, *Schur complement based low-rank correction method for algebraic domain decomposition preconditioners*. Submitted, 2014.
[23] 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.
[24] J. K. Merikoski and R. Kumar, *Inequalities for spreads of matrix sums and products*, Appl. Math. E-Notes, 4 (2004), pp. 150–159.
[25] OpenMP Architecture Review Board, *OpenMP application program interface version 3.1*, July 2011.
[26] Y. Saad, *A flexible inner-outer preconditioned gmres algorithm*, SIAM J. Sci. Comput., 14 (1993), pp. 461–469.
[27] ———, *Ilum: A multi-elimination ilu preconditioner for general sparse matrices*, SIAM J. Sci. Comput., 17 (1996), pp. 830–847.
[28] Y. Saad, *Iterative Methods for Sparse Linear Systems, 2nd edition*, SIAM, Philadelpha, PA, 2003.
[29] Y. Saad and M. Schultz, *Gmres: A generalized minimal residual algorithm for solving non-symmetric linear systems*, SIAM J. Sci. Statist. Comput., 7 (1986), pp. 856–869.
[30] Y. Saad and B. Suchomel, *Arms: an algebraic recursive multilevel solver for general sparse linear systems*, Numer. Linear Algebra Appl., 9 (2002), pp. 359–378.
[31] Y. Saad and J. Zhang, *Bilum: Block versions of multielimination and multilevel ilu preconditioner for general sparse linear systems*, SIAM J. Sci. Comput., 20 (1999), pp. 2103–2121.
[32] B. Smith, *Domain decomposition algorithms for the partial differential equations of linear elasticity*, tech. report, New York, 1990.
[33] B. Smith, P. Bjørstad, and W. Gropp, *Domain Decomposition: Parallel Multilevel Methods for Elliptic Partial Differential Equations*, Cambridge University Press, New York, NY, USA, 1996.
[34] Y. Xi, J. Xia, S. Cauley, and V. Balakrishnan, *Superfast and stable structured solvers for toeplitz least squares via randomized sampling*, SIAM J. Matrix Anal. Appl., 35 (2014), pp. 44–72.
[35] Y. Xi, J. Xia, and R. Chan, *A fast randomized eigensolver with structured ldl factorization update*, SIAM J. Matrix Anal. Appl., 35 (2014), pp. 974–996.
[36] J. Xia, *Efficient structured multifrontal factorization for general large sparse matrices*, SIAM J. Sci. Comput., 35 (2013), pp. A832–A860.
[37] ———, *Randomized sparse direct solvers*, SIAM J. Matrix Anal. Appl., 34 (2013), pp. 197–227.
[38] J. Xia, S. Chandrasekaran, M. Gu, and X. S. Li, *Fast algorithms for hierarchically semiseparable matrices*, Numer. Linear Algebra Appl., 17 (2010), pp. 953–976.
[39] ———, *Superfast multifrontal method for large structured linear systems of equations*, SIAM J. Matrix Anal. Appl., 31 (2010), pp. 1382–1411.
[40] 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.