

RESEARCH ARTICLE

Multi-Color Low-Rank preconditioner for general sparse linear systems

Qingqing Zheng¹ | Yuanzhe Xi^{*2} | Yousef Saad³¹Department of Mathematical Sciences,
Tsinghua University, Beijing, China²Department of Mathematics, Emory
University, GA, U.S.A³Computer Science & Engineering,
University of Minnesota, Twin Cities, MN,
U.S.A**Correspondence**

*Yuanzhe Xi, Department of Mathematics,
Emory University, Atlanta, GA 30322,
U.S.A. Email: yxi26@emory.edu
Research supported in part by the U.S.
National Science Foundation under grants
DMS-1521573 and DMS-1912048, Shuimu
Scholar of Tsinghua University and the
Minnesota Supercomputing Institute

Summary

This paper presents a multilevel parallel preconditioning technique for solving general large sparse linear systems of equations. Subdomain coloring is invoked to reorder the coefficient matrix by multicoloring the adjacency graph of the subdomains, resulting in a two-level block diagonal structure. A full binary tree structure \mathcal{T} is then built to facilitate the construction of the preconditioner. A key property that is exploited is the observation that the difference between the inverse of the original matrix and that of its block diagonal approximation is often well approximated by a low-rank matrix. This property and the block diagonal structure of the reordered matrix lead to a Multi-Color Low-Rank (MCLR) preconditioner. The construction procedure of the MCLR preconditioner follows a bottom-up traversal of the tree \mathcal{T} . All irregular matrix computations, such as ILU factorizations and related triangular solves, are restricted to leaf nodes where these operations can be performed independently. Computations in non-leaf nodes only involve easy-to-optimize dense matrix operations. In order to further reduce the number of iteration of the Preconditioned Krylov subspace procedure, we combine MCLR with a few classical block-relaxation techniques. Numerical experiments on various test problems are proposed to illustrate the robustness and efficiency of the proposed approach for solving large sparse symmetric and nonsymmetric linear systems.

KEYWORDS:

Low-rank approximation, parallel preconditioners, domain decomposition, recursive multilevel methods, Krylov subspace methods

1 | INTRODUCTION

In this paper, we consider solving a general large sparse linear system of the form

$$Ax = b, \quad (1)$$

where $A \in \mathbb{R}^{n \times n}$ and $b \in \mathbb{R}^n$, by iterative methods. Preconditioned Krylov subspace methods can solve equation (1) efficiently when a good preconditioner is available. In this approach a preconditioner M is obtained and a Krylov subspace method is then applied to solve an equivalent linear system such as $M^{-1}Ax = M^{-1}b$. A basic requirement on M is that it should be inexpensive to solve systems with the matrix M , since such operations are performed at every step of the iterative procedure. A class of general purpose preconditioners is one based on Incomplete LU (ILU) factorizations, see, e.g.,^{26, 28, 30}. These preconditioners

are relatively easy to obtain and are effective for a large class of sparse linear systems. However, their performance deteriorates when the coefficient matrix is ill-conditioned or indefinite. In addition, due to their sequential nature, their performance is unacceptably poor on highly parallel computers, e.g., those equipped with GPUs¹⁸ or Intel Xeon Phi processors.

Sparse approximate inverse preconditioners^{5, 10} were originally developed to overcome these difficulties. However, their cost can be quite high in both arithmetic and memory usage. Recently, a new class of approximate inverse preconditioners were developed that exploit low-rank properties rather than the standard nonzero sparsity. These techniques show superior performance on modern architectures. The Schur complement low-rank (SLR) preconditioner¹⁹ is based on the domain decomposition framework. The Multilevel Schur complement Low-Rank (MSLR) preconditioner³¹ utilizes the multilevel Hierarchical Interface Decomposition (HID) ordering¹² to further improve the scalability of the SLR preconditioner. Generalized Multilevel Schur complement Low-Rank (GMSLR) preconditioner⁷ extends the applicability of the MSLR preconditioner to nonsymmetric problems. These preconditioners all seek to exploit the low-rank property associated with the inverse of the Schur complement. On the other hand, the Multilevel Low-Rank (MLR) preconditioner¹⁷ takes advantage of the low-rank property associated with the difference between its inverse and the inverse of a block diagonal matrix. This approach can be quite efficient especially when solving symmetric indefinite linear systems. Another class of preconditioners that exploit the low-rank properties are rank structured methods. They include \mathcal{H} -matrix^{2, 3}, HOLDR-matrix¹, \mathcal{H}^2 -matrix¹¹ and hierarchically semiseparable (HSS) matrices^{4, 8, 14, 15}. These techniques partition the coefficient matrix into blocks and approximate certain off-diagonal blocks with low-rank matrices. They have recently been embedded into sparse matrix techniques to compress the intermediate dense Schur complements leading to rank structured sparse direct solvers and preconditioners^{20, 22, 23, 25, 33–35}.

Following the same direction, this paper presents a preconditioner called Multi-Color Low-Rank (MCLR) preconditioner that combines domain decomposition with domain coloring ideas. In contrast with previous techniques such as SLR, MSLR and GMSLR preconditioners which are Schur complement based methods, where the low-rank property associated with the inverse of Schur complement is exploited, the MCLR preconditioner explores the low-rank decay property of the matrix $I - AA_0^{-1}$, where A_0 is the block diagonal part of the reordered coefficient matrix A . By making use of this block structure of the reordered matrix recursively, MCLR has higher parallelism. First of all, the size of generators obtained from the bisection varies a lot across different levels of the tree while the size of each block from multicolor ordering is more balanced. This is beneficial for load balancing. Second, in order to limit the size of the diagonal blocks at the leaf level (to reduce the computational cost), the binary tree used in the MLR preconditioner requires more levels than that of the MCLR preconditioner. Because of a larger height of the tree used, the approximation accuracy of the MLR preconditioner is inferior to that of the MCLR preconditioner, which in turn requires a larger rank to compensate for the accuracy loss. For the same reason, the rank used in the MCLR preconditioner tends to also be much lower than that used in the MSLR/GMSLR preconditioner. Since the cost of computing the low-rank correction terms can be quite high, this property can dramatically reduce the cost of building the MCLR preconditioner. Moreover, although these low-rank approximate inverse preconditioners utilize a binary tree structure to perform operations, the MCLR preconditioner organizes the operations in the tree in a different way from its counterparts. It restricts all the irregular matrix operations such as ILU factorizations and the resulting triangular solves to leaf nodes only, which can be performed independently among different nodes. Since each leaf node is associated with a block diagonal matrix, the operations within each leaf node can also be easily parallelized. The non-leaf nodes only involve easy-to-optimize dense low-rank correction computations. In addition, the tree associated with the MCLR preconditioner often has a much smaller height for a given matrix. This reduced height further improves the approximation accuracy and limits the number of recursive calls in both the construction and application phases. In order to improve the MCLR convergence in practice, we also propose a modified scheme that combines the MCLR preconditioner with classical relaxation type methods.

The organization of the paper is as follows. In Section 2, we briefly review the multicoloring algorithm used to reorder the coefficient matrix. We analyze the low rank property associated with an inverse matrix and present a multilevel preconditioner based on this property in Section 3. In Section 4, we provide numerical results from model problems as well as some general matrices from the SuiteSparse matrix collection⁶. Finally, concluding remarks are presented in Section 5.

2 | MULTICOLOR ORDERING

Graph coloring is a technique used to label (color) the nodes of a graph such that no two adjacent nodes have the same label (color). In the context of relaxation-type iterative methods, graph coloring helps improve parallelism²⁶. This section gives a brief review of the greedy multicoloring approach and discusses an alternative that exploits (group) independent sets.

Classical multicoloring is applied to the adjacency graph of the coefficient matrix A in (1), see, e.g.,²⁶ Sec. 12.4. The vertices of the adjacency graph are colored (labeled) in such a way that no two adjacent vertices have the same color. If the matrix is reordered in such a way that the nodes are listed by color, then it will have a block structure in which each diagonal block is a diagonal matrix, corresponding to a color. The number of the colors required can be large in situations where the vertices have a high degree and this can result in slow convergence for the iterative solution procedure in a parallel environment. In order to keep a good balance between parallel efficiency and speed of convergence, it is possible to combine domain decomposition techniques with multicoloring by first partitioning the graph and then coloring the partitions (or ‘subdomains’) by a multicoloring algorithm. This technique is common for unravelling parallelism in the multiplicative Schwarz procedure, see, e.g.,²⁶ Sec. 14.3. The standard greedy algorithm applied for multicoloring subdomains obtained from a graph partitioner is sketched in Algorithm 1. Here, $\text{Adj}(D_i)$ denotes the set of the subdomains that are adjacent to subdomain D_i in the graph.

Algorithm 1 Subdomain coloring with the greedy algorithm

- 1: Partition vertices of the adjacency graph \mathcal{A} of A into n_d subdomains $\{D_i\}$
 - 2: **for** $i = 1, 2, \dots, n_d$ **do**
 - 3: Set $\text{Color}(D_i)=0$
 - 4: **end for**
 - 5: **for** $i = 1, 2, \dots, n_d$ **do**
 - 6: Set $\text{Color}(D_i)=\min\{k > 0 | k \neq D_j, \forall j \in \text{Adj}(D_i)\}$
 - 7: **end for**
-

Step 1 in Algorithm 1 can be implemented by any graph partitioner such as METIS¹⁶. Steps 2-7 apply the greedy multicoloring algorithm on $\{D_i\}$. Figure 1 shows an illustration of the greedy multicoloring algorithm in action. In the figure the central node is the node being visited and it will be assigned color number 3, which is the smallest admissible color. Note that a label of zero means ‘not yet colored’.

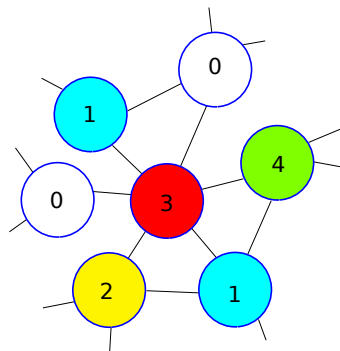


FIGURE 1 Coloring 7 vertices, where 0 indicates the node has not been colored yet.

The matrix A in (1) can be reordered by renumbering the unknowns so that they are listed by color: nodes of color 1, followed by those of color 2, etc. This results in the permuted system

$$(P^T A P)(P^T x) = P^T b, \quad (2)$$

where P denotes the permutation matrix obtained from Algorithm 1. Because the original system (1) disappears from the picture in the remainder of the paper, we will still denote by A the permuted matrix $P^T A P$ and by b the permuted right-hand side $P^T b$. As is well-known the result of this permutation is that the new matrix has a block structure with c diagonal blocks where c is the number of colors. Each diagonal block corresponds to one set D_i and is in a block diagonal form. Figure 2 is an illustration with a reordered discretized 3D Laplacian using centered finite differences with Dirichlet boundary conditions. There are $c = 4$

colors in this case. The first three colors each has four sets (subdomains) and the last one has three sets. The block corresponding to a color is a block diagonal matrix.

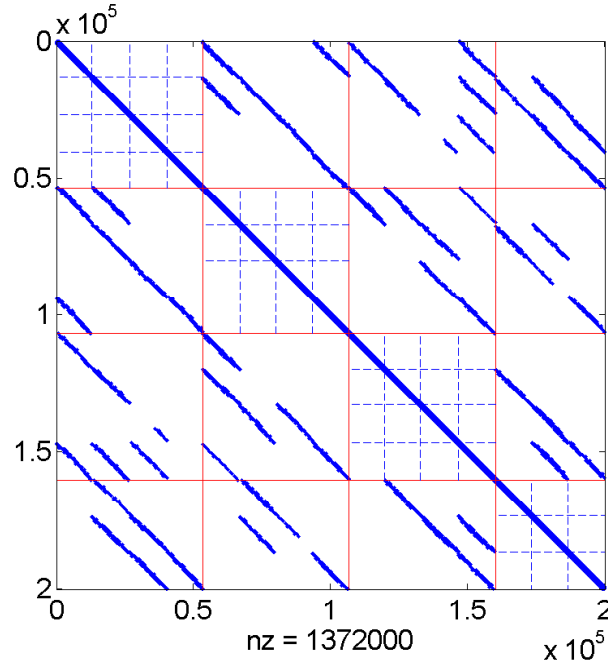


FIGURE 2 Nonzero pattern of a reordered discretized negative 3D Laplacian with the zero Dirichlet boundary condition using Algorithm 1 with 4 colors and $n_d = 15$.

3 | MULTI-COLOR LOW-RANK PRECONDITIONER

This section considers the construction and application of the Multi-Color Low-Rank preconditioner. For motivation, we begin by describing a two-level method and then discuss the multilevel approach which exploits multicoloring.

3.1 | A Two-level method

This section discusses a way to compute an approximation to A^{-1} in order to define a two-level version of the proposed preconditioner. Assume that we have a general symmetric positive definite (SPD) matrix A partitioned into a block 2×2 form as shown below, and call A_0 its block diagonal:

$$A = \begin{pmatrix} A_{11} & A_{12} \\ A_{12}^T & A_{22} \end{pmatrix}, \quad A_0 = \begin{pmatrix} A_{11} & \\ & A_{22} \end{pmatrix}, \quad (3)$$

where $A_{11} \in \mathbb{R}^{p \times p}$ and $A_{22} \in \mathbb{R}^{q \times q}$ with $p + q = n$. The premise of the method developed in this paper is similar to that of earlier papers^{7, 19, 31} namely, we exploit the property, to be established in Section 3.3, that the matrix $A^{-1} - A_0^{-1}$ can be well approximated by a low-rank matrix.

However, it is not practical from a computational point of view to seek an approximation of $A^{-1} - A_0^{-1}$. Indeed an Arnoldi-type procedure with this matrix would require solving linear systems with A at each step. Instead, we first compute a low-rank approximation to

$$I - AA_0^{-1} = A(A^{-1} - A_0^{-1}). \quad (4)$$

The matrix $I - AA_0^{-1}$ inherits the eigenvalue low-rank decay property satisfied by $A^{-1} - A_0^{-1}$ but it is much easier to approximate. For example, we can compute such an approximation by performing a few steps of the Arnoldi procedure with $I - AA_0^{-1}$, and

this will yield an approximate factorization of the form:

$$I - AA_0^{-1} \approx VHV^T, \quad (5)$$

where $V \in \mathbb{R}^{n \times k}$ is a matrix with orthonormal columns and $H \in \mathbb{R}^{k \times k}$ is an upper Hessenberg matrix. This scheme only requires linear system solutions associated with A_{11} and A_{22} , which can be obtained by either a direct or an iterative method. If the matrix is large, we may be able to develop a multilevel method by applying the same process recursively to the blocks A_{11} and A_{22} and this will be discussed in detail in the next section. We have $A \approx (I - VHV^T)A_0$ and we can exploit the Sherman-Morrison formula:

$$A^{-1} \approx A_0^{-1}(I - VHV^T)^{-1} \quad (6)$$

$$\begin{aligned} &= A_0^{-1}(I + VH(I - H)^{-1}V^T) \\ &= A_0^{-1}(I + V[(I - H)^{-1} - I]V^T) \\ &= A_0^{-1}(I + VGV^T), \end{aligned} \quad (7)$$

where $G = (I - H)^{-1} - I$ is a k -by- k matrix. The above formula shows how to define what we call a two-level preconditioner: the matrix $A_0^{-1}(I + VGV^T)$ defines the inverse of the preconditioner and requires only that we solve systems with the matrix A_0 and then apply a low-rank correction. When the size of the matrix A under consideration is large, approximating A^{-1} with (7) may become expensive and so it becomes mandatory to extend the scheme discussed above into a multilevel technique.

3.2 | A multilevel scheme

We will now combine multicoloring with the scheme (7) to obtain a hierarchical approximation to A^{-1} . This approximation can be used as a preconditioner for solving linear systems associated with the matrix A .

The procedure starts with reordering the coefficient matrix using Algorithm 1. Assuming that c different colors are found, then a full binary tree \mathcal{T} with c leaf nodes is constructed to facilitate subsequent matrix computations. Figure 3 shows an illustration of a reordered matrix with 7 colors obtained by Algorithm 1 and its associated full binary tree \mathcal{T} . Each node i of \mathcal{T} is associated with an index set \mathbf{I}_i . For a leaf node i , \mathbf{I}_i denotes the set of row/column indices of the i th diagonal block in the reordered matrix. It is associated with a color. Recall that subdomains of the same color are not coupled, and so the submatrix associated with one given color has a block diagonal structure. Figure 2 illustrates a case with 4 colors in a partition with 15 subdomains. For any non-leaf node i that has c_1 and c_2 as its two children, \mathbf{I}_i is defined to be

$$\mathbf{I}_i = \mathbf{I}_{c_1} \cup \mathbf{I}_{c_2}. \quad (8)$$

In the following sections, we will use the notation

$$A|_{\mathbf{I}_i, \mathbf{I}_j}$$

to denote a submatrix of A with row index set \mathbf{I}_i and column index set \mathbf{I}_j and

$$A|_{\mathbf{I}_i}$$

to denote a submatrix of A with row index set \mathbf{I}_i .

The procedure to build the preconditioner follows a bottom-up traversal of \mathcal{T} . The basic idea is to recursively utilize (7) to reduce the computational cost in approximating each $(A|_{\mathbf{I}_i, \mathbf{I}_i})^{-1}$. For convenience, the computed approximation to $(A|_{\mathbf{I}_i, \mathbf{I}_i})^{-1}$ will be denoted by M_i^{-1} :

$$M_i^{-1} \approx (A|_{\mathbf{I}_i, \mathbf{I}_i})^{-1}.$$

If i is a leaf node, M_i^{-1} is defined to be the inverse obtained from the ILU factorization of $A|_{\mathbf{I}_i, \mathbf{I}_i}$:

$$M_i^{-1} = U_i^{-1}L_i^{-1},$$

where L_i and U_i are the incomplete LU factors of $A|_{\mathbf{I}_i, \mathbf{I}_i}$. Since $A|_{\mathbf{I}_i, \mathbf{I}_i}$ has a block diagonal form, its ILU factorization can be performed independently for each diagonal block.

If i is a non-leaf node with children c_1, c_2 , then, based on (7), we define :

$$M_i^{-1} = \begin{pmatrix} M_{c_1}^{-1} & \\ & M_{c_2}^{-1} \end{pmatrix} (I + V_i G_i V_i^T), \quad (9)$$

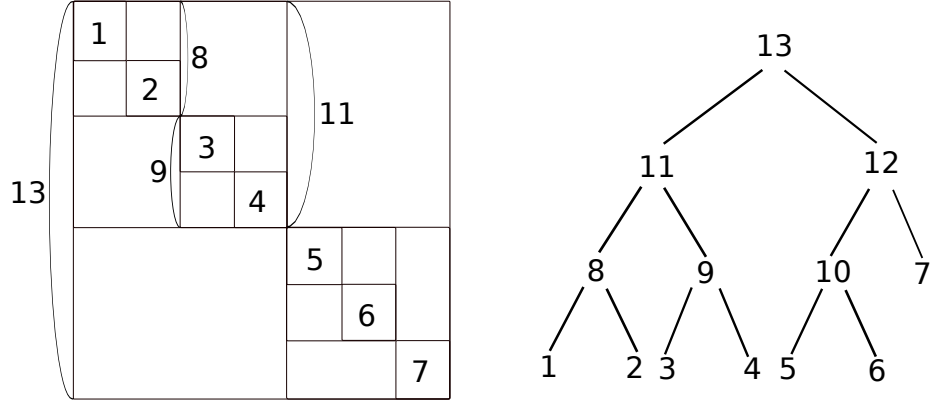


FIGURE 3 The block diagonal structure of a reordered matrix using Algorithm 1 with 7 colors (left) and its associated full binary tree \mathcal{T} (right). Each leaf node of \mathcal{T} corresponds to one diagonal block in the reordered matrix.

where $V_i G_i V_i^T$ is a rank- k matrix. When computing $V_i G_i V_i^T$ based on (5) and (7), the Arnoldi procedure requires a subroutine to solve linear systems associated with the matrix

$$\begin{pmatrix} A|_{\mathbf{I}_{c_1}, \mathbf{I}_{c_1}} & \\ & A|_{\mathbf{I}_{c_2}, \mathbf{I}_{c_2}} \end{pmatrix}$$

at each iteration. This operation will be replaced by matrix-vector multiplications associated with $\begin{pmatrix} M_{c_1}^{-1} & \\ & M_{c_2}^{-1} \end{pmatrix}$. The action of M_i^{-1} on a vector is described in Algorithm 2.

Algorithm 2 Computing $\mathbf{x} = M_i^{-1} \mathbf{r}$

- 1: **procedure** $\mathbf{x} = \text{RecSolv}(\mathbf{r}, \mathbf{i})$
 - 2: **if** i is leaf node **then**
 - 3: Solve $L_i U_i \mathbf{x} = \mathbf{r}$
 - 4: **else**
 - 5: Compute $\mathbf{z} = \mathbf{r} + V_i(G_i(V_i^T \mathbf{r}))$.
 - 6: Partition \mathbf{z} into $\begin{pmatrix} \mathbf{z}_{c_1} \\ \mathbf{z}_{c_2} \end{pmatrix}$ – according to sets $\mathbf{I}_{c_1}, \mathbf{I}_{c_2}$
 - 7: Solve $\mathbf{y}_1 = \text{RecSolv}(\mathbf{z}_{c_1}, c_1)$.
 - 8: Solve $\mathbf{y}_2 = \text{RecSolv}(\mathbf{z}_{c_2}, c_2)$.
 - 9: Stack $\mathbf{x} = \begin{pmatrix} \mathbf{y}_1 \\ \mathbf{y}_2 \end{pmatrix}$.
 - 10: **end if**
 - 11: **end procedure**
-

In summary, the matrix M_i^{-1} related to the application of Algorithm 2 satisfies the following relation

$$M_i^{-1} = \begin{cases} \begin{pmatrix} M_{c_1}^{-1} & \\ & M_{c_2}^{-1} \end{pmatrix} (I + V_i G_i V_i^T), & \text{if } i \text{ is non-leaf node,} \\ U_i^{-1} L_i^{-1}, & \text{if } i \text{ is leaf node.} \end{cases} \quad (10)$$

Note that when the root node root of \mathcal{T} is reached, we obtain

$$A^{-1} \approx M_{\text{root}}^{-1}. \quad (11)$$

This M_{root}^{-1} can be used as a preconditioner for solving (2) and its construction procedure is summarized in Algorithm 3.

Algorithm 3 Construction of M_{root}^{-1}

```

1: Apply Algorithm 1 to reorder the coefficient matrix
2: Construct a binary tree  $\mathcal{T}$  with  $L$  levels
3: for level  $l$  from  $L - 1$  to  $0$  do
4:   for node  $i$  on level  $l$  do
5:     if  $i$  is leaf node then
6:       Compute incomplete LU factorization  $A|_{I_i, I_i} \approx L_i U_i$  and set  $M_i = L_i U_i$ 
7:     else
8:       Compute  $k$  eigenpairs by the Arnoldi procedure
9:          $[V_i, H_i] = \text{Arnoldi} \left( I - A|_{I_i, I_i} \begin{pmatrix} M_{c_1}^{-1} & \\ & M_{c_2}^{-1} \end{pmatrix}, k \right)$ 
10:        Set  $M_i^{-1} = \begin{pmatrix} M_{c_1}^{-1} & \\ & M_{c_2}^{-1} \end{pmatrix} (I + V_i G_i V_i^T)$ 
11:      end if
12:    end for
13: end for

```

Since Algorithm 3 loops in \mathcal{T} by level, the construction procedure can be easily parallelized based on both the binary tree structure and the block diagonal structure associated with each leaf node.

3.3 | Decay property

We now study the low-rank property associated with the 2-level approximation of Section 3.1. Consider Equation (12). Since A is SPD, A_{11} and A_{22} are also SPD and admit square root factorizations. Hence, we have

$$A = A_0^{1/2} \begin{pmatrix} I & A_{11}^{-\frac{1}{2}} A_{12} A_{22}^{-\frac{1}{2}} \\ A_{22}^{-\frac{1}{2}} A_{12}^T A_{11}^{-\frac{1}{2}} & I \end{pmatrix} A_0^{1/2} = A_0^{1/2} \begin{pmatrix} I & \Phi \\ \Phi^T & I \end{pmatrix} A_0^{1/2}, \quad (12)$$

where $\Phi = A_{11}^{-\frac{1}{2}} A_{12} A_{22}^{-\frac{1}{2}} \in \mathbb{R}^{p \times q}$. We will denote by J_0 the matrix

$$J_0 \equiv \begin{pmatrix} I & \Phi \\ \Phi^T & I \end{pmatrix} = A_0^{-1/2} A A_0^{-1/2}. \quad (13)$$

From this we can write:

$$A^{-1} - A_0^{-1} = A_0^{-1/2} [J_0^{-1} - I] A_0^{-1/2}. \quad (14)$$

We would like to show that the matrix

$$J = J_0^{-1} - I \quad (15)$$

can be well approximated by low-rank matrices. It is well-known, see e.g.,⁹ sec. 8.6.1, that if Φ has rank r , then the matrix $\begin{pmatrix} 0 & \Phi \\ \Phi^T & 0 \end{pmatrix}$ has $2r$ nonzero eigenvalues given by $\pm \sigma_i$ where $\sigma_i, i = 1, \dots, r$, are the r singular values of Φ and the eigenvalue zero repeated $n - 2r$ times. This results in the following lemma.

Lemma 1. Let $\Phi \in \mathbb{R}^{p \times q}$ be of rank r and σ_i ($i = 1, 2, \dots, r$) its non-zero singular values. Then the matrix J_0 defined by (13) has $2r$ eigenvalues of the form $1 \pm \sigma_i$ for $i = 1, 2, \dots, r$, and the eigenvalue one of multiplicity $n - 2r$.

Therefore, the matrix J has eigenvalues $\lambda_i = 0$ with multiplicity $n - 2r$ and $\lambda_i = \frac{1}{1 \pm \sigma_i} - 1$, where σ_i ($i = 1, 2, \dots, r$) are the non-zero singular values of Φ . The derivative of $\lambda = \frac{1}{1 - \sigma} - 1$ with respect to σ is

$$\frac{d\lambda}{d\sigma} = \frac{1}{(1 - \sigma)^2}. \quad (16)$$

This derivative can become large when σ increases or decreases toward one.

Consider now the matrix $A_0^{-1}A$ which has the same eigenvalues as the matrix J_0 defined above. This is the Block-Jacobi iteration matrix and from what is known in the domain decomposition literature, its eigenvalues are between 0 and the number of subdomains, which is equal to two in this context, see²⁶ Theorem 14.5. Therefore, the eigenvalues of $J_0 - I$ are between -1 and 1 and this shows that the singular values σ_i of Φ are less than one. Those singular values that are close to one, correspond to the largest eigenvalues of J_0 , i.e., those that are close to 2. These are often termed high frequency modes in the multigrid literature. Thus, the largest singular values σ_i that are close to one, will yield large eigenvalues for J that are of the form $\lambda_i = (1 - \sigma_i)^{-1} - 1$ and these will be well separated because of the derivative (16). The result is that the matrix J can be well approximated by a low-rank matrix.

Thus, based on (14) we can approximate A^{-1} as follows:

$$A^{-1} \approx A_0^{-1} + \begin{pmatrix} A_{11}^{-\frac{1}{2}} & 0 \\ 0 & A_{22}^{-\frac{1}{2}} \end{pmatrix} V H V^T \begin{pmatrix} A_{11}^{-\frac{1}{2}} & 0 \\ 0 & A_{22}^{-\frac{1}{2}} \end{pmatrix}, \quad (17)$$

where $V H V^T$ is a rank- k approximation to the matrix J in (15). However, this does not lead to a practical scheme and we saw in Section 3.1 a different approach that is more amenable to efficient computations.

The low-rank property is further analyzed in the next theorem.

Theorem 1. Assume A is SPD matrix factored in the form (12) and let $\Phi = A_{11}^{-\frac{1}{2}} A_{12} A_{22}^{-\frac{1}{2}}$, then we have $0 \leq \sigma_i < 1$ for each singular value σ_i of Φ ($i = 1, 2, \dots, t$), where $t = \min\{p, q\}$.

Proof

Consider the eigenvalues of matrix $\Phi^T \Phi = A_{22}^{-\frac{1}{2}} A_{12}^T A_{11}^{-1} A_{12} A_{22}^{-\frac{1}{2}}$. We have

$$\lambda(\Phi^T \Phi) = \lambda(A_{22}^{-1} A_{12}^T A_{11}^{-1} A_{12}) = \lambda(A_{22}^{-1} (A_{22} - S)) = \lambda(I - A_{22}^{-1} S),$$

where $S = A_{22} - A_{12}^T A_{11}^{-1} A_{12}$ is the Schur complement of A . Because $A_{22}^{-1} S$ is similar to $A_{22}^{-\frac{1}{2}} S A_{22}^{-\frac{1}{2}}$, we obtain

$$\lambda(\Phi^T \Phi) = 1 - \lambda(A_{22}^{-\frac{1}{2}} S A_{22}^{-\frac{1}{2}}). \quad (18)$$

Since A is SPD, it is easy to see that A_{22} and S are also SPD. Thus, $A_{22}^{-\frac{1}{2}} S A_{22}^{-\frac{1}{2}}$ is an SPD matrix with positive eigenvalues. Then, equation (18) implies $\lambda(\Phi^T \Phi) < 1$. Therefore, $0 \leq \lambda(\Phi^T \Phi) < 1$ and hence:

$$\sigma_i = \sqrt{\lambda_i(\Phi^T \Phi)} \in [0, 1), \quad (i = 1, 2, \dots, t).$$

This completes the proof. \square

Theorem 1 shows that 1 is an upper bound for the singular values of Φ in (12). Therefore, one can expect that a few of the largest singular values of Φ will be around 1 causing a fast decay property of $A^{-1} - A_0^{-1}$. Theorem 1 can also be numerically verified by a discretized negative $2D$ Laplacian A with zero Dirichlet boundary condition. Assume this matrix A is of size $2,000 \times 2,000$ and is partitioned into 2 even-sized blocks. All the singular values of Φ and the leading 30 singular values of $A^{-1} - A_0^{-1}$ for this test matrix are plotted in Figure 4. Here, the matrix A is reordered by the multicoloring reordering with 2 colors, where both A_{11} and A_{22} contain three diagonal blocks corresponding to the same color.

As can be seen from Figure 4, while the singular values of Φ are clustered around 1 with a very slow decay rate, the largest singular values of $A^{-1} - A_0^{-1}$ are well separated. For example, the ratio of the second largest singular value of $A^{-1} - A_0^{-1}$ to the largest one is approximately equal to $3.8e-1$ and the ratio of the fifth largest singular value of $A^{-1} - A_0^{-1}$ to the largest one decreases to only $6.3e-2$. This implies that A^{-1} can be approximated by the sum of A_0^{-1} and a low-rank correction term when A is partitioned into a block 2-by-2 form.

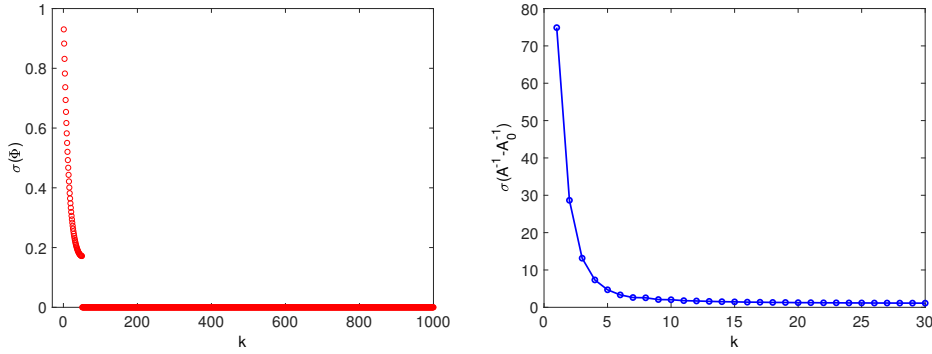


FIGURE 4 Singular values of Φ (left) and the leading 30 singular values of $A^{-1} - A_0^{-1}$ (right) for a 2,000-by-2,000 discretized negative $2D$ Laplacian with the zero Dirichlet boundary condition. Here, the matrix A is reordered by the multicoloring reordering with 2 colors, where both A_{11} and A_{22} contain three diagonal blocks corresponding to the same color.

3.4 | Improving the preconditioner

In order to further reduce the iteration number of the preconditioned Krylov subspace method, we can combine M_i^{-1} with some classical (block) relaxation type methods. This combination is somewhat similar to *post-smoothing* procedures of multigrid-type methods and it will generally provide a boost to the MCLR convergence in practice. Consider the linear system $A|_{\mathbf{I}_i, \mathbf{I}_i} u_i = v_i$, where both u_i and v_i are partitioned conformally into s parts, according to their colors, as (Matlab semi-colon notation used):

$$u_i = [u_{i_1}; u_{i_2}; \dots; u_{i_s}] \quad \text{and} \quad v_i = [v_{i_1}; v_{i_2}; \dots; v_{i_s}]$$

where $j = i_1, i_2, \dots, i_s$ are all the descendent leaf nodes of node i in \mathcal{T} .

For node i , denote

$$\delta_i = [\delta_{i_1}; \delta_{i_2}; \dots; \delta_{i_s}].$$

In a block Jacobi, also called Schwarz, procedure the block coordinate u_j is corrected by a vector δ_j in such a way that: $A|_{\mathbf{I}_j, \mathbf{I}_j} u_i + A|_{\mathbf{I}_j, \mathbf{I}_j} \delta_j = v_j$ ($j = i_1, i_2, \dots, i_s$). This leads to the following equation which is solved for color j :

$$A|_{\mathbf{I}_j, \mathbf{I}_j} \delta_j = v_j - A|_{\mathbf{I}_j, \mathbf{I}_j} u_i. \quad (19)$$

In the block Gauss-Seidel (or multiplicative Schwarz) procedure, the above equation is solved for each color in turn and the solution is immediately updated by changing the j -th block coordinate as $u_j := u_j + \delta_j$. In the experiments we only consider the additive Schwarz – or block-Jacobi iteration – which consists of solving the systems (19) at once for all colors and updating block coordinates simultaneously. This is illustrated in Algorithm 4.

Algorithm 4 Block-Jacobi type correction scheme for solving $A|_{\mathbf{I}_i, \mathbf{I}_i} u_i = v_i$

- 1: **procedure** $u_i = \text{BJ}(A|_{\mathbf{I}_i, \mathbf{I}_i}, u_i, v_i)$
 - 2: **For** $j = i_1, i_2, \dots, i_s$ (leaf nodes in \mathcal{T} which are descendants of node i)
 - 3: Compute residual $f_j = v_j - A|_{\mathbf{I}_j, \mathbf{I}_j} u_i$
 - 4: Solve $L_j U_j \delta_j = f_j$ (By Incomplete LU factorization of M_j)
 - 5: **EndFor**
 - 6: Update solution $u_j := u_j + \delta_j$ for $j = i_1, i_2, \dots, i_s$
 - 7: **end procedure**
-

The combination of M_i^{-1} and Algorithm 4 for solving $A|_{\mathbf{I}_i, \mathbf{I}_i} u_i = v_i$ is summarized in Algorithm 5. This algorithm takes an initial solution u_i obtained from applying M_i^{-1} from (10) to a right-hand side v_i (Steps 2-10) and outputs an improved solution also called u_i (Steps 11-13). The procedure can be repeated a few times and we will call m the number of Block-Jacobi steps

used for the correction. This would have the effect of improving the solution accuracy of $A|_{\mathbf{I}_i, \mathbf{I}_i} x = u_i$ and it would lead to more accurate low-rank correction term $V_i G_i V_i^T$.

Algorithm 5 Nested block-Jacobi correction scheme

```

1: procedure  $u_i = \text{NSBJ}(A|_{\mathbf{I}_i, \mathbf{I}_i}, v_i, i)$ 
2:   if  $i$  is leaf node then
3:     Solve  $L_i U_i u_i = v_i$  (By Incomplete LU factorization of  $M_j$ )
4:   else
5:     Compute  $\mathbf{z} = v_i + V_i(G_i(V_i^T v_i))$ .
6:     Partition  $\mathbf{z}$  into  $\begin{pmatrix} \mathbf{z}_{c_1} \\ \mathbf{z}_{c_2} \end{pmatrix}$  – according to sets  $\mathbf{I}_{c_1}, \mathbf{I}_{c_2}$ 
7:     Compute  $u_{c_1} = \text{NSBJ}(A|_{\mathbf{I}_{c_1}, \mathbf{I}_{c_1}}, \mathbf{z}_{c_1}, c_1)$ .
8:     Compute  $u_{c_2} = \text{NSBJ}(A|_{\mathbf{I}_{c_2}, \mathbf{I}_{c_2}}, \mathbf{z}_{c_2}, c_2)$ .
9:     Stack  $u_{c_1}$  over  $u_{c_2}$  to get:  $u_i = \begin{pmatrix} u_{c_1} \\ u_{c_2} \end{pmatrix}$ 
10:   end if
11:   For  $k = 1 : m$ 
12:      $u_i := \text{BJ}(A|_{\mathbf{I}_i, \mathbf{I}_i}, u_i, v_i)$ 
13:   Endfor
14: end procedure

```

Thus, Algorithm 2 and Algorithm 4 are combined together, resulting in the above nested block-Jacobi correction scheme. This scheme is used in the construction procedure. That is, Algorithm 3 replaces

$$I - A|_{\mathbf{I}_i, \mathbf{I}_i} \begin{pmatrix} M_{c_1}^{-1} \\ M_{c_2}^{-1} \end{pmatrix}$$

by the following operation:

$$v_i - A|_{\mathbf{I}_i, \mathbf{I}_i} \text{NSBJ}(A|_{\mathbf{I}_i, \mathbf{I}_i}, v_i, i), \quad (20)$$

where NSBJ is defined in Algorithm 5. We will refer to this combination of MCLR and Nested block Jacobi type correction scheme as the Multi-Color Low-Rank (MCLR) Preconditioner with *corrections*.

The only difference between Algorithm 2 and Algorithm 5 is that Algorithm 5 adds the block Jacobi correction procedures in Steps 7-8. The following proposition establishes an expression for the action of Algorithm 5 in the ideal case when the solves are exact.

Proposition 1. Consider Algorithm 5 in which u_{c_j} ($j = 1, 2$) returned by Lines 7-8 satisfies $A|_{\mathbf{I}_{c_j}, \mathbf{I}_{c_j}} u_{c_j} = v_{c_j}$ exactly, and let m be the number of Block-Jacobi correction steps in the loop of Lines 11–13 of the Algorithm. Let \tilde{A}_i denote the block diagonal matrix

$$\tilde{A}_i := \begin{pmatrix} A|_{\mathbf{I}_{i_1}, \mathbf{I}_{i_1}} & & & \\ & A|_{\mathbf{I}_{i_2}, \mathbf{I}_{i_2}} & & \\ & & \ddots & \\ & & & A|_{\mathbf{I}_{i_s}, \mathbf{I}_{i_s}} \end{pmatrix}, \quad (21)$$

where i_1, i_2, \dots, i_s are the leaf nodes in \mathcal{T} which are also the descendants of node i . Then we have:

$$\text{NSBJ}(A|_{\mathbf{I}_i, \mathbf{I}_i}, v_i, i) = X_i v_i,$$

where $X_i = (A|_{\mathbf{I}_i, \mathbf{I}_i})^{-1} + (I - \tilde{A}_i^{-1} A|_{\mathbf{I}_i, \mathbf{I}_i})^m \left(\begin{pmatrix} A|_{\mathbf{I}_{c_1}, \mathbf{I}_{c_1}} \\ A|_{\mathbf{I}_{c_2}, \mathbf{I}_{c_2}} \end{pmatrix}^{-1} - A|_{\mathbf{I}_i, \mathbf{I}_i}^{-1} \right)$.

Proof

Since we assume u_{c_j} ($j = 1, 2$) are the exact solutions of $A|_{\mathbf{I}_{c_j}, \mathbf{I}_{c_j}} u = v_{c_j}$, we have $u_i = \left(\begin{array}{c} A|_{\mathbf{I}_{c_1}, \mathbf{I}_{c_1}} \\ A|_{\mathbf{I}_{c_2}, \mathbf{I}_{c_2}} \end{array} \right)^{-1} v_i$ in Step 9. Next, applying m Block-Jacobi correction steps in Lines 11-13 is equivalent to computing the following sequence :

$$\begin{cases} u^{(i)} = (I - \tilde{A}_i^{-1} A|_{\mathbf{I}_i, \mathbf{I}_i}) u^{(i-1)} + \tilde{A}_i^{-1} v_i, \\ i = 1, 2, \dots, m, \end{cases} \quad (22)$$

with $u^{(0)} = u_i$. If u^* is the exact solution of $A|_{\mathbf{I}_i, \mathbf{I}_i} u = v_i$, then we have

$$u^{(i)} - u^* = (I - \tilde{A}_i^{-1} A|_{\mathbf{I}_i, \mathbf{I}_i})(u^{(i-1)} - u^*).$$

This leads to

$$\begin{aligned} u^{(m)} &= u^* + (I - \tilde{A}_i^{-1} A|_{\mathbf{I}_i, \mathbf{I}_i})^m (u^{(0)} - u^*) \\ &= (A|_{\mathbf{I}_i, \mathbf{I}_i})^{-1} v_i + (I - \tilde{A}_i^{-1} A|_{\mathbf{I}_i, \mathbf{I}_i})^m \left(\begin{array}{c} A|_{\mathbf{I}_{c_1}, \mathbf{I}_{c_1}} \\ A|_{\mathbf{I}_{c_2}, \mathbf{I}_{c_2}} \end{array} \right)^{-1} v_i - A|_{\mathbf{I}_i, \mathbf{I}_i}^{-1} v_i \\ &= \left\{ (A|_{\mathbf{I}_i, \mathbf{I}_i})^{-1} + (I - \tilde{A}_i^{-1} A|_{\mathbf{I}_i, \mathbf{I}_i})^m \left(\begin{array}{c} A|_{\mathbf{I}_{c_1}, \mathbf{I}_{c_1}} \\ A|_{\mathbf{I}_{c_2}, \mathbf{I}_{c_2}} \end{array} \right)^{-1} - A|_{\mathbf{I}_i, \mathbf{I}_i}^{-1} \right\} v_i. \end{aligned}$$

This completes the proof. \square

Proposition 1 merely establishes an expression for the operator that performs the action of the operation of Algorithm 5. The purpose is to unravel the m -th power of a certain matrix that is implicit in the process. From this proposition, one can see that the residual matrix $I - A|_{\mathbf{I}_i, \mathbf{I}_i} X_i$ at node i is:

$$\begin{aligned} I - A|_{\mathbf{I}_i, \mathbf{I}_i} X_i &= A|_{\mathbf{I}_i, \mathbf{I}_i} (I - \tilde{A}_i^{-1} A|_{\mathbf{I}_i, \mathbf{I}_i})^m \left(A|_{\mathbf{I}_i, \mathbf{I}_i}^{-1} - \begin{array}{c} A|_{\mathbf{I}_{c_1}, \mathbf{I}_{c_1}} \\ A|_{\mathbf{I}_{c_2}, \mathbf{I}_{c_2}} \end{array} \right)^{-1} \\ &= A|_{\mathbf{I}_i, \mathbf{I}_i} (I - \tilde{A}_i^{-1} A|_{\mathbf{I}_i, \mathbf{I}_i})^m A|_{\mathbf{I}_i, \mathbf{I}_i}^{-1} \left(I - A|_{\mathbf{I}_i, \mathbf{I}_i} \begin{array}{c} A|_{\mathbf{I}_{c_1}, \mathbf{I}_{c_1}} \\ A|_{\mathbf{I}_{c_2}, \mathbf{I}_{c_2}} \end{array} \right)^{-1} \\ &= (I - A|_{\mathbf{I}_i, \mathbf{I}_i} \tilde{A}_i^{-1})^m \left(I - A|_{\mathbf{I}_i, \mathbf{I}_i} \begin{array}{c} A|_{\mathbf{I}_{c_1}, \mathbf{I}_{c_1}} \\ A|_{\mathbf{I}_{c_2}, \mathbf{I}_{c_2}} \end{array} \right)^{-1}. \end{aligned} \quad (23)$$

Equation (23) suggests that matrix $I - A|_{\mathbf{I}_i, \mathbf{I}_i} X_i$ will have a more favorable low-rank representation than matrix $I - A|_{\mathbf{I}_i, \mathbf{I}_i} \left(\begin{array}{c} A|_{\mathbf{I}_{c_1}, \mathbf{I}_{c_1}} \\ A|_{\mathbf{I}_{c_2}, \mathbf{I}_{c_2}} \end{array} \right)^{-1}$ which is the matrix in (5). In the extreme case when $\tilde{A}_i = \begin{array}{c} A|_{\mathbf{I}_{c_1}, \mathbf{I}_{c_1}} \\ A|_{\mathbf{I}_{c_2}, \mathbf{I}_{c_2}} \end{array}$,

$$I - A|_{\mathbf{I}_i, \mathbf{I}_i} X_i = \left(I - A|_{\mathbf{I}_i, \mathbf{I}_i} \begin{array}{c} A|_{\mathbf{I}_{c_1}, \mathbf{I}_{c_1}} \\ A|_{\mathbf{I}_{c_2}, \mathbf{I}_{c_2}} \end{array} \right)^{-1} {}^{m+1}.$$

In this case the eigenvalue decay rate of $I - A|_{\mathbf{I}_i, \mathbf{I}_i} X_i$ is $m + 1$ times faster than that of $I - A|_{\mathbf{I}_i, \mathbf{I}_i} \left(\begin{array}{c} A|_{\mathbf{I}_{c_1}, \mathbf{I}_{c_1}} \\ A|_{\mathbf{I}_{c_2}, \mathbf{I}_{c_2}} \end{array} \right)^{-1}$.

In fact, the eigenvalue distribution of the preconditioned matrix depends highly on how well $I - A|_{\mathbf{I}_i, \mathbf{I}_i} X_i$ is approximated by the low-rank correction term. To conclude this section, we illustrate this property with two examples shown in Figure 5. Here, the matrix A is the non-symmetric facsimile convergence matrix obtained from the SuiteSparse collection⁶. This matrix has dimension $n = 541$ and is indefinite. Figure 5 shows the spectra of the preconditioned matrix when the low-rank correction term uses different ranks with a two level binary tree. Observe that, as expected, the spectrum of the preconditioned matrix with rank 20 is more clustered around 1 than that with rank 5. For this particular problem, preconditioned GMRES^{27, 36} with a low rank correction of rank 20 converges in 34 iterations to reach 6 digits of accuracy while a rank of 5 yields onvergence in 76 iterations.

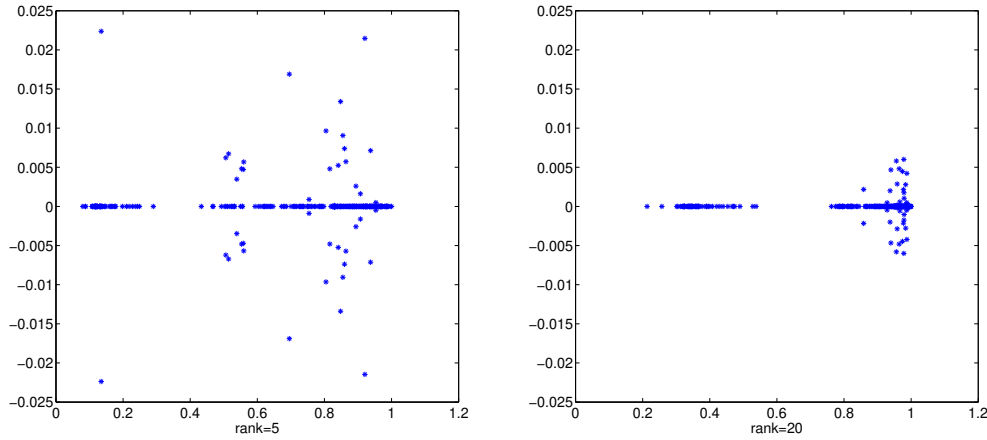


FIGURE 5 Spectra of the preconditioned matrix: rank 5 correction (left) and rank 20 correction (right), where \mathcal{T} has two levels. Here, the number of Block-Jacobi correction steps is taken as $m = 3$.

4 | NUMERICAL EXPERIMENTS

This section discusses a few numerical experiments to demonstrate the robustness and efficiency of the proposed approach. The testing platform consists of two Intel Haswell processors (12 cores each at 2.5 GHz) and 64 GB of memory. The MCLR preconditioner was implemented in C++ and the code was compiled with the Intel C++ compiler using `-O3` optimization.

In the numerical experiments, the original sparse matrices were partitioned by using `PartGraphKway` from the METIS¹⁶ package. The factorization of the diagonal blocks of the reordered matrix and the computation of the low-rank correction terms account for the preconditioner construction time. The times for reordering the matrices are considered as preprocessing, hence they are not reported (often the problems come in partitioned form based on the mesh and the time for coloring the quotient graph is negligible). In actual computations, we chose the right-hand-side vector b such that $Ax = b$, where x is a random vector. In addition, the initial guess was chosen as a zero vector. Also, the stopping criterion is based on the original residual.

Right preconditioning is used for all the tests in this section. For the SPD matrices, the MCLR preconditioner with *corrections* was compared with the incomplete Cholesky factorization with threshold dropping (ICT) and the MSLR preconditioner³¹. The accelerator we used for the MCLR with *corrections*, MSLR and ICT preconditioners is the conjugate gradient (CG) method¹³. For other test matrices, the MCLR preconditioner with *corrections* was compared with the incomplete LU factorization with threshold dropping (ILUT) and the GMCLR preconditioner⁷ and the accelerator used is GMRES^{27, 29}.

The following notation will be used in the remainder of this section where all times are in seconds:

- fill: Fill factor defined as $\frac{\text{nnz}(\text{prec})}{\text{nnz}(A)}$;
- o-t: wall clock time to reorder the matrix;
- p-t: wall clock time to build the preconditioner;
- its: number of iterations of GMRES or CG to reduce the initial residual norm by a factor of 10^6 . Moreover, an “F” indicates that GMRES or CG failed to converge in 300 iterations;
- i-t: wall clock time for the iteration phase. We do not report this time when GMRES or CG fails to converge in 300 iterations.
- t-t: total wall clock time, i.e., sum of the time to build the preconditioner and the iteration time.
- n_d : number of subdomains used to partition the adjacency graph of A ;
- n_c : number of colors;
- r_k : rank used in the low-rank corrections;

- m : number of steps of Block-Jacobi type correction scheme (Algorithm 4).

In all the tests, applications of the MCLR with *corrections*, MSLR and GMSLR preconditioners were parallelized using OpenMP²⁴ where the number of threads used is equal the number of cores, which is 24.

4.1 | 3-D model problem

We first consider the following convection-diffusion equation:

$$\begin{aligned} -\Delta u - \alpha \cdot \nabla u - \beta u &= f \text{ in } \Omega, \\ u &= 0 \text{ on } \partial\Omega, \end{aligned} \quad (24)$$

where $\Omega = (0, 1)^3$ and $\alpha \in \mathbb{R}^3$. The discretization of this equation is via centered finite differences with the standard 7-point stencil in three dimensions. If $\alpha = (0, 0, 0)$, then (24) is a symmetric problem. Otherwise, it is nonsymmetric.

4.1.1 | Effect of n_d

In this subsection, we study the effect of the number of subdomains n_d on the performance of the MCLR preconditioner with *corrections*. First, we discretized (24) on a 50^3 grid with $\alpha = [.05, .05, .05]$ and $\beta > 0$ in order to make the problem indefinite. In the case where $\beta > 0$, we shift the discretized convection-diffusion operator by sI , where $s = h^2\beta$ for mesh size h . The rank used in the low-rank correction was set to $r_k = 5$ and m was fixed at 5.

As we can see from Table 1, as the value of n_d increases from 10 to 90, the fill-factor for the ILU decompositions decreases monotonously while the fill factor from the low-rank correction term increases. This is because a large number of n_d usually leads to a tree \mathcal{T} with more levels and thus causes more low-rank correction terms. Meanwhile, a large n_d results in a reduced block size for each diagonal block and this reduces both the computational costs and storage for the ILU factorizations. These trade-offs between the two fill factors can be visualized in Figure 6. Based on various tests, we find that $n_d = 50$ is often optimal for this problem. Therefore, we set n_d to $n_d = 50$ in the remaining experiments.

TABLE 1 The fill factors and iteration counts for solving (24) discretized on a 50^3 grid with $\alpha = [.05, .05, .05]$ and $s = 0.1$ by the GMRES-MCLR with *corrections* method. Here, the rank for the low-rank correction matrices was fixed at 5, m was fixed at 5 and the threshold used in the incomplete LU factorization was taken as 10^{-2} .

n_d	fill (ILU)	fill (Low-rank)	fill (total)	its	p-t	i-t
10	2.83	2.04	4.87	73	0.80	3.52
30	2.72	2.15	4.87	86	0.82	3.60
50	2.66	2.16	4.86	83	0.81	3.40
70	2.61	2.18	4.79	85	0.84	3.58
90	2.58	2.20	4.78	82	0.86	3.56

4.1.2 | Effect of r_k in the low-rank corrections

Next we study the performance of the MCLR preconditioner with *corrections* when the rank used in the low-rank correction varies. To illustrate this, we solve the same test problem as in the previous section with different values for the rank r_k . As is seen in Table 2 the ILU fill factor remains constant since we kept n_d fixed. From Table 2, one can also see that both the iteration time and the total time for $r_k = 0$ are longer than those of the case $r_k > 0$. Hence, the low-rank correction can improve the performance of the MCLR preconditioner. On the other hand, the iteration number only decreases slightly when the rank increases from 5 to 45 while the low-rank fill factor increases dramatically during this process. This implies that for this problem there is no need to use a large r_k in order to achieve a good performance. In our next tests we will often set r_k to some small values (i.e., $r_k \leq 5$).

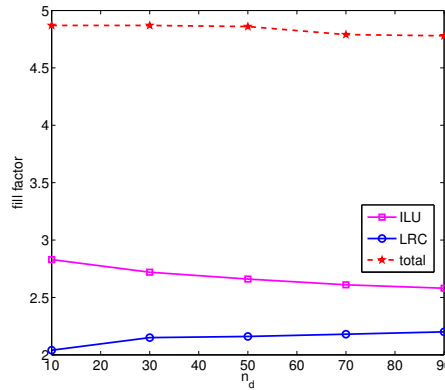


FIGURE 6 The fill factors from ILU and low-rank correction with respect to different n_d for the test problem in Table 1.

TABLE 2 The fill factors and iteration counts for solving (24) with $\alpha = [.05, .05, .05]$ and $s = 0.1$ on a $50 \times 50 \times 50$ grid with the GMRES-MCLR with *corrections* method. Here, n_d was taken as 50, m was fixed at 5 and the threshold used in the incomplete LU factorization was fixed to 10^{-2} .

r_k	fill (ILU)	fill (Low-rank)	fill (total)	its	p-t	i-t
0	2.66	0.00	2.66	91	0.76	4.45
5	2.66	2.16	4.86	83	0.81	3.50
15	2.66	6.54	9.30	80	1.04	3.92
25	2.66	10.90	13.56	75	1.90	5.01
35	2.66	15.26	17.92	71	2.79	6.60
45	2.66	20.01	22.67	68	8.54	7.98

4.1.3 | Effect of the threshold

It is also helpful to understand the effect of the threshold used in ILU factorizations on the quality of the preconditioner. A smaller threshold leads to more accurate approximations to the diagonal blocks, and so we expect that the iteration number should decrease and the ILU fill factors should increase as the threshold decreases. This is verified with the results in Table 3 when the threshold is reduced from 0.1 to 0.0005. Note that similarly to what was seen with the rank, the higher cost of using a lower tolerance is not counter-balanced by the small decrease in the number of iterations to converge.

TABLE 3 The fill factors and iteration counts for solving (24) with $\alpha = [.05, .05, .05]$ and $s = 0.1$ discretized on a 50^3 grid with the GMRES-MCLR with *corrections* method. Here, n_d was taken as 50, m was fixed at 5 and the rank for the low-rank correction was set to 5.

threshold	fill (ILU)	fill (Low-rank)	fill (total)	its	t-t
.1	2.66	2.18	4.84	88	4.53
.05	2.70	2.18	4.88	86	4.51
.01	2.76	2.18	4.94	83	4.49
.005	2.81	2.18	4.99	80	4.50
.001	2.95	2.18	5.13	78	4.55
.0005	2.96	2.18	5.14	77	4.58

4.1.4 | Effect of the number of correction steps

The next experiments illustrate the effect of the number of correction steps m on the performance of the MCLR preconditioner with *corrections*. Here, the test matrix is obtained by discretizing (24) with $\beta = 0.1$ on a 50^3 grid. This is an indefinite problem. There are 44 negative eigenvalues for this specific problem. We fixed r_k at 5 in the experiments. The left subfigure of Figure 7, shows a steady improvement in the iteration number when m increases from 0 to 10, indicating that the Block-Jacobi correction is quite effective at improving convergence. Of course, the time to build the preconditioner increases with higher values of m and the time to apply will decrease. The right subfigure of Figure 7 shows this but it also shows that the total iteration time first decreases as m increases from 0 to 5, and then increases when m increases from 5 to 10. Hence, although Algorithm 4 can reduce the iteration count, too many correction steps (large m) may result in a higher CPU time. For this test matrix, $m = 5$ is the optimal number of correction steps, resulting in the smallest total time (i.e., 1.56 seconds to build preconditioner and 4.78 seconds to converge) for this test.

4.1.5 | Effect of the number of threads

Table 4 shows the fill factors, iteration counts and iteration numbers for solving (24) with $\alpha = [.05, .05, .05]$ and $s = 0.1$ discretized on a 50^3 grid using different number of threads. The total wall clock time decreases as the number of threads increases from 4 to 24. This table shows that the MCLR with *corrections* preconditioner scales well.

TABLE 4 The fill factors and iteration counts for solving (24) with $\alpha = [.05, .05, .05]$ and $s = 0.1$ discretized on a 50^3 grid with the GMRES-MCLR with *corrections* method. Here, n_d was taken as 50, m was fixed at 5 and the rank for the low-rank correction was set to 5.

number of threads	fill (ILU)	fill (Low-rank)	fill (total)	its	t-t
1	2.66	2.16	4.82	83	89.06
4	2.66	2.16	4.82	83	25.43
8	2.66	2.16	4.82	83	13.40
16	2.66	2.16	4.82	83	6.89
24	2.66	2.16	4.82	83	4.43

4.2 | Model symmetric problems

We consider the symmetric case (i.e., $\alpha = (0, 0, 0)$) for test (24) in this section. Here, we solve (24) with $\beta = 0$ (SPD case) and $\beta > 0$ (indefinite case). For the indefinite case, we shifted the discretized Laplacian by sI , where $s = h^2\beta$ for mesh size h . The numerical results are reported in Table 5. We chose $r_k = 2$ for the SPD case and $r_k = 5$ for the indefinite case. The parameters of MSLR and GMSLR preconditioners were chosen such that they yield roughly the same fill-factors as those of MCLR. Table 5 shows that the CPU time to construct a MCLR preconditioner with corrections is the least of these three preconditioners. In addition, the MCLR preconditioner with *corrections* requires the smallest number of iterations and the least CPU time to converge for the positive definite problems as well as for the indefinite problems. Moreover, the same table also shows that the GMSLR and ILUT preconditioned GMRES methods have difficulties converging when dealing with (24) on the 128^3 grid with $\beta > 0$. This test matrix has 217 negative eigenvalues and in this sense it is more indefinite than the other test matrices.

4.3 | General sparse linear systems

We now test MCLR with *corrections* on various general matrices obtained from the SuiteSparse Matrix Collection⁶. The set of test problems from SuiteSparse are picked from the tests in paper^{7, 17, 31}. Here, the matrices `cf1d1`, `cf1d2` and `thermal2` are picked from¹⁷. In addition, since the largest nonsymmetric problem tested in¹⁷ only has the size of 71, 505, we chose three additional large nonsymmetric matrices `Atmosmodd` (with size 1, 270, 432), `Atmosmodl` (with size 1, 489, 752) and `Transport` (with size 1, 602, 111) from^{7, 31}. Table 6 presents a short description.

TABLE 5 Comparison among MCLR with *corrections*, ICT/ILUT and MSLR/GMSLR preconditioners for solving symmetric positive definite/indefinite linear systems from the 3-D PDEs (24) with the CG/GMRES methods, where m was fixed at 5.

Mesh	s	MCLR								ICT/ILUT				MSLR/GMSLR							
		n_c	lev	r_k	fill	its	o-t	p-t	i-t	fill	its	p-t	i-t	lev	r_k	fill	its	o-t	p-t	i-t	
32^3	0.00	8	4	2	3.09	6	.02	.11	.19	3.05	16	.15	.48	7	32	3.12	31	.03	.14	.42	
64^3		8	4	2	3.53	10	.04	1.02	1.50	3.53	25	1.08	3.21	10	32	3.56	60	.05	1.03	2.55	
128^3		9	5	2	3.53	15	.09	3.30	4.69	3.01	40	8.21	10.79	13	32	4.00	116	.11	4.95	7.10	
32^3	0.04	8	4	5	3.31	9	.03	.09	.25	3.32	29	.28	.44	7	16	3.33	28	.03	.10	.30	
64^3		8	4	5	4.78	47	.04	1.23	4.70	4.67	78	1.59	8.09	10	25	4.76	202	.05	1.24	7.10	
128^3		9	5	5	3.39	216	.09	5.00	11.99	3.42	F	12.01	-	13	64	3.40	F	.11	8.02	-	

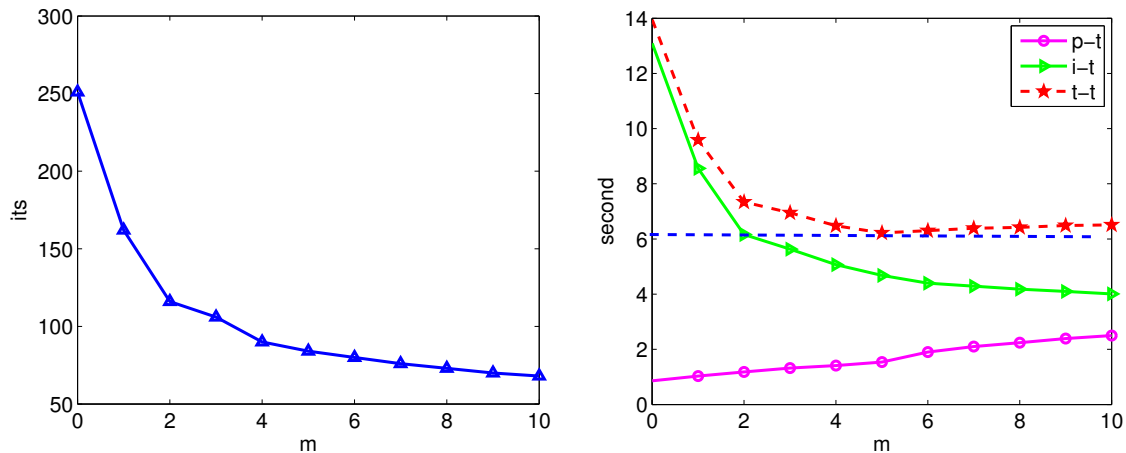


FIGURE 7 Left: the iteration numbers with respect to different m . Right: the iteration time and the preconditioner construction time with respect to different m . Here, the grid is fixed to 50^3 .

Numerical results are presented in Table 7. As can be seen, for these 15 problems, MCLR with *corrections* outperforms the ICT/ILUT and MSLR/GMSLR preconditioners: it needs fewest iterations and least time to converge. Observe from the table that Krylov methods preconditioned with MCLR with *corrections* converge for all the test problems given in Table 6. In contrast, ICT fails to converge for *cf d1* and *cf d2*, ILUT fails to converge to solve *xenon1* and *xenon2* and MSLR can not converge for *cf d1*. In these experiments, m was set to $m = 5$.

5 | CONCLUSIONS

We described a parallelizable preconditioner for solving general sparse linear systems, named MCLR, that is based on a combination of algebraic domain-decomposition, domain multicoloring, and recursive multilevel low-rank corrections. METIS is used to partition the original matrix and then the quotient graph of the partitions is multicolored leading to a hierarchical block structure for the reordered matrix. The construction procedure and the solve stage can be parallelized based on both the binary tree structure and the block diagonal structure associated with each leaf node. The MCLR preconditioner can be applied to solve

TABLE 6 Names, orders (N), numbers of nonzeros (nnz) and short descriptions of the test matrices.

Matrix	Order	nnz	symmetric	Description
cf1	70,656	1,825,580	yes	CFD problem
cf2	123,440	3,085,406	yes	CFD problem
ecology1	1,000,000	4,996,000	yes	Landscape ecology problem
thermal2	1,228,045	8,580,313	yes	Thermal problem
xenon1	48,600	1,181,120	no	Materials Problem
xenon2	157,464	3,866,688	no	Materials Problem
ML_Laplace	377,002	27,582,698	no	meshless local Petrov-Galerkin method
Atmosmodd	1,270,432	8,814,880	no	Atmospheric model
Atmosmodl	1,489,752	10,319,760	no	Atmospheric model
Transport	1,602,111	23,500,731	no	CFD problem

TABLE 7 Comparison among MCLR with *corrections*, ICT/ILUT and MSLR/GMSLR preconditioners for solving general sparse linear systems along with CG or GMRES, where *m* was fixed at 5.

Matrix	MCLR							ICT/ILUT				MSLR/GMSLR							
	n_c	lev	r_k	fill	its	o-t	p-t	i-t	fill	its	p-t	i-t	lev	r_k	fill	its	o-t	p-t	i-t
cf1	7	4	2	1.70	82	.01	1.20	2.19	1.76	F	.63	-	7	64	1.74	F	.01	.94	-
cf2	6	4	2	1.89	95	.02	.98	4.06	1.89	F	.99	-	8	80	1.86	108	.03	1.56	5.34
ecology1	5	4	2	2.60	13	.05	.99	2.30	2.59	25	1.01	4.08	8	64	2.71	60	.08	.90	3.42
thermal2	5	4	2	4.40	39	.09	4.60	6.76	4.44	115	6.03	14.59	8	64	4.54	65	.10	5.34	9.36
xenon1	7	4	5	1.31	42	.01	.59	3.90	1.37	F	3.62	-	3	32	1.37	156	.03	0.63	6.54
xenon2	8	4	5	2.80	57	.03	1.40	4.51	3.01	F	5.80	-	10	64	2.78	269	.05	1.66	7.62
ML_Laplace	5	4	5	1.01	72	.07	2.10	1.23	1.05	167	1.80	3.22	6	64	1.06	150	.09	2.20	2.03
Atmosmodd	8	4	5	3.23	18	.09	2.03	5.02	3.50	44	3.43	11.01	10	4	3.21	30	.10	3.34	7.34
Atmosmodl	6	4	5	3.45	8	.09	2.87	4.21	3.51	26	3.76	10.00	11	4	3.55	15	.12	3.29	7.59
Transport	5	4	5	2.12	43	.12	4.50	9.02	2.15	92	8.28	18.33	6	4	2.10	79	.14	6.89	12.09

both symmetric and nonsymmetric problems and is highly parallelizable due to the block diagonal structure obtained from the coloring.

One of the biggest advantages of using the MCLR preconditioner is its robustness when solving highly indefinite problems. As the numerical experiments show, the MCLR preconditioner with a form of Block-Jacobi post-smoothing, outperforms MSLR/GMSLR and is far more robust than the standard ICT or ILUT techniques. A small number of Block-Jacobi correction steps is usually enough to boost the speed of convergence and decrease the overall time required by MCLR. Because of these appealing

features we plan to extend MCLR to complex (non-Hermitian) systems and eigenvalue problems^{21, 32}. We will also explore other low-rank correction techniques to try to further reduce the construction time of the preconditioner.

ACKNOWLEDGMENTS

We would like to thank anonymous referees for their useful suggestions which led to substantial improvements of the original version of this paper. This work does not have any conflicts of interest.

References

1. Aminfar, A., Ambikasaran, S., Darve, E.: A fast block low-rank dense solver with applications to finite-element matrices. *J. Comput. Phys.* 304, 170–188 (2016)
2. Bebendorf, M.: Hierarchical Matrices: A Means to Efficiently Solve Elliptic Boundary Value Problems. Lecture Notes in Computational Science and Engineering, Springer Berlin Heidelberg, 2008.
3. Borne, S. L., Grasedyck, L.: \mathcal{H} -matrix preconditioners in convection-dominated problems. *SIAM J. Matrix Anal. Appl.* 27(4), 1172–1183 (2006)
4. Cai, D., Chow, E., Erlandson, L. Saad, Y., Xi, Y.: SMASH: Structured Matrix Approximation by Separation and Hierarchy. *Numer. Linear Algebra Appl.* 26(5), (2018)
5. Chow, E., Saad, Y.: Approximate inverse preconditioners via sparse-sparse iterations. *SIAM J. Sci. Comput.* 19(3), 995–1023 (1998)
6. Davis, T. A., Hu, Y.: The University of Florida Sparse Matrix Collection. *ACM Trans. Math. Software* 38 (2011)
7. Dillon, G., Kalantzis, V., Xi, Y., Saad, Y.: A hierarchical low-rank Schur complement preconditioner for in definite linear systems. *SIAM J. Sci. Comput.* 40(4), A2234–A2252 (2018)
8. Gillman, A., Young, P. M., Martinsson, P. G.: A direct solver with $O(N)$ complexity for integral equations on one-dimensional domains. *Front. Math. China* 7(2), 1673–3576 (2012)
9. Golub, G. H., Van Loan, C. F.: *Matrix Computations*. 4th edition, Johns Hopkins University Press, Baltimore, MD, 4th ed., 2013.
10. Grote, M., Huckle, T.: Parallel preconditioning with sparse approximate inverses. *SIAM J. Sci. Comput.* 18(3), 838–853 (1997)
11. Hackbusch, W., Börm, S.: \mathcal{H}^2 -matrix approximation of integral operators by interpolation. *Appl. Numer. Math.* 1(43), 2266–2293 (2002)
12. Hénon, P., Saad, Y.: A parallel multistage ILU factorization based on a hierarchical graph decomposition. *SIAM J. Sci. Comput.* 28(6), 2266–2293 (2006)
13. Hestenes, M. R., Stiefel, E. L.: Methods of conjugate gradients for solving linear systems. *J. Research Nat. Bur. Standards* 49, 409–436 (1952)
14. Ho, K. L., Greengard, L.: A Fast Direct Solver for Structured Linear Systems by Recursive Skeletonization *SIAM J. Sci. Comput.* 34(5), A2507–A2532 (2012)
15. Ho, K. L., Ying, L.: Hierarchical Interpolative Factorization for Elliptic Operators: Integral Equations. *Commun. Pur. Appl. Math.* 69(7), 1097-0312 (2016)
16. Karypis, G., Kumar, V.: A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM J. Sci. Comput.* 20, 359–392(1998)
17. Li, R., Saad, Y.: Divide and conquer low-rank preconditioners for symmetric matrices. *SIAM J. Sci. Comput.* 35(4), A2069–A2095 (2013)
18. Li, R., Saad, Y.: GPU-accelerated preconditioned iterative linear solvers. *J. Supercomput.* 63(2), 443–466 (2013)
19. Li, R., Xi, Y., Saad, Y.: Schur complement-based domain decomposition preconditioners with low-rank corrections. *Numer. Linear Algebra Appl.* 23(4), 706–7295 (2016)
20. Liu, X., Xia, J., de Hoop, M. V.: Parallel Randomized and Matrix-Free Direct Solvers for Large Structured Dense Linear Systems. *SIAM J. Sci. Comput.* 38(5), S508–S538 (2016)

21. Liu, X., Xi, Y., Saad, Y., de Hoop, M. V.: Solving the 3D high-frequency Helmholtz equation using contour integration and polynomial preconditioning. *SIAM J. Matrix Anal. Appl.* 41(1), pp. 58–82 (2020).
22. Liu, X., Xi, Y., de Hoop, M. V.: Fast factorization update for general elliptic equations under multiple coefficient updates, Purdue CCAM Report CCAM-2018-3.
23. Liu, X., Xia, J., and de Hoop, M. V. : A fast direct elliptic solver via interconnected hierarchical structures, Purdue CCAM Report CCAM-2019-2.
24. OpenMP Architecture Review Board: OpenMP Application Program Interface. Version 3.1, 2011
25. Rouet, F. H., Li, X. S., Ghysels, P., Napov, A.: A distributed-memory package for dense hierarchically semi-separable matrix computations using randomization. *ACM Trans. Math. Softw.* 42(4), (2016)
26. Saad, Y.: *Iterative Methods for Sparse Linear Systems*, 2nd edition. SIAM Philadelphia, PA, 2003
27. Saad, Y.: A flexible inner-outer preconditioned GMRES algorithm. *SIAM J. Sci. Comput.* 14, 461–469 (1993)
28. Saad, Y.: ILUT: a dual threshold incomplete ILU factorization. *Numer. Linear Algebra Appl.* 4(1), 387–402 (1994)
29. Saad, Y., Schultz, M. H.: GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems. *SIAM J. Sci. Statist. Comput.* 7, 856–869 (1986)
30. Saad, Y., Zhang, J.: BILUM: Block versions of multielimination and multilevel ILU preconditioner for general sparse linear systems. *SIAM J. Sci. Comput.* 20(6), 2103–2121 (1999)
31. Xi, Y., Li, R., Saad, Y.: An Algebraic Multilevel Preconditioner with Low-Rank Corrections for Sparse Symmetric Matrices. *SIAM J. Matrix Anal. Appl.* 37(1), 235–259 (2016)
32. Xi, Y., Saad, Y.: A rational function preconditioner for indefinite sparse linear systems. *SIAM J. Sci. Comput.* 39(3), A1145-A1167 (2017)
33. Xia, J.: Efficient Structured Multifrontal Factorization for General Large Sparse Matrices. *SIAM J. Sci. Comput.* 35(2), A832–A860 (2013)
34. Xia, J., Chandrasekaran, S., Gu, M., Li, X. S.: Superfast Multifrontal Method for Large Structured Linear Systems of Equations. *SIAM J. Matrix Anal. Appl.* 31(3), 1382-1411 (2010)
35. Xia, J. and Xin, Z.: Effective and robust preconditioning of general SPD matrices via structured incomplete factorization. *SIAM J. Matrix Anal. Appl.* 38, 1298-1322 (2017)
36. Ye, X., Xi, Y., Saad, Y.: Preconditioning via GMRES in Polynomial Space. Preprint (2019)

