MULTI-COLOR LOW-RANK PRECONDITIONER FOR GENERAL SPARSE LINEAR SYSTEMS *

QINGQING ZHENG †, YUANZHE XI ‡, AND YOUSEF SAAD §

Abstract. 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 quotient graph of 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. We show that the difference between the inverse of a general block 2-by-2 SPD matrix and that of its block diagonal part can be well approximated by a low-rank matrix. This property and the block diagonal structure of the reordered matrix are exploited to develop 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.

Key words. Low-rank approximation; parallel preconditioners; domain decomposition; recursive multilevel methods; Krylov subspace methods.

AMS subject classifications. 65F08, 65F10, 65F50, 65N55, 65Y05

1. Introduction.

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

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

where $A \in \mathbb{R}^{n\times n}$ and $b \in \mathbb{R}^n$, by iterative methods. Preconditioned Krylov subspace methods can solve equation (1.1) 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]. 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 GPUs [20] or Intel Xeon Phi processors.

Sparse approximate inverse preconditioners [11, 6] 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

*This work was supported by NSF under grant NSF/DMS-1521573 and by the Minnesota Supercomputing Institute

†School of Mathematical Science, Xiamen University. {zqq.stu.xmu@hotmail.com}

‡Department of Mathematics, Emory University. {yxi26@emory.edu}

§Computer Science & Engineering, University of Minnesota, Twin Cities. {saad@umn.edu}

complement low-rank (SLR) preconditioner [21] is based on the domain decomposition framework. The Multilevel Schur complement Low-Rank (MSLR) preconditioner [29] utilizes the multilevel Hierarchical Interface Decomposition (HID) ordering [14] to further improve the scalibility of the SLR preconditioner. Generalized Multilevel Schur complement Low-Rank (GMSLR) preconditioner [8] 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) preconditioner [19] 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 [4, 2], HOLDR-matrix [1], $\mathcal{H}^2$-matrix [13, 12] and hierarchically semiseparable (HSS) matrices [16, 5, 17, 9, 23, 31]. 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 [33, 32, 24, 34].

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 MLR preconditioners which can only be applied to symmetric problems, the MCLR preconditioner can handle both the symmetric and the nonsymmetric cases. In addition, a much smaller rank is usually required to reach the same approximation accuracy for the MCLR 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 precoditioner 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 collection [7]. Finally, concluding remarks are presented in Section 5.

2. Multicoloring 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

parallelism [26]. 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 (1.1), 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 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, Adj(i) denotes the set of the vertices that are adjacent to vertex 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: Construct the quotient graph $\mathcal{G}$ of $\mathcal{A}$ based on $\{D_i\}$
3: **for** $i = 1, 2, \cdots, n_0$ **do**
4: Set Color(i)=0
5: **end for**
6: **for** $i = 1, 2, \cdots, n_0$ **do**
7: Set Color(i)=min$\{k > 0 | k \neq \text{Color}(j),\ \forall j \in \text{Adj}(i)\}$
8: **end for**

Step 1 in Algorithm 1 can be implemented by any graph partitioner such as METIS [18]. Step 2 constructs the quotient graph $\mathcal{G}$ of the adjacency graph $\mathcal{A}$ of A. Here the quotient graph is with respect to the partition in step 1, i.e., each vertex i of $\mathcal{G}$ represents a set D_i in the partition. Vertex i is adjacent to vertex j in the quotient graph $\mathcal{G}$ if some vertex in D_i is adjacent to at least one vertex in D_j in the original graph $\mathcal{A}$. Steps 3-8 apply the greedy multicoloring algorithm on $\mathcal{G}$. Figure 2.1 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'.

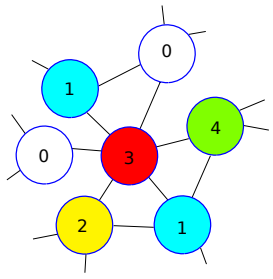

FIG. 2.1. *Coloring 7 vertices with 5 colors on a graph.*

The matrix A in (1.1) 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

$$(P^T AP)(P^T x) = P^T b, \tag{2.1}$$

where P denotes the permutation matrix obtained from Algorithm 1. Because the original system (1.1) disappears from the picture in the remainder of the paper, we will still denote by A the permuted matrix $P^T AP$ 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 2.2 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.

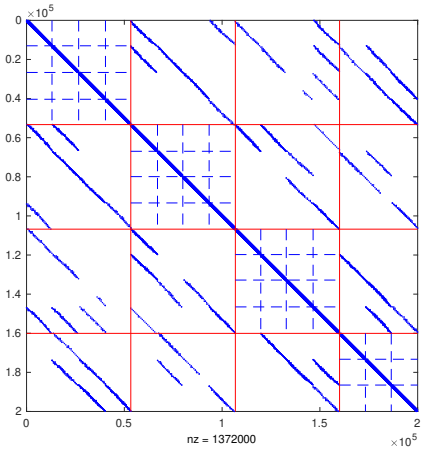

FIG. 2.2. *Nonzero pattern of a reordered discretized negative 3D Laplacian with the zero Dirichlet boundary condition using Algorithm 1 with* 4 *colors and* $n_d = 15$.

Another way to find a multicoloring of the quotient graph is to exploit block (or group)-independent sets, as was done in ARMS for example [26, 28]. An independent set of vertices is characterized by the property that any two nodes in the set are not adjacent. We can find an independent set by a greedy algorithm similar in spirit to the multicoloring algorithm described above. The complement to the independent set is then considered and it is in turn separated into an independent set and a complement. This process can be repeated a few times until a sufficient number of independent sets are found. Figure 2.3 illustrates the process of finding three diagonal blocks by invoking the block independent set algorithm three times.

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.

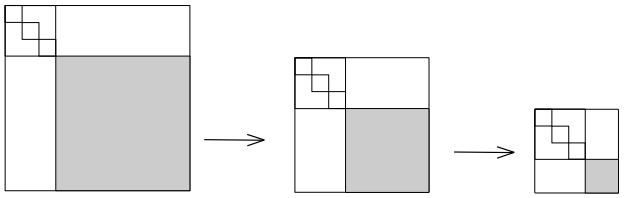

FIG. 2.3. *Illustration of a block independent set based multicoloring procedure where* 3 *diagonal blocks are labeled with the same color each time.*

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:

$$A = \begin{pmatrix} A_{11} & A_{12} \\ A_{12}^T & A_{22} \end{pmatrix}, \quad A_0 = \begin{pmatrix} A_{11} & \\ & A_{22} \end{pmatrix}, \tag{3.1}$$

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 [21, 29, 8] 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

$$I - AA_0^{-1} = A(A^{-1} - A_0^{-1}). \tag{3.2}$$

The matrix $I - AA_0^{-1}$ inherits the low-rank 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:

$$I - AA_0^{-1} \approx VHV^T, \tag{3.3}$$

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:

$$\begin{aligned} A^{-1} &\approx A_0^{-1}(I - VHV^T)^{-1} && (3.4)\\ &= A_0^{-1}(I + VH(I - H)^{-1}V^T) \\ &= A_0^{-1}(I + V[(I - H)^{-1} - I]V^T) \\ &= A_0^{-1}(I + VGV^T), && (3.5) \end{aligned}$$

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 (3.5) 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 (3.5) to obtain a hierarchical approximation to A^{-1}. This approximation can be used as a preconditioner for solving linear systems associated with 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 3.1 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 ith diagonal block in the reordered matrix. It is associated with a color. Recall that a color consists of subdomains that are not coupled, leading to a block that is block diagonal. Figure 2.2 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

$$\mathbf{I}_i = \mathbf{I}_{c_1} \cup \mathbf{I}_{c_2}. \tag{3.6}$$

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 $\mathbf{I}_i$.

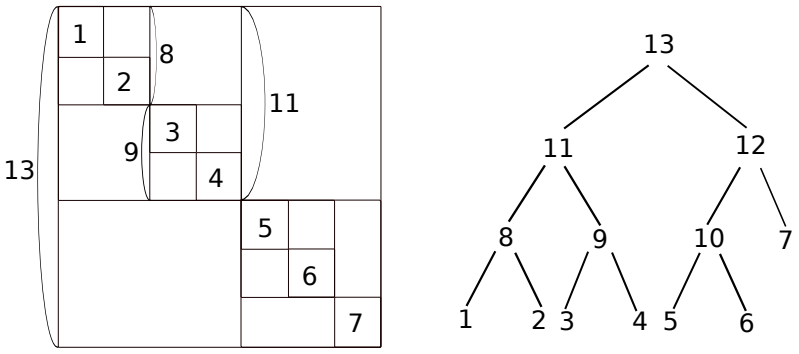

FIG. 3.1. *The block diagonal structure of a reordered matrix using Algorithm 1 with 7 colors (left) and its associated full binary tree $\mathcal{T}$ (right). Each leaf node of $\mathcal{T}$ corresponds to one diagonal block in the reordered matrix.*

The procedure to build the preconditioner follows a bottom-up traversal of $\mathcal{T}$. The basic idea is to recursively utilize (3.5) 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 (3.5), we define :

$$M_i^{-1} = \begin{pmatrix} M_{c_1}^{-1} & \\ & M_{c_2}^{-1} \end{pmatrix} (I + V_i G_i V_i^T), \tag{3.7}$$

where $V_i G_i V_i^T$ is a rank-k matrix. When computing $V_i G_i V_i^T$ based on (3.3) and (3.5), 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 $\mathbf{x} = M_i^{-1}\mathbf{r}$

procedure `x = RecSolv(r,i)`
  if i is leaf node **then**
    Solve $L_i U_i \mathbf{x} = \mathbf{r}$
  else
    Compute $\mathbf{z} = \mathbf{r} + V_i(G_i(V_i^T \mathbf{r}))$.
    Partition $\mathbf{z}$ into $\begin{pmatrix} \mathbf{z}_{c_1} \\ \mathbf{z}_{c_2} \end{pmatrix}$ – according to sets $\mathbf{I}_{c_1}, \mathbf{I}_{c_2}$
    Solve $\mathbf{y}_1 = \text{RecSolv}(\mathbf{z}_{c_1}, c_1)$.
    Solve $\mathbf{y}_2 = \text{RecSolv}(\mathbf{z}_{c_2}, c_2)$.
    Stack $\mathbf{x} = \begin{pmatrix} \mathbf{y}_1 \\ \mathbf{y}_2 \end{pmatrix}$.
  end if
end procedure

In summary, the computed M_i^{-1} satisfies the following relation

$$M_i^{-1} = \begin{cases} \begin{pmatrix} M_{c_1}^{-1} & \\ & M_{c_2}^{-1} \end{pmatrix} (I + V_i G_i V_i^T), & \text{if } i \text{ is non-leaf node,} \\ U_i^{-1} L_i^{-1}, & \text{if } i \text{ is leaf node.} \end{cases} \tag{3.8}$$

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

$$A^{-1} \approx M_{\texttt{root}}^{-1}. \tag{3.9}$$

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

Apply Algorithm 1 to reorder the coefficient matrix
Construct a binary tree $\mathcal{T}$ with L levels
for level l from $L-1$ to 0 **do**
  for node i on level l **do**
    if i is leaf node **then**
      Compute incomplete LU factorization $A|_{\mathbf{I}_i,\mathbf{I}_i} \approx L_i U_i$ and set $M_i = L_i U_i$
    else
      Compute k eigenpairs by the Arnoldi procedure

$$[V_i, H_i] = \text{Arnoldi}\left(I - A|_{\mathbf{I}_i,\mathbf{I}_i} \begin{pmatrix} M_{c_1}^{-1} & \\ & M_{c_2}^{-1} \end{pmatrix}, k\right)$$

      Compute $G_i = (I - H_i)^{-1} - I$
      Set $M_i^{-1} = \begin{pmatrix} M_{c_1}^{-1} & \\ & M_{c_2}^{-1} \end{pmatrix} (I + V_i G_i V_i^T)$
    end if
  end for
end for

This $M_{\texttt{root}}^{-1}$ can be used as a preconditioner for solving (2.1) and its construction procedure is summarized in Algorithm 3.

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 approximation exploited in Section 3.1. Consider Equation (3.10). Since A is SPD, A_{11} and A_{22} are also SPD and admit a square root factorization. Hence, we have

$$A = A_0^{1/2} \begin{pmatrix} I & A_{11}^{-\frac{1}{2}} A_{12} A_{22}^{-\frac{1}{2}} \\ A_{22}^{-\frac{1}{2}} A_{12}^T A_{11}^{-\frac{1}{2}} & I \end{pmatrix} A_0^{1/2} = A_0^{1/2} \begin{pmatrix} I & \Phi \\ \Phi^T & I \end{pmatrix} A_0^{1/2}, \tag{3.10}$$

where $\Phi = A_{11}^{-\frac{1}{2}} A_{12} A_{22}^{-\frac{1}{2}} \in \mathbb{R}^{p \times q}$. From this we can write:

$$A^{-1} - A_0^{-1} = A_0^{-1/2} \left[\begin{pmatrix} I & \Phi \\ \Phi^T & I \end{pmatrix}^{-1} - \begin{pmatrix} I & \\ & I \end{pmatrix} \right] A_0^{-1/2}. \tag{3.11}$$

We would like to show that the matrix

$$J = \begin{pmatrix} I & \Phi \\ \Phi^T & I \end{pmatrix}^{-1} - \begin{pmatrix} I & \\ & I \end{pmatrix} \tag{3.12}$$

can be well approximated by low-rank matrices. It is well known [10, 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 $\sigma_i, i = 1, \cdots, r$, are the r singular values of Φ and the eigenvalue zero repeated $n - 2r$ times. Hence, the matrix $\begin{pmatrix} I & \Phi \\ \Phi^T & I \end{pmatrix}$ has $2r$ eigenvalues of the form $1 \pm \sigma_i$ for

$i = 1, 2, \ldots, r$, and an eigenvalue equal to one which is of multiplicity $n - 2r$. This is stated as a lemma.

LEMMA 3.1. *Let $\Phi \in \mathbb{R}^{p\times q}$ and $r = rank(\Phi)$, then the eigenvalues of $\begin{pmatrix} I & \Phi \\ \Phi^T & I \end{pmatrix}$ are either 1 or $1 \pm \sigma_i$, where σ_i $(i = 1, 2, \ldots, r)$ are the non-zero singular values of Φ.*

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 $(i = 1, 2, \ldots, r)$ are the non-zero singular values of Φ. The derivative of $\lambda = \frac{1}{1-\sigma} - 1$ with respect to σ is

$$\frac{d\lambda}{d\sigma} = \frac{1}{(1-\sigma)^2},$$

and it can become very large when σ increases or decreases toward one. This implies that the largest eigenvalues of J will be well separated from the others, i.e., the matrix J can be well approximated by a low-rank matrix.

Thus, based on (3.11) we can approximate A^{-1} as follows:

$$A^{-1} \approx A_0^{-1} + \begin{pmatrix} A_{11}^{-\frac{1}{2}} & 0 \\ 0 & A_{22}^{-\frac{1}{2}} \end{pmatrix} VHV^T \begin{pmatrix} A_{11}^{-\frac{1}{2}} & 0 \\ 0 & A_{22}^{-\frac{1}{2}} \end{pmatrix}, \tag{3.13}$$

where VHV^T is a rank-k approximation to the matrix J in (3.12). 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 3.2. *Assume A is SPD matrix factored in the form (3.10) and let $\Phi = A_{11}^{-\frac{1}{2}} A_{12} A_{22}^{-\frac{1}{2}}$, then we have $0 \le \sigma_i < 1$ for each singular value σ_i of Φ $(i = 1, 2, \ldots, t)$, 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

$$\lambda(\Phi^T\Phi) = 1 - \lambda(A_{22}^{-\frac{1}{2}} S A_{22}^{-\frac{1}{2}}). \tag{3.14}$$

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 (3.14) implies $\lambda(\Phi^T\Phi) < 1$. Therefore, $0 \le \lambda(\Phi^T\Phi) < 1$ and hence:

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

This completes the proof. ∎

Theorem 3.2 shows that 1 is an upper bound for the singular values of Φ in (3.10). 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 3.2 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

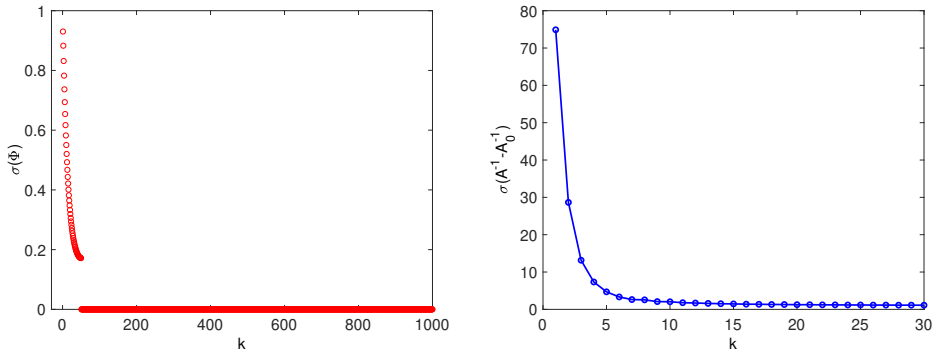

FIG. 3.2. *Singular values of* Φ *(left) and the leading* 30 *singular values of* $A^{-1} - A_0^{-1}$ *(right) for a* $2,000$*-by-*$2,000$ *discretized negative* 2*D Laplacian with the zero Dirichlet boundary condition where* A *is partitioned into* 2 *even-sized blocks.*

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 3.2.

As can be seen from Figure 3.2, 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 fifth largest singular value of $A^{-1} - A_0^{-1}$ to the largest one is approximately equal to $6.3e{-}2$ and the ratio of the tenth largest singular value of $A^{-1} - A_0^{-1}$ to the largest one decreases to only $2.7e{-}2$. 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.

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):

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

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

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: $A|_{\mathbf{I}_j,\mathbf{I}_i}(u_i + \delta_i) = v_j$ $(j = i_1, i_2, \ldots, i_s)$. This leads to the following equation which is solved for color j:

$$A|_{\mathbf{I}_j,\mathbf{I}_j}\delta_j = v_j - A|_{\mathbf{I}_j,\mathbf{I}_i} u_i. \tag{3.15}$$

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 (3.15) at once for all colors and updating block coordinates simultaneously. This is illustrated in Algorithm 4.

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

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

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

M_i^{-1} from (3.8) 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.

ALGORITHM 5
Combination of M_i^{-1} *and Algorithm 4 for solving* $A|_{\mathbf{I}_i,\mathbf{I}_i}u_i = v_i$

1: **procedure** u_i = RecSolv1($A|_{\mathbf{I}_i,\mathbf{I}_i}, v_i, i$)
2: **if** i is leaf node **then**
3: Solve $L_iU_iu_i = v_i$
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 $u_{c_1} = \texttt{RecSolv1}(A|_{\mathbf{I}_{c_1},\mathbf{I}_{c_1}}, \mathbf{z}_{c_1}, c_1)$.
8: Compute $u_{c_2} = \texttt{RecSolv1}(A|_{\mathbf{I}_{c_2},\mathbf{I}_{c_2}}, \mathbf{z}_{c_2}, c_2)$.
9: Stack $u_i = \begin{pmatrix} u_{c_1} \\ u_{c_2} \end{pmatrix}$.
10: **end if**
11: **For** $i = 1 : m$
12: u_i=BJ($A|_{\mathbf{I}_i,\mathbf{I}_i}, u_i, v_i$)
13: **Endfor**
14: **end procedure**

Let us consider how to integrate this combination into the construction phase, i.e., Algorithm 3. First, we would like to replace the following operation in Line 8 of Algorithm 3:

$$\left(I - A|_{\mathbf{I}_i,\mathbf{I}_i}\begin{pmatrix} M_{c_1}^{-1} & \\ & M_{c_2}^{-1} \end{pmatrix}\right)\begin{pmatrix} v_{c_1} \\ v_{c_2} \end{pmatrix}$$

by the operation:

$$\begin{pmatrix} v_{c_1} \\ v_{c_2} \end{pmatrix} - A|_{\mathbf{I}_i,\mathbf{I}_i}\begin{pmatrix} \texttt{RecSolv1}(A|_{\mathbf{I}_{c_1},\mathbf{I}_{c_1}}, v_{c_1}, c_1) \\ \texttt{RecSolv1}(A|_{\mathbf{I}_{c_2},\mathbf{I}_{c_2}}, v_{c_2}, c_2) \end{pmatrix}, \tag{3.16}$$

in order to account for the fact that the low-rank term should correct the combined iteration, i.e., the *corrected* block-diagonal solves from lower levels. This would have the effect of improving the solution accuracy of $A|_{\mathbf{I}_{c_i},\mathbf{I}_{c_i}}x = u_{c_i}$ $(i = 1, 2)$ and it would lead to more accurate low-rank correction term $V_iG_iV_i^T$.

However, Algorithm 5 is only applicable for two children nodes c_1 and c_2 in Line 8 of Algorithm 3 because the low-rank correction is not available at node i at the time of construction. The children c_1, c_2 can themselves invoke `RecSolv1` because in a bottom-up approach the low-rank corrections are already available in these nodes. Therefore, we need to modify Algorithm 5 into an algorithm that skips the low-rank correction at the current level, but calls `RecSolv1` for the children. The modification will be referred to as *Nested Block Jacobi* iteration and is sketeched in Algorithm 6. Note that this algorithm is used only in the construction phase and that the only difference with Algorithm 5 is that it omits the low rank correction at level i, i.e., the current level.

ALGORITHM 6
Nested block-Jacobi correction scheme

1: **procedure** u_i = `NSBJ`$(A|_{\mathbf{I}_i,\mathbf{I}_i}, v_i, i)$
2: **if** i is leaf node **then**
3: Solve $L_iU_iu_i = v_i$ (By Incomplete LU factorization of M_j)
4: **else**
5: Partition v_i into $\begin{pmatrix} v_{c_1} \\ v_{c_2} \end{pmatrix}$ – according to sets $\mathbf{I}_{c_1}, \mathbf{I}_{c_2}$
6: Compute $u_{c_1} =$ `RecSolv1`$(A|_{\mathbf{I}_{c_1},\mathbf{I}_{c_1}}, v_{c_1}, c_1)$
7: Compute $u_{c_2} =$ `RecSolv1`$(A|_{\mathbf{I}_{c_2},\mathbf{I}_{c_2}}, v_{c_2}, c_2)$
8:
9: Stack $u_i = \begin{pmatrix} u_{c_1} \\ u_{c_2} \end{pmatrix}$
10: **end if**
11: **For** $i = 1 : m$
12: u_i=BJ$(A|_{\mathbf{I}_i,\mathbf{I}_i}, u_i, v_i)$
13: **Endfor**
14: **end procedure**

Thus, Algorithm 3 replaces (3.16) by the following operation:

$$v_i - A|_{\mathbf{I}_i,\mathbf{I}_i}\,\mathrm{NSBJ}(A|_{\mathbf{I}_i,\mathbf{I}_i}, v_i, i), \tag{3.17}$$

where `NSBJ` is defined in Algorithm 6. 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 advantage of using (3.17) over (3.16) can be easily seen from a two-level version of the proposed preconditioner. Now, we will first explore the operation `NSBJ`$(A|_{\mathbf{I}_i,\mathbf{I}_i}, v_i, i)$ under mild assumptions.

PROPOSITION 3.3. *Let $\tilde{A}_i$ denote the block diagonal matrix*

$$\tilde{A}_i := \begin{pmatrix} A|_{\mathbf{I}_{i_1},\mathbf{I}_{i_1}} & & & \\ & A|_{\mathbf{I}_{i_2},\mathbf{I}_{i_2}} & & \\ & & \ddots & \\ & & & A|_{\mathbf{I}_{i_s},\mathbf{I}_{i_s}} \end{pmatrix}, \tag{3.18}$$

where $i_1, i_2, \ldots, i_s$ are the leaf nodes in $\mathcal{T}$ which are also the descendants of node i. Assume that u_{c_j} $(j = 1, 2)$ returned by Lines 6-7 in Algorithm 6 satisfies $A|_{\mathbf{I}_{c_j},\mathbf{I}_{c_j}} u_{c_j} = v_{c_j}$ exactly, then we have

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

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 $u_i = \begin{pmatrix} A|_{\mathbf{I}_{c_1},\mathbf{I}_{c_1}} & \\ & A|_{\mathbf{I}_{c_2},\mathbf{I}_{c_2}} \end{pmatrix}^{-1} v_i$ in Step 9. Next, applying m Block-Jacobi correction steps in Lines 11-13 is equivalent to computing the following sequence :

$$\begin{cases} u^{(i)} = (I - \tilde{A}_i^{-1}A|_{\mathbf{I}_i,\mathbf{I}_i})u^{(i-1)} + \tilde{A}_i^{-1}v_i, \\ i = 1, 2, \ldots, m, \end{cases} \tag{3.19}$$

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

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

The proof is complete. ∎

Therefore, the residual matrix $I - A|_{\mathbf{I}_i,\mathbf{I}_i} X_i$ at node i is:

$$\begin{aligned} I - A|_{\mathbf{I}_i,\mathbf{I}_i} X_i &= A|_{\mathbf{I}_i,\mathbf{I}_i} (I - \tilde{A}_i^{-1}A|_{\mathbf{I}_i,\mathbf{I}_i})^m \left(A|_{\mathbf{I}_i,\mathbf{I}_i}^{-1} - \begin{pmatrix} A|_{\mathbf{I}_{c_1},\mathbf{I}_{c_1}} & \\ & A|_{\mathbf{I}_{c_2},\mathbf{I}_{c_2}} \end{pmatrix}^{-1} \right) \\ &= A|_{\mathbf{I}_i,\mathbf{I}_i} (I - \tilde{A}_i^{-1}A|_{\mathbf{I}_i,\mathbf{I}_i})^m A|_{\mathbf{I}_i,\mathbf{I}_i}^{-1} \left(I - A|_{\mathbf{I}_i,\mathbf{I}_i} \begin{pmatrix} A|_{\mathbf{I}_{c_1},\mathbf{I}_{c_1}} & \\ & A|_{\mathbf{I}_{c_2},\mathbf{I}_{c_2}} \end{pmatrix}^{-1} \right) \\ &= (I - A|_{\mathbf{I}_i,\mathbf{I}_i} \tilde{A}_i^{-1})^m \left(I - A|_{\mathbf{I}_i,\mathbf{I}_i} \begin{pmatrix} A|_{\mathbf{I}_{c_1},\mathbf{I}_{c_1}} & \\ & A|_{\mathbf{I}_{c_2},\mathbf{I}_{c_2}} \end{pmatrix}^{-1} \right). \end{aligned} \tag{3.20}$$

Equation (3.20) suggests that matrix $I - A|_{\mathbf{I}_i,\mathbf{I}_i} X_i$ will have a more favorable low-rank representation than matrix $I - A|_{\mathbf{I}_i,\mathbf{I}_i} \begin{pmatrix} A|_{\mathbf{I}_{c_1},\mathbf{I}_{c_1}} & \\ & A|_{\mathbf{I}_{c_2},\mathbf{I}_{c_2}} \end{pmatrix}^{-1}$ which is the matrix in (3.3). In the extreme case when $\tilde{A}_i = \begin{pmatrix} A|_{\mathbf{I}_{c_1},\mathbf{I}_{c_1}} & \\ & A|_{\mathbf{I}_{c_2},\mathbf{I}_{c_2}} \end{pmatrix}$,

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

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

In fact, the eigenvalue distribution of the preconditioned matrix highly depends 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 demonstrate this property with two examples shown in Figure 3.3. Here, the matrix A is the non-symmetric `facsimile convergence` matrix obtained from the SuiteSparse collection [7]. This matrix has dimension $n = 541$ and is indefinite. Figure 3.3 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 with a rank 20 low rank correction converges in 34 iterations to reach 6 digits of accuracy while preconditioned GMRES with a rank 5 low rank correction converges in 76 iterations.

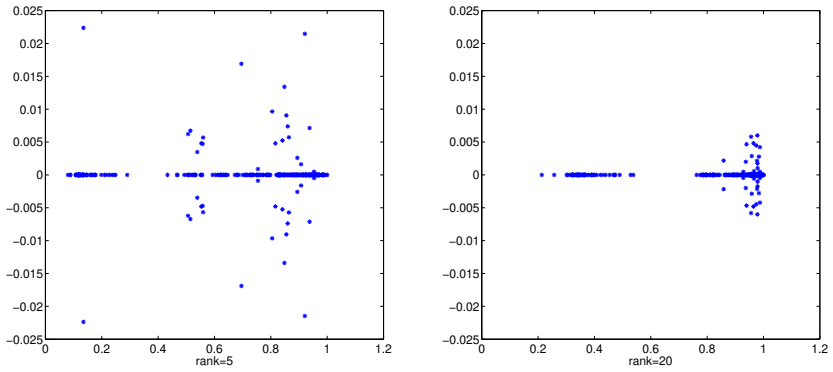

FIG. 3.3. *Spectra of the preconditioned matrix: rank* 5 *correction (left) and rank* 20 *correction (right), where* $\mathcal{T}$ *has two levels. Here, the number of Block-Jacobi correction steps is taken as* $m = 3$.

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 Intel C++ compiler using -O3 optimization.

In the numerical experiments, the original sparse matrices were partitioned by using `PartGraphKway` from the METIS [18] 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.

For the SPD matrices, the MCLR preconditioner with *corrections* was compared with the incomplete Cholesky factorization with threshold dropping (ICT) and the MSLR preconditioner [29]. The accelerator we used for the MCLR with *corrections*, MSLR and ICT preconditioners is the conjugate gradient (CG) method [15]. For other test matrices, the MCLR preconditioner with *corrections* was compared with the incomplete LU factorization with threshold dropping (ILUT) and the GMSLR

preconditioner [8] and the accelerator used is GMRES [25, 27].

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

- fill: Fill factor defined as $\frac{nnz(prec)}{nnz(A)}$;
- 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;
- 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 OpenMP [3] where the number of threads used is the number of cores, which is 24.

4.1. Test 1. We first consider the following convection-diffusion equation:

$$\begin{aligned} -\triangle u - \alpha \cdot \nabla u - \beta u &= f \text{ in } \Omega, \\ u &= 0 \text{ on } \partial\Omega, \end{aligned} \tag{4.1}$$

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.

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 (4.1) 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 4.1, 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 4.1. 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.

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 4.2 the ILU fill factor remains constant since we kept n_d fixed. 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., $r_k \leq 5$).

TABLE 4.1

The fill factors and iteration counts for solving (4.1) 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

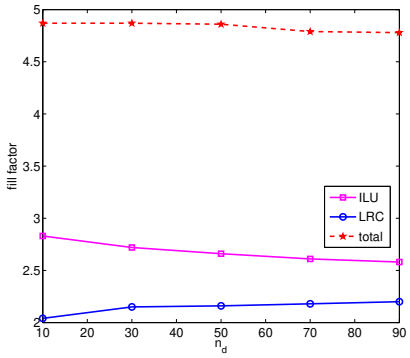

FIG. 4.1. *The fill factors from ILU and low-rank correction with respect to different n_d for the test problem in Table 4.1.*

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 4.3 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.

4.2. Test 2. The second test is the symmetric problem:

$$\begin{aligned} -\triangle u - \beta u &= f \text{ in } \Omega, \\ u &= 0 \text{ on } \partial\Omega, \end{aligned} \tag{4.2}$$

where $\Omega = (0,1)^3$. These PDEs were discretized by the 7-point stencil.

We solve (4.2) 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 4.4. We chose $r_k = 2$ for the SPD case and $r_k = 5$ for the indefinite case. In order to have a fair comparison, the parameters of MSLR and GMSLR preconditioners were chosen such that they yield roughly the same fill-factors as those of MCLR. Table 4.4 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

TABLE 4.2
The fill factors and iteration counts for solving (4.1) 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}.

rk	fill (ILU)	fill (Low-rank)	fill (total)	its	p-t	i-t
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

TABLE 4.3
The fill factors and iteration counts for solving (4.1) 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

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 (4.2) 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.

TABLE 4.4
Comparison among MCLR *with* corrections*,* ICT/ILUT *and* MSLR/GMSLR *preconditioners for solving symmetric positive definite/indefinite linear systems from the 3-D PDEs* (4.2) *with the* CG/GMRES *methods, where* m *was fixed at* 5.

Mesh	s	MCLR					ICT/ILUT				MSLR/GMSLR					
		rk	fill	its	p-t	i-t	fill	its	p-t	i-t	lev	rk	fill	its	p-t	i-t
32^3		2	3.29	6	.11	.19	3.05	16	.15	.48	7	32	3.12	31	.14	.42
64^3	0.00	2	3.53	10	1.02	1.50	3.03	25	1.08	3.21	10	32	3.66	60	1.03	2.55
128^3		2	3.53	15	3.30	4.69	3.01	40	8.21	10.79	13	32	4.00	116	4.95	7.10
32^3		5	3.31	9	.09	.25	3.02	29	.28	.44	7	16	3.33	28	.10	.30
64^3	0.04	5	4.78	47	1.23	4.70	4.47	78	1.59	8.09	10	25	5.26	202	1.24	7.10
128^3		5	3.32	216	5.00	11.99	3.42	F	12.01	–	13	64	3.40	F	8.02	–

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 (4.2) with $\beta = 0$ on a 50^3 grid. We fixed r_k at 2 in the experiments. The left subfigure of Figure 4.2, shows a steady improvement in

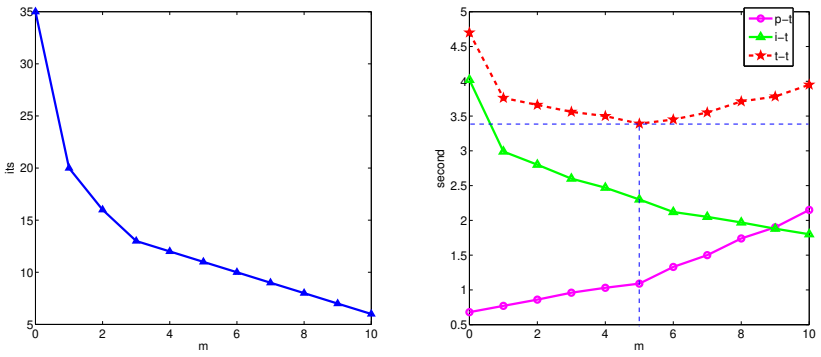

FIG. 4.2. *Left: the iteration numbers with respect to different m. Right: the iteration time and the precondtioner construction time with respect to different m. Here, the grid is fixed to 50^3.*

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 4.2 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., 2.25 seconds to build preconditioner and 2.30 seconds to converge) for this test.

4.3. General sparse linear systems. We now test MCLR with *corrections* on various general matrices obtained from the SuiteSparse Matrix Collection [7]. These matrices arise from a wide range of application areas, including both symmetric and nonsymmetric problems. Table 4.5 presents a short description.

TABLE 4.5
Names, orders (N), numbers of nonzeros (nnz) and short descriptions of the test matrices.

Matrix	Order	nnz	symmetric	Description
cfd1	70,656	1,825,580	yes	CFD problem
cfd2	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

Numerical results are presented in Table 4.6. 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 4.5. In contrast, ICT fails to converge for `cfd1` and `cfd2`, ILUT fails to converge to solve `xenon1` and `xenon2` and MSLR can not converge for `cfd1`. In these experiments, m was set to $m = 5$.

TABLE 4.6
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					
	rk	fill	its	p-t	i-t	fill	its	p-t	i-t	lev	rk	fill	its	p-t	i-t
cfd1	2	1.40	82	1.20	2.19	1.56	F	.63	–	7	64	1.74	F	.94	–
cfd2	2	1.79	95	.98	4.06	1.89	F	.99	–	8	80	1.86	108	1.56	5.34
ecology1	2	2.60	13	.99	2.30	2.59	25	1.01	4.08	8	64	2.71	60	.90	3.42
thermal2	2	4.40	39	4.60	6.76	4.44	115	6.03	14.59	8	64	4.94	65	5.34	9.36
xenon1	5	1.31	42	.59	3.90	1.37	F	3.62	–	3	32	1.17	156	0.63	6.54
xenon2	5	2.80	57	1.40	4.51	3.11	F	5.80	–	10	64	2.78	269	1.66	7.62
ML_Laplace	5	1.01	72	2.10	1.23	1.05	167	1.80	3.22	6	64	1.06	150	2.20	2.03
Atmosmodd	5	3.23	18	2.03	5.02	3.50	44	3.43	11.01	10	4	3.21	30	3.34	7.34
Atmosmodl	5	3.45	8	2.87	4.21	3.51	26	3.76	10.00	11	4	3.60	15	3.29	7.59
Transport	5	2.12	43	4.50	4.02	2.15	92	7.28	9.33	6	4	2.10	79	6.89	6.09

5. Conclusion. We described a parallel 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 MCLR preconditioner can be applied to solve both symmetric and non-symmetric 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.

Because of these appealing features we plan to extend MCLR to complex (non-Hermitian) systems and eigenvalue problems [30, 22]. We will also explore other low-rank correction techniques to try to further reduce the construction time of the preconditioner.

REFERENCES

[1] A. Aminfar, S. Ambikasaran, and E. Darve, *A fast block low-rank dense solver with applications to finite-element matrices*, J. Comput. Phys., 304 (2016), pp. 170–188.
[2] M. Bebendorf, *Hierarchical Matrices: A Means to Efficiently Solve Elliptic Boundary Value Problems*, Lecture Notes in Computational Science and Engineering, Springer Berlin Heidelberg, 2008.
[3] OpenMP Architecture Review Board, *OpenMP Application Program Interface*, Version 3.1, (2011).
[4] S. L. Borne and L. Grasedyck, *$\mathcal{H}$-matrix preconditioners in convection-dominated problems*, SIAM J. Matrix Anal. Appl., 27 (2006), pp. 1172–1183.
[5] D. Cai, E. Chow, L. Erlandson, Y. Saad, and Y. Xi, *SMASH: Structured Matrix Approximation by Separation and Hierarchy*, Numer. Linear Algebra Appl., 25 (2018).
[6] E. Chow and Y. Saad, *Approximate inverse preconditioners via sparse-sparse iterations*, SIAM J. Sci. Comput., 19 (1998), pp. 995–1023.
[7] T. A. Davis and Y. Hu, *The university of florida sparse matrix collection*, ACM Trans. Math. Software, 38 (2011).
[8] G. Dillon, V. Kalantzis, Y. Xi, and Y. Saad, *A hierarchical low-rank schur complement preconditioner for indefinite linear systems*, SIAM J. Sci. Comput., 40 (2018), pp. A2234–A2252.
[9] A. Gillman, P. M. Young, and P. G. Martinsson, *A direct solver with o(N) complexity for integral equations on one-dimensional domains*, Front. Math. China, 7 (2012), pp. 217–247.
[10] G. H. Golub and C. F. Van Loan, *Matrix Computations, 4th edition*, Johns Hopkins University Press, Baltimore, MD, 4th ed., 2013.
[11] M. Grote and T. Huckle, *Parallel preconditioning with sparse approximate inverses*, SIAM J. Sci. Comput., 18 (1997), pp. 838–853.
[12] W. Hackbusch and S. Börm, *$\mathcal{H}^2$-matrix approximation of integral operators by interpolation*, Appl. Numer. Math., 43 (2002), pp. 129–143.
[13] W. Hackbusch, B. N. Khoromskij, and S. A. Sauter, *On $\mathcal{H}^2$-matrices*, in Lectures on applied mathematics, Springer, Berlin, 2000, pp. 9–29.
[14] P. Hénon and Y. Saad, *A parallel multistage ILU factorization based on a hierarchical graph decomposition*, SIAM J. Sci. Comput., 28 (2006), pp. 2266–2293.
[15] M. R. Hestenes and E. L. Stiefel, *Methods of conjugate gradients for solving linear systems*, J. Research Nat. Bur. Standards, 49 (1952), pp. 409–436.
[16] K. L. Ho and L. Greengard, *A fast direct solver for structured linear systems by recursive skeletonization*, SIAM J. Sci. Comput., 34 (2012), pp. A2507–A2532.
[17] K. L. Ho and L. Ying, *Hierarchical interpolative factorization for elliptic operators: Integral equations*, Commun. Pur. Appl. Math., 69 (2016), pp. 1314–1353.
[18] 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.
[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*, J. Supercomput., 63 (2013), pp. 443–466.
[21] R. Li, Y. Xi, and Y.Saad, *Schur complement-based domain decomposition preconditioners with low-rank corrections*, Numer. Linear Algebra Appl., 23 (2016), pp. 706–729.
[22] X. Liu, Y. Xi, Y. Saad, and M. V. de Hoop, *Solving the 3D high-frequency Helmholtz equation using contour integration and polynomial preconditioning*, arXiv:1811.12378, (2018).
[23] X. LIU, J. XIA, and M. V. De HOOP, *Parallel randomized and matrix-free direct solvers for large structured dense linear systems*, SIAM J. Sci. Comput., 38 (2016), pp. S508–S538.
[24] F. H. Rouet, X. S. Li, P. Ghysels, and A. Napov, *A distributed-memory package for dense hierarchically semi-separable matrix computations using randomization*, ACM Trans. Math. Softw., 42 (2016).
[25] Y. Saad, *A flexible inner-outer preconditioned GMRES algorithm*, SIAM J. Sci. Comput., 14 (1993), pp. 461–469.
[26] ———, *Iterative Methods for Sparse Linear Systems, 2nd edition*, SIAM, Philadelpha, PA, 2003.
[27] Y. Saad and M. H. Schultz, *GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems*, SIAM J. Sci. Statist. Comput., 7 (1986), pp. 856–869.
[28] Y. Saad and B. Suchomel, *ARMS: An algebraic recursive multilevel solver for general sparse linear systems*, Numer. Lin. Alg. Appl., 9 (2002).
[29] Y. Xi, R. Li, and Y. Saad, *An algebraic multilevel preconditioner with low-rank corrections for sparse symmetric matrices*, SIAM J. Matrix Anal. Appl., 37 (2016), pp. 235–259.
[30] Y. Xi and Y. Saad, *A rational function preconditioner for indefinite sparse linear systems*,

SIAM J. Sci. Comput., 39 (2017), pp. A1145–A1167.
[31] Y. Xi and J. Xia, *On the stability of some hierarchical rank structured matrix alogrithms*, SIAM J. Matrix Anal. Appl., 37 (2016), pp. 1279–1303.
[32] J. Xia, *Efficient structured multifrontal factorization for general large sparse matrices*, SIAM J. Sci. Comput., 35 (2013), pp. A832–A860.
[33] J. Xia, S. Chandrasekaran, M. Gu, and X. S. Li, *Superfast multifrontal method for large structured linear systems of equations*, SIAM J. Matrix Anal. Appl., 31 (2010), pp. 1382–1411.
[34] J. Xia, Y. Xi, S. Cauley, and V. Balakrishnan, *Fast sparse selected inversion*, SIAM J. Matrix Anal. Appl., 36 (2015), pp. 1283–1314.