DIVIDE AND CONQUER LOW-RANK PRECONDITIONING TECHNIQUES *

RUIPENG LI † AND YOUSEF SAAD†

Abstract. This paper presents a preconditioning method based on a recursive multilevel low-rank approximation approach. The basic idea is to recursively divide the problem into two and apply a low-rank approximation to a matrix obtained from the Sherman-Morrison formula. The low-rank approximation may be computed by the partial Singular Value Decomposition (SVD) or it can be approximated by the Lanczos bidiagonalization method. This preconditioning method is designed for handling different levels of parallelism on multi-processor systems of multi(many)-core processors. Numerical experiments indicate that, when combined with Krylov subspace accelerators, this method can be efficient and robust for solving symmetric indefinite sparse linear systems.

Key words. Sherman-Morrison formula, low-rank approximation, singular value decomposition, recursive multilevel preconditioner, parallel preconditioner, incomplete LU factorization, Krylov subspace method, domain decomposition

1. Introduction. Krylov subspace methods preconditioned with a form of Incomplete LU (ILU) factorization can be quite effective in solving general linear systems but there are situations where the ILU preconditioners will fail. For instance when the matrix is highly indefinite, an incomplete LU preconditioner is unlikely to work, either because the construction of the factors will not complete or because the resulting factors are unstable. Another situation is related to the architecture under which the system is being solved. Building and using an ILU factorization is a highly scalar process. Blocking is often not even exploited the way it is for direct solvers. As a result ILU preconditioners are ruled out for computers equipped with massively parallel co-processors like GP-GPUs.

These situations have motivated the development of a class of "approximate inverse preconditioners" in the 1990s as alternatives to ILUs. However, the success of these methods was mixed in part because of the 'local' nature of these preconditioners. Generally, the cost of developing the preconditioners and the number of iterations required for convergence can be high.

This paper describes another class of preconditioners which is also rooted in approximate inverses. However, these preconditioners do not rely on sparsity but on low-rank approximations. In a nutshell the idea is based on the observation that if a domain is partitioned in two subdomains one can get the inverse of the matrix based on the whole domain by the block-diagonal of the inverses on each subdomain plus a low-rank correction. In fact the rank needed to obtain the exact inverse is equal to the number of interface points. However, an excellent approximation suitable for preconditioning can often be obtained from a much smaller rank. The idea then is to exploit this in a recursive fashion assuming the domain is recursively divided in two until the domains reached are small. The resulting technique will be called the Recursive Multilevel Low-Rank (RMLR) preconditioner.

It is useful to compare the potential advantages and disadvantages of an approach of this type relative to those of traditional ILU-type preconditioners, or to ARMS-type approaches [19]. On the negative side, this new approach may lead to some

*This work was supported by DOE under grant DOE/DE-FG-08ER25841 and by the Minnesota Supercomputing Institute

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

obvious difficulties in the implementation. Exploiting recursivity as much as possible simplifies the implementation. On the other hand, this method appears to be simpler than that based on the HSS framework where LU factorizations are developed instead of approximate inverses, exploiting again low-rank matrices, see, e.g., [26, 25, 10].

The potential advantages of RMLR techniques outnumber its disadvantages. First, the preconditioner is very suitable for SIMD-type calculations. In fact one of the main motivations for developing a strategy of this type is the difficulty that is encountered when one attempts to obtain good performance from sparse matrix computations on GP-GPUs [3, 2, 16, 1, 24, 21, 15, 14]. Apart from the last level, which will require a standard solve of some sort, all other levels involve vector operations when a solve is performed. This is ideal for GPU processors. In addition, the domain decomposition (DD) framework lends itself easily to a macro-level parallel approach. Thus, one can easily imagine implementing this approach on a multiprocessor system based on a multi(many)-core architecture exploiting two levels of parallelism.

A second appeal of the RMLR approach is that it seems that this method is oblivious to indefiniteness. What matters is how well we approximate the inverses of the matrices at the lowest levels, and how good is the low rank approximation used. This could lead to an analysis regarding the quality of the preconditioner, independently of the spectrum of the original matrix.

A third appeal, shared by all approximate inverse-type preconditioners, is that the preconditioner is 'updatable' in the sense that if one is not satisfied with its performance on a particular problem, the preconditioner accuracy can be improved without foregoing work previously done to build the preconditioner.

2. A Recursive Multilevel Low-rank Preconditioner. This section considers a class of divide-and-conquer methods for computing a preconditioner for a matrix $A \in \mathbb{R}^{n\times n}$, based on *recursive low-rank approximations.* For simplicity we begin with a model problem in which A is a *symmetric* matrix derived from a 5-point stencil discretization of a 2-D problem on an $n_x \times n_y$ grid. In this case the coefficient matrix will be as follows:

$$A = \left(\begin{array}{cccc|ccc} A_1 & D_2 & & & & & \\ D_2 & A_2 & D_3 & & & & \\ & \ddots & \ddots & \ddots & & & \\ & & D_\alpha & A_\alpha & D_{\alpha+1} & & \\ \hline & & & D_{\alpha+1} & A_{\alpha+1} & \ddots & \\ & & & & \ddots & \ddots & \ddots \\ & & & & & D_{n_y} & A_{n_y} \end{array}\right), \tag{2.1}$$

where $\{A_j\}$ and $\{D_j\}$ are sets of n_y tridiagonal and $n_y - 1$ diagonal matrices of dimension n_x, so that $n = n_x n_y$. In equation (2.1) the matrix A has been split as

$$A = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} \equiv \begin{pmatrix} A_{11} & \\ & A_{22} \end{pmatrix} + \begin{pmatrix} & A_{12} \\ A_{21} & \end{pmatrix}, \tag{2.2}$$

with $A_{11} \in \mathbb{R}^{m\times m}$ and $A_{22} \in \mathbb{R}^{(n-m)\times(n-m)}$, where we assume that $0 < m < n$, and m is a multiple of n_x, i.e., $m = \alpha n_x$ with $\alpha \in \{1, 2, \ldots, n_y - 1\}$. Then, an interesting

observation is that the submatrix $A_{12} = A_{21}^T \in \mathbb{R}^{m\times(n-m)}$ has rank n_x because

$$A_{12} = A_{21}^T = \begin{pmatrix} & \\ & \\ D_{\alpha+1} & \end{pmatrix} = -E_1 E_2^T, \tag{2.3}$$

with

$$E_1 := \begin{pmatrix} & \\ & \\ D_{E_1} & \end{pmatrix} \in \mathbb{R}^{m\times n_x} \quad \text{and} \quad E_2 := \begin{pmatrix} D_{E_2} \\ & \\ & \end{pmatrix} \in \mathbb{R}^{(n-m)\times n_x}, \tag{2.4}$$

where D_{E_1} and D_{E_2} are diagonal matrices of dimension n_x such that $D_{E_1} D_{E_2} = -D_{\alpha+1}$. For example, in the common case when $D_{\alpha+1} = -I$, we can take $D_{E_1} = D_{E_2} = I_{n_x}$. Therefore, (2.2) can be rewritten as

$$A = \begin{pmatrix} A_{11} + E_1 E_1^T & \\ & A_{22} + E_2 E_2^T \end{pmatrix} - \begin{pmatrix} E_1 E_1^T & E_1 E_2^T \\ E_2 E_1^T & E_2 E_2^T \end{pmatrix}. \tag{2.5}$$

Thus, we have

$$A = B - EE^T, \quad B := \begin{pmatrix} B_1 & \\ & B_2 \end{pmatrix} \in \mathbb{R}^{n\times n}, \quad E := \begin{pmatrix} E_1 \\ E_2 \end{pmatrix} \in \mathbb{R}^{n\times n_x}, \tag{2.6}$$

with

$$B_1 := A_{11} + E_1 E_1^T \in \mathbb{R}^{m\times m}, \quad B_2 := A_{22} + E_2 E_2^T \in \mathbb{R}^{(n-m)\times(n-m)}.$$

Note that the diagonal matrix $E_1 E_1^T$ perturbs the last n_x diagonal entries of A_{11}, while the diagonal matrix $E_2 E_2^T$ perturbs the first n_x diagonal entries of A_{22}.

Consider the relation (2.2) again for a symmetric matrix and note that we have rewritten this in the form of a correction shown in (2.6). From (2.6) and the Sherman-Morrison formula we can derive the equation:

$$A^{-1} = B^{-1} + B^{-1}E(\underbrace{I - E^T B^{-1} E}_{X})^{-1} E^T B^{-1} \equiv B^{-1} + B^{-1} E X^{-1} E^T B^{-1}, \tag{2.7}$$

with B, E, B_1, B_2 defined above. A formalism similar to the one described above was exploited in [22] for the problem of determining the diagonal of the inverse of a matrix.

2.1. Recursive solves. A first thought for exploiting (2.7) in a recursive framework, one that will turn out to be impractical, is to approximate X by some nonsingular matrix $\tilde{X}$. Then, the preconditioning operation applied to a vector v is given by:

$$B^{-1}[v + E\tilde{X}^{-1} E^T B^{-1} v]. \tag{2.8}$$

Each application of a preconditioner based on an approach of this type will require two solves with B, one solve with $\tilde{X}$ (a small matrix) and products with E and E^T. In a recursive scheme the number of solves (with the B's) will increase like 2^{nlev} with the number of levels, $nlev$, and so the cost will explode for a moderately large $nlev$. Thus, this scheme will not work and we will need to be careful about ways to exploit equality (2.7). Practical implementations of the recursive scheme just

sketched will be based on various approximations to $B^{-1}E$ and the related matrix X. One possibility is to compute a sparse approximation to $B^{-1}E$ using ideas from approximate inverses and sparse-sparse techniques, see, e.g., [6]. Sparse approximate inverse methods have been used in a context that is somewhat related in [5]. We expect this approach not to work very well for highly indefinite matrices, as this is a well-known weakness of approximate inverse methods in general. Instead, we consider an alternative, described next, which relies on low-rank approximations.

2.2. Use of low-rank approximations. Low-rank approximations have become very popular recently for computing preconditioners. For example, the ILU factorization methods based on H-matrices rely on low-rank approximations [4]. A similar method has been introduced for solving the Helmholtz problem [10]. The main idea is based on the observation that intermediate Schur complement matrices in the LDL^T factorization of a specific order have numerically low-rank off-diagonal blocks, which makes them highly compressible and naturally represented in the hierarchical matrix (or H-matrix) framework. Similar ideas were exploited in the context of so-called hierarchically semi-separable (HSS) matrices [27, 26]. There are similarities between our work and the work based on hierarchical matrices but the main difference is that we do not resort to an LU factorization.

Our starting point is the relation (2.7). Assume that we have a low-rank approximation to the block $B^{-1}E$ in the form

$$B^{-1}E \approx U_k V_k^T, \tag{2.9}$$

where $U_k \in \mathbb{R}^{n\times k}$, $V_k \in \mathbb{R}^{n_x\times k}$, and k is small (for example $k = 3$). Then, (2.7) yields several possible approximations.

First, one can just substitute an approximation $\tilde{X}$ for X, as in (2.8). So by replacing $U_k V_k^T$ for $B^{-1}E$ in $X = I - (E^T B^{-1})E$, we let

$$G_k = I - V_k U_k^T E, \tag{2.10}$$

which is an approximation to X. The matrix G_k is of size $n_x \times n_x$ and it is inexpensive to solve systems with it once it is factored at the outset. As was seen above, a preconditioner based on a recursive application of (2.8), with $\tilde{X}$ replaced by G_k, will see its cost explodes with the number of levels, and so this option is avoided.

A computationally viable alternative is to replace $B^{-1}E$ by its approximation based on (2.9). This leads to:

$$M^{-1} = B^{-1} + U_k V_k^T G_k^{-1} V_k U_k^T, \tag{2.11}$$

which means that we can build approximate inverses based on low-rank approximations of the form

$$M^{-1} = B^{-1} + U_k H_k U_k^T, \quad \text{with} \quad H_k = V_k^T G_k^{-1} V_k. \tag{2.12}$$

It turns out that the matrix H_k can be computed in a simpler way than by the expression above. Specifically, it will be shown in Section 3 that

$$H_k = (I - U_k^T E V_k)^{-1}, \tag{2.13}$$

which will also be shown to be a symmetric matrix. The alternative expression (2.12) avoids the use of V_k explicitly. In other words, only the vectors in U_k and the matrix

H_k are required to carry out the solve associated with (2.12) in addition to the solve with B. Hence, applying the preconditioner to an arbitrary vector b requires computing $B^{-1}b$ to which the correction $U_k H_k U_k^T b$ is added. For now, we assume that the system with B can be solved in some unspecified manner.

It may appear somewhat wasteful for the accuracy of the scheme to approximate both $B^{-1}E$ and its transpose when deriving (2.11) from (2.7). Instead, a middle ground approach which approximates $B^{-1}E$ on one side only of the expression, will lead to the following:

$$M^{-1} = B^{-1} + B^{-1}EG_k^{-1}V_k U_k^T = B^{-1}[I + EG_k^{-1}V_k U_k^T]. \tag{2.14}$$

One drawback of the above scheme is that symmetry is lost. However, we will show in Section 3 that the preconditioning matrices defined by (2.11) and (2.14) are equal. Therefore, in what follows, we will only consider the two-sided low-rank approximations defined by (2.12).

2.3. Recursive multilevel low-rank approximation. This section describes a framework for constructing recursive multilevel low-rank approximations. We start by defining matrices at the first level as $A_0 \equiv A$, $E_0 \equiv E$ and $B_0 \equiv B$, where A, E and B are matrices defined in (2.6). The index i will be used for the i-th level. Recall that B_i is a 2×2 block diagonal matrix. If the two diagonal blocks are labeled as i_1 and i_2 respectively, then we can write,

$$A_i = B_i - E_i E_i^T, \quad B_i \equiv \begin{pmatrix} B_{i_1} & \\ & B_{i_2} \end{pmatrix}. \tag{2.15}$$

If another level is built from A_i, then by letting $A_{i_1} = B_{i_1}$ and $A_{i_2} = B_{i_2}$, we can recursively repeat the same process on A_{i_1} and A_{i_2}. Otherwise, if A_i is at the last level, it is factored by the incomplete Cholesky factorization (IC) as $A_i \approx \bar{R}_i^T \bar{R}_i$ if it is symmetric positive definite (SPD), or by the Incomplete LU factorization (ILU) as $A_i \approx \bar{L}_i \bar{U}_i$. If we view i_1 and i_2 as the children of i, this process can be represented by the binary tree displayed in Figure 2.1. It should be clarified that this binary tree is not necessarily a complete binary tree as shown.

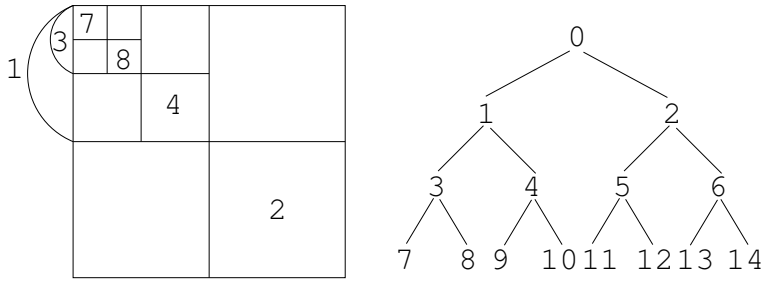

FIG. 2.1. *Recursive domain bisection and the binary tree structure in the RMLR method.*

Assume that i is a non-leaf node in the tree and that $(U_i)_k (V_i)_k^T$ is a rank-k approximation to $B_i^{-1}E_i$ and $(H_i)_k$ is the matrix defined in (2.12). For simplicity, we omit the subscript k from $(U_i)_k$, $(V_i)_k$ and $(H_i)_k$ in the remainder of this paper without loss of clarity. If we denote by M_i the preconditioning matrix of A_i and

assume that the low-rank approximation of the form (2.12) is used, then we have,

$$\begin{aligned} A_i^{-1} &\approx B_i^{-1} + U_i H_i U_i^T \\ &= \begin{pmatrix} A_{i_1}^{-1} & \\ & A_{i_2}^{-1} \end{pmatrix} + U_i H_i U_i^T \\ &\approx \begin{pmatrix} M_{i_1}^{-1} & \\ & M_{i_2}^{-1} \end{pmatrix} + U_i H_i U_i^T \\ &\equiv M_i^{-1}. \end{aligned} \tag{2.16}$$

On the other hand, if i is a leaf node, then M_i is given by $M_i^{-1} = \bar{R}_i^{-1}\bar{R}_i^{-T}$ or $M_i^{-1} = \bar{U}_i^{-1}\bar{L}_i^{-1}$. This gives a recursive definition of M_i,

$$M_i^{-1} = \begin{cases} \begin{pmatrix} M_{i_1}^{-1} & \\ & M_{i_2}^{-1} \end{pmatrix} + U_i H_i U_i^T, & \text{if } i \text{ is not a leaf node} \\ \bar{R}_i^{-1}\bar{R}_i^{-T} \text{ or } \bar{U}_i^{-1}\bar{L}_i^{-1}, & \text{otherwise} \end{cases} \tag{2.17}$$

Accordingly, the preconditioning operation $x = M_i^{-1}b$ can be performed as a recursive multilevel solve with low-rank approximation, $x =$ `RMLRSolve(i,b)`, which is shown below. In Line 6 and 7 of this function, vectors $b^{[i_1]}$ and $b^{[i_2]}$ are defined by, $b = \begin{pmatrix} b^{[i_1]} \\ b^{[i_2]} \end{pmatrix}$, which corresponds to the partition of i's rows into i_1 and i_2.

Function `RMLRSolve`(i, b)

Data: Low-rank approximations U_j, H_j for each non-leaf descendant j of i;
Factors $\bar{U}_j$, $\bar{L}_j$ or $\bar{R}_j$ for each leaf descendant j of i
Output: $x = M_i^{-1}b$, with M_i given by (2.17)

1 **if** *i is a leaf-node* **then**
2 Solve $A_i x = b$ by $x = \bar{U}_i^{-1}\bar{L}_i^{-1}b$ or $x = \bar{R}_i^{-1}\bar{R}_i^{-T}b$
3 **else**
4 $i_1 \leftarrow i$'s left child
5 $i_2 \leftarrow i$'s right child
6 $y_1 =$ `RMLRSolve(`i_1, $b^{[i_1]}$`)`
7 $y_2 =$ `RMLRSolve(`i_2, $b^{[i_2]}$`)`
8 $y = (y_1, y_2)^T$
9 $x = y + U_i H_i U_i^T b$

Finally, we analyze the storage requirement of the RMLR method. For every non-leaf node i, matrices U_i and H_i are saved, so the memory cost of saving the low-rank approximations is $O(nlev \cdot k \cdot n + (2^{nlev} - 1) \cdot \frac{k^2}{2})$, which increases with the number of levels $nlev$ and the rank k. For all leaf-nodes, ILU or IC factors are stored at a cost that depends on fill-ratios of the factorizations. In general, this cost increases as sizes of the last-level matrices increase and as dropping tolerances become smaller. The matrices stored in the RMLR approach are illustrated schematically in Figure 2.2.

2.4. Computation of low-rank approximations. In the above RMLR method, for every non-leaf node i, a rank-k approximation $U_i V_i^T$ to $B_i^{-1}E_i$ is required. This

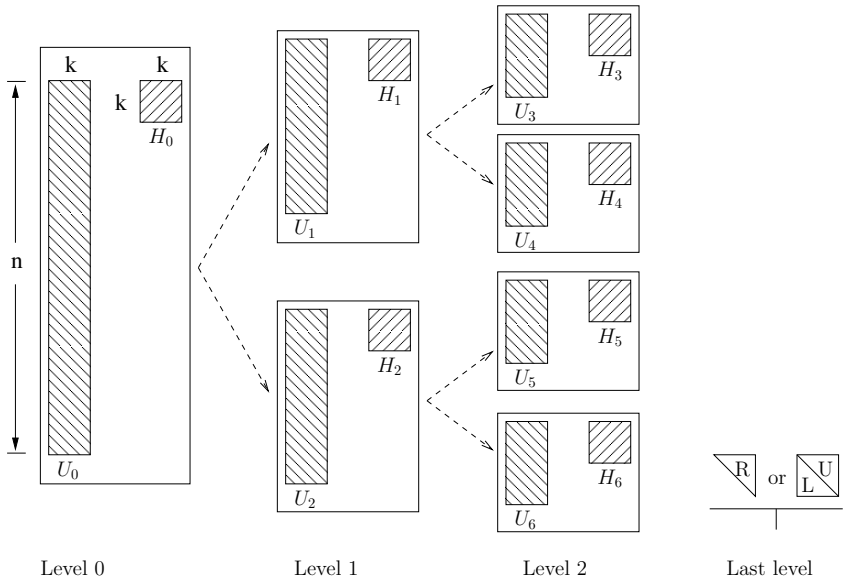

FIG. 2.2. *The matrices stored in the RMLR method.*

section describes three approaches for computing this low-rank approximation. In the first two approaches, we explicitly compute a matrix Z_i and then compute its SVD, i.e., $Z_i = \bar{U}\Sigma\bar{V}^T$. Thus, a rank-$k$ approximation $U_i V_i^T$ can be given by taking $U_i = \bar{U}_{:,1:k}\Sigma_{1:k,1:k}$ and $V_i = \bar{V}_{:,1:k}$.

In the first approach, we use $Z_i = B_i^{-1}E_i$. Recall that $B_i = \begin{pmatrix} A_{i_1} & \\ & A_{i_2} \end{pmatrix}$ and define $E_i = \begin{pmatrix} E_i^{[i_1]} \\ E_i^{[i_2]} \end{pmatrix}$ which corresponds to the row partition of i into i_1 and i_2. Then it follows that,

$$Z_i = B_i^{-1}E_i = \begin{pmatrix} A_{i_1} & \\ & A_{i_2} \end{pmatrix}^{-1} E_i = \begin{pmatrix} A_{i_1}^{-1}E_i^{[i_1]} \\ A_{i_2}^{-1}E_i^{[i_2]} \end{pmatrix} \equiv \begin{pmatrix} Y_1 \\ Y_2 \end{pmatrix}, \tag{2.18}$$

in which Y_1 and Y_2 are computed by exploiting the recursivity in the multilevel structure. By changing Line 9 in Function `RMLRSolve`, we obtain Function `MLSolve` shown below, which performs a recursive multilevel solve of A_i. If we assume the factorizations at the last level are exact, then this function indeed solves $x = A_i^{-1}b$ exactly. Therefore, Y_1 and Y_2 can be computed as follows. Each column j of Y_1 is given by $Y_1(:,j) =$ `MLSolve(`i_1`,` $e_j^{[i_1]}$`)`, in which $e_j^{[i_1]}$ is the jth column of $E_i^{[i_1]}$. Similarly, each column j of Y_2 is given by $Y_2(:,j) =$ `MLSolve(`i_2`,` $e_j^{[i_2]}$`)`, where $e_j^{[i_2]}$ is the jth column of $E_i^{[i_2]}$. Note that here we assume that before we start computing Z_i, matrices Z_j for all $j \in s(j)$ have been computed, where $s(j)$ is defined by $s(j) = \{x \mid x \text{ is a descendant of } i \text{ and } x \text{ is not a leaf node}\}$. This approach demands that we compute the low-rank approximations in a *postorder traversal* of the binary tree (for details, see, e.g., [7]).

However, this approach is not computationally feasible since computing Z_i requires Z_j for all $j \in s(j)$ and its cost will explode with the number of levels. It will

Function `MLSolve`$(i,\ b)$

Data: E_j, Z_j for each non-leaf descendant j of i; Factors $\bar{U}_j$, $\bar{L}_j$ or $\bar{R}_j$ for each leaf descendant j of i

Output: $x = A_i^{-1} b$

if *i is a leaf-node* **then**
  Solve $A_i x = b$ by $x = \bar{U}_i^{-1}\bar{L}_i^{-1} b$ or $x = \bar{R}_i^{-1}\bar{R}_i^{-T} b$
else
  $i_1 \leftarrow i$'s left child
  $i_2 \leftarrow i$'s right child
  $y_1 =$ `MLSolve(`$i_1,\ b^{[i_1]}$`)`
  $y_2 =$ `MLSolve(`$i_2,\ b^{[i_2]}$`)`
  $y = (y_1, y_2)^T$
  $x = y + Z_i(I - E_i^T Z_i)^{-1} Z_i^T$

only serve as a reference for the following two approaches for a few small problems in the numerical experiments.

In the second approach, we compute the low-rank approximations for $Z_i = \tilde{B}_i^{-1} E_i$, where $\tilde{B}_i$ is an approximation to B_i, which is defined by

$$\tilde{B}_i = \begin{pmatrix} M_{i_1} & \\ & M_{i_2} \end{pmatrix}. \tag{2.19}$$

Then we can write,

$$Z_i = \tilde{B}_i^{-1} E_i = \begin{pmatrix} M_{i_1} & \\ & M_{i_2} \end{pmatrix}^{-1} E_i \ = \begin{pmatrix} M_{i_1}^{-1} E_i^{[i_1]} \\ M_{i_2}^{-1} E_i^{[i_2]} \end{pmatrix} \equiv \begin{pmatrix} \tilde{Y}_1 \\ \tilde{Y}_2 \end{pmatrix}. \tag{2.20}$$

Thus, we can compute each column j of $\tilde{Y}_1$ and $\tilde{Y}_2$ by $\tilde{Y}_1(:, j) =$ `RMLRSolve(`$i_1,\ e_j^{[i_1]}$`)` and $\tilde{Y}_2(:, j) =$ `RMLRSolve(`$i_2,\ e_j^{[i_2]}$`)`, in which $e_j^{[i_1]}$ and $e_j^{[i_2]}$ are the jth column of $E_i^{[i_1]}$ and $E_i^{[i_2]}$ respectively. Note that this approach requires again a *postorder traversal* processing since when computing Z_i, the low-rank approximations at all i's non-leaf descendants must have already been computed. However, in contrast to the first approach, Z_i is not needed by i's ancestors, so Z_i can be discarded after it is used to obtain its low-rank approximation.

Since we have $Z_i \in \mathbb{R}^{n_i \times n_x}$, in the above method, we need to perform n_x recursive solves in order to compute Z_i. This will be inefficient when $k \ll n_x$. In the third method, we use the Lanczos bidiagonalization method without forming a matrix Z_i to compute the low-rank approximation.

First we briefly review the Lanczos bidiagonalization method. Given a matrix $A \in \mathbb{R}^{m \times n}$ and an arbitrary unit vector v_1, the Lanczos bidiagonalization method [11] (see also [17, 9]) builds two sequences of orthonormal vectors $u_1, u_2, \ldots, u_n$ and $v_1, v_2, \ldots, v_n$, from which we can extract approximations to the left and right singular vectors respectively. In addition, it reduces A to a bidiagonal matrix B, i.e., B is nonzero only on the main diagonal and the first superdiagonal. Let $U = [u_1, \ldots, u_n]$ and $V = [v_1, \ldots, v_n]$. Then we have $U^T A V = B$, where B is bidiagonal and $U^T U = I_m$, $V^T V = I_n$. This method is essentially equivalent to applying the symmetric

Lanczos algorithm to A^TA with the starting vector $q_1 = v_1$, or to AA^T with $q_1 = u_1$, or to a symmetric matrix

$$C = \begin{pmatrix} 0 & A \\ A^T & 0 \end{pmatrix}$$

with a special starting vector, $q_1 = \begin{bmatrix} 0 \\ v_1 \end{bmatrix}$.

Because of the above equivalences and the relationship between the symmetric eigenvalue problem of A^TA, AA^T or C and the SVD of A, the larger singular values of B are typically good approximations to the larger singular values of A. If $B = U_B\Sigma_B V_B^T$ defines the SVD of B, then σ_B, the singular values of B, approximate some of the singular values of A while UU_B and VV_B approximate the corresponding left and right singular vectors. As is well-known, in the presence of rounding error, orthogonality in the Lanczos procedure is quickly lost and a form of reorthogonalization is needed in practice. For simplicity, we use complete reorthogonalization, which is inexpensive when a small number of steps are performed. More efficient reorthogonalization schemes have been proposed, for details, see, e.g., [12, 18, 20], but these will not be considered in this paper.

The Lanczos bidiagonalization method requires the matrix A only in the form of matrix-vector products and matrix-transpose-vector products, i.e., $y = Ax$ and $y = A^Tx$, which is quite appealing in this situation where the matrix is implicitly available and recursively defined. In this approach, we perform the Lanczos bidiagonalization method on Z_i, in which $Z_i = \tilde{B}_i^{-1}E_i$. Therefore, the matrix-vector product and the matrix-transpose-vector product are given by,

$$y = Z_i x = \left(\tilde{B}_i^{-1}E_i\right)x = \tilde{B}_i^{-1}w \quad \text{with} \quad w = E_i x; \tag{2.21}$$

and

$$y = Z_i^T x = \left(\tilde{B}_i^{-1}E_i\right)^T x = E_i^T w \quad \text{with} \quad w^T = x^T\tilde{B}_i^{-1}. \tag{2.22}$$

We can compute $\tilde{B}_i^{-1}w$ in (2.21) by

$$\tilde{B}_i^{-1}w = \begin{pmatrix} M_{i_1}^{-1}w_1 \\ M_{i_2}^{-1}w_2 \end{pmatrix} \equiv \begin{pmatrix} e \\ f \end{pmatrix}, \tag{2.23}$$

in which e and f are given by $e :=$ `RMLRSolve(`i_1`,` w_1`)` and $f :=$ `RMLRSolve(`i_2`,` w_2`)`.

Likewise, we can compute $x^T\tilde{B}_i^{-1}$ in (2.22) by

$$x^T\tilde{B}_i^{-1} = \begin{pmatrix} x_1^TM_{i_1}^{-1} & x_2^TM_{i_2}^{-1} \end{pmatrix} \equiv \begin{pmatrix} g & h \end{pmatrix}, \tag{2.24}$$

where g and h can be computed by a similar function as `RMLRSolve`, which performs a recursive solve on the left. We omit the details of this function.

Note that this third approach based on the Lanczos bidiagonalization method also requires a *postorder traversal* implementation. Indeed, when performing the recursive solve with $\tilde{B}_i$, all low-rank approximations at its descendants must have already been computed.

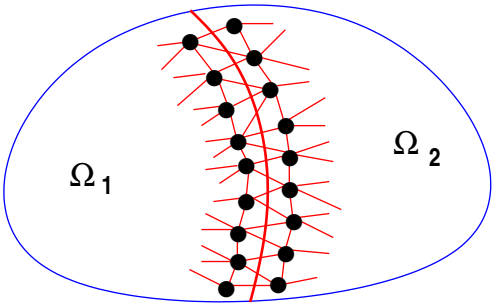

FIG. 2.3. *Illustration of a domain partitioned into two domains (DD with an edge separator). The edge separator and the interface nodes are shown.*

2.5. Generalization: domain decomposition with edge separators. To generalize the above scheme we need to take a domain decomposition viewpoint. We first consider the situation illustrated in Figure 2.3, in which a domain Ω is partitioned into two domains Ω_1 and Ω_2. If we call interior nodes all those nodes inside a domain excluding interface nodes then the variables are naturally partitioned into 4 parts: Interior nodes in Ω_1; Interface points in Ω_1; Interior nodes in Ω_2; Interface points in Ω_2. We refer to these nodes as x_1, y_1, x_2, y_2, respectively, and denote by n_i and m_i the number of interior variables and interface variables in domain Ω_i, for $i = 1, 2$. Therefore, the global matrix representation of the matrix associated with the mesh represented in the figure is as follows,

$$PAP^T = \left(\begin{array}{cc|cc} \hat{B}_1 & \hat{F}_1 & & \\ \hat{F}_1^T & C_1 & & -X \\ \hline & & \hat{B}_2 & \hat{F}_2 \\ & -X^T & \hat{F}_2^T & C_2 \end{array}\right), \tag{2.25}$$

in which $A \in \mathbb{R}^{n\times n}$ and P is the corresponding permutation matrix. The above representation also assumes that the interface nodes in each domain are listed last. The matrix $\hat{B}_i$ is of size $n_i \times n_i$ and the matrix C_i is of size $m_i \times m_i$, for $i = 1, 2$. They represent the interior variables and the interface variables of the two domains respectively. Note that the matrix X represents the coupling between interface variables in one domain with those of the other domain, which is essentially the *edge separator* composed of the edges between y_1 and y_2 and it is of size $m_1 \times m_2$.

Consider the following $n \times m_1$ matrix defined with respect to the above partitioning of the variables:

$$E = \begin{pmatrix} 0 \\ I \\ 0 \\ X^T \end{pmatrix}. \tag{2.26}$$

Then, we have

$$PAP^T = \begin{pmatrix} B_1 & \\ & B_2 \end{pmatrix} - EE^T \quad \text{with} \quad B_i = \begin{pmatrix} \hat{B}_i & \hat{F}_1 \\ \hat{F}_i^T & C_i + D_i \end{pmatrix} \quad \text{and} \quad \begin{cases} D_1 & = I \\ D_2 & = X^T X \end{cases}. \tag{2.27}$$

Since $D_1 \neq D_2$, there is an imbalance between the expressions of B_1 and B_2. However, this is not an issue because X is typically a strongly diagonal dominant (non-square) matrix and as a result one can mitigate the imbalance by weighing the two nonzero terms in (2.26). The matrix E is then redefined as

$$E_\alpha = \begin{pmatrix} 0 \\ \alpha I \\ 0 \\ \frac{X^T}{\alpha} \end{pmatrix}. \tag{2.28}$$

The matrices D_i then become to $D_1 = \alpha^2 I$ and $D_2 = \frac{1}{\alpha^2} X^T X$. Therefore, α should be taken as the square root of the largest singular value of X in order to make the spectral radius of D_1 equal that of D_2.

Finally, the ultimate balancing option consists of using a factorization of X. Recalling that X is of size $m_1 \times m_2$, assume that we have $X = LU$, where $L \in \mathbb{R}^{m_1 \times l}$ and $U \in \mathbb{R}^{l \times m_2}$, in which $l = \min(m_1, m_2)$. Then we can take

$$E_{LU} = \begin{pmatrix} 0 \\ L \\ 0 \\ U^T \end{pmatrix}, \tag{2.29}$$

which leads $D_1 = LL^T$ and $D_2 = U^T U$. Note that now E is of size $n \times l$. A method based on the corresponding modification of the decomposition (2.27) can now be defined following the same steps and definitions as those of Section 2 for RMLR. We will refer to this as the edge-separator-based RMLR (RMLR-E) approach.

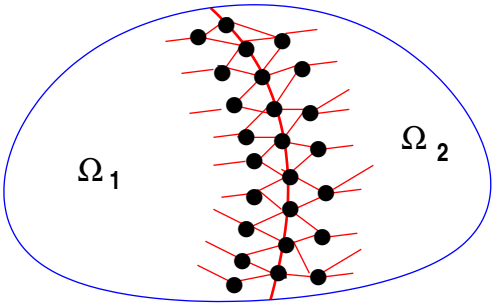

FIG. 2.4. *Illustration of a domain partitioned into two domains (DD with a vertex separator). The vertex separator and the interface nodes are shown.*

2.6. Generalization: domain decomposition with vertex separators. Another way to generalize the RMLR scheme is to exploit a domain decomposition approach based on a *vertex separator*. We consider the situation illustrated in Figure 2.4 in which a domain Ω is partitioned into three parts: domains Ω_1, Ω_2 and the vertex separator S, which is the interface shared by Ω_1 and Ω_2. We now distinguish between 5 different pieces of the variable space: interior nodes in Ω_1 that are not coupled to S; interface nodes in Ω_1 that are coupled to S; interior nodes in Ω_2 that are not coupled to S; interface nodes in Ω_2 that are coupled to S; and the nodes in the vertex separator. We refer to these nodes as x_1, y_1, x_2, y_2, z and denote their sizes by n_1, m_1, n_2, m_2, p respectively. If we assume that the separator nodes are

labeled after the nodes in Ω_1 and Ω_2, then the global matrix representation of the matrix associated with the mesh represented in the figure is as follows,

$$PAP^T = \left(\begin{array}{cc|cc|c} & B_1 & & & \begin{matrix}0\\-F_1\end{matrix} \\ \hline & & & B_2 & \begin{matrix}0\\-F_2\end{matrix} \\ \hline 0 & -F_1^T & 0 & -F_2^T & C \end{array}\right), \tag{2.30}$$

in which $A \in \mathbb{R}^{n\times n}$ and P is the corresponding permutation matrix. For the matrix in (2.30), B_i is of size $(n_i+m_i)\times(n_i+m_i)$, which represents the nodes of domain i. The two zero blocks in the last column of size $n_i \times p$ indicate that interior nodes x_i are not coupled to z. The matrix $-F_i$ is of size $m_i \times p$, which represents the coupling between y_i and z, for $i = 1, 2$. The block C is of size $p \times p$ and represents the vertex separator. Note that F_1 and F_2 are no longer identity matrices and not necessarily square, although they should be close to being square. Consider the following $n \times (m_1 + m_2)$ matrix defined with respect to the above partitioning of the variables:

$$E = \begin{pmatrix} 0 & 0 \\ I & 0 \\ 0 & 0 \\ 0 & I \\ F_1^T & F_2^T \end{pmatrix}. \tag{2.31}$$

Then, it is easy to verify that

$$PAP^T = \begin{pmatrix} \hat{B}_1 & & \\ & \hat{B}_2 & \\ & & \hat{C} \end{pmatrix} - EE^T \qquad \text{with} \quad \hat{B}_i = B_i + \hat{I}_i; \quad \hat{C} = C + F_1^T F_1 + F_2^T F_2,$$

where $\hat{I}_i$ is a matrix consisting of zeros except in the bottom right corner which is an identity matrix of size $m_i \times m_i$, i.e., $\hat{I}_i = \left(\begin{smallmatrix} 0 & 0 \\ 0 & I_{m_i} \end{smallmatrix}\right)$. A method based on the above decomposition can now be defined following the same steps and definitions as those of Section 2 for RMLR. We refer to it as the vertex-separator-based RMLR (RMLR-V) approach.

3. Analysis. We were surprised to find in our experiments that the preconditioners given by (2.11) and (2.14) are identical. In addition, we also noted that the matrix H_k (as well as G_k in some cases) is symmetric which is not immediately obvious. This section explores some of these issues. The main assumption we make in order to prove these results has to do with the form of the rank-k approximation $U_k V_k^T$. We will assume that $U_k V_k^T$ in (2.9) is the best 2-norm rank k approximation to $B^{-1}E$. A second assumption is that we have $V_k^T V_k = I$. We begin with a simple lemma which provides an explicit formula for the inverse of the matrix G_k of (2.10).

LEMMA 3.1. *Let G_k be defined by (2.10) and assume that the matrix $I - U_k^T E V_k$ is nonsingular. Then*

$$G_k^{-1} = I + V_k \hat{H}_k U_k^T E \quad \textit{with} \quad \hat{H}_k = (I - U_k^T E V_k)^{-1}. \tag{3.1}$$

Furthermore, the following relation holds:

$$V_k^T G_k^{-1} V_k = \hat{H}_k, \tag{3.2}$$

i.e., the matrices H_k in (2.12) and the matrix $\hat{H}_k$ given in (3.1) are the same.

Proof. The expression for the inverse of G_k is provided by the Sherman-Morrison formula:

$$(I - V_k(U_k^T E))^{-1} = I + V_k \hat{H}_k U_k^T E \quad \text{with} \quad \hat{H}_k = (I - U_k^T E V_k)^{-1},$$

which is a valid expression under the assumption that $I - U_k^T E V_k$ is nonsingular.

Relation (3.2) follows from the observation that $\hat{H}_k^{-1} + U_k^T E V_k = I$, which yields

$$V_k^T G_k^{-1} V_k = V_k^T (I + V_k \hat{H}_k U_k^T E) V_k = I + \hat{H}_k U_k^T E V_k = \hat{H}_k [\hat{H}_k^{-1} + U_k^T E V_k] = \hat{H}_k.$$

□

Since the matrices $\hat{H}_k$ and H_k are identical we will use the symbol H_k to denote them both in what follows.

Recall that $U_k \in \mathbb{R}^{n \times k}$, $V_k \in \mathbb{R}^{n_x \times k}$. Under the assumptions we have

$$B^{-1}E = U_k V_k^T + Z \quad \text{with} \quad Z V_k = 0 \tag{3.3}$$

where $Z \in \mathbb{R}^{n \times n_x}$. This is because if $B^{-1}E = U \Sigma V^T$ then the best rank-k approximation is given by $\sum_{i \leq k} \sigma_i u_i v_i^T$ and so $Z = \sum_{i>k} \sigma_i u_i v_i^T$. A number of simple properties follow from this observation.

PROPOSITION 3.2. *Assume that the approximation $U_k V_k^T$ in (2.9) is the best 2-norm rank k approximation to $B^{-1}E$ as obtained from the SVD and that we have $V_k^T V_k = I$ and that B is symmetric positive definite. Then we have $U_k^T E V_k = U_k^T B U_k$ and the matrix $U_k^T E V_k$ is therefore symmetric positive definite.*

Proof. We write,

$$U_k^T E V_k = U_k^T B (B^{-1}E) V_k = U_k^T B (U_k V_k^T + Z) V_k = U_k^T B U_k$$

Since B is SPD and U_k is of full rank then it follows that $U_k^T E V_k$ is SPD. □

PROPOSITION 3.3. *Assume that the approximation $U_k V_k^T$ in (2.9) is the best 2-norm rank k approximation to $B^{-1}E$ as obtained from the SVD and that the matrix $I - U_k^T E V_k$ is nonsingular. Then the two preconditioning matrices defined by (2.11) and (2.14), respectively, are equal.*

Proof. Comparing (2.11) and (2.14) we note that all we have to prove is that $B^{-1} E G_k^{-1} V_k U_k^T = U_k V_k^T G_k^{-1} V_k U_k^T$. The proof requires the expression for the inverse of G_k which is provided by Equation (3.1) of Lemma3.1, which is valid under the assumption that $I - U_k^T E V_k$ is nonsingular. From this it follows that:

$$\begin{aligned}
B^{-1} E G_k^{-1} V_k U_k^T &= (U_k V_k^T + Z) G_k^{-1} V_k U_k^T \\
&= (U_k V_k^T + Z) \left[I + V_k H_k U_k^T E \right] V_k U_k^T \\
&= (U_k V_k^T + Z) V_k \left[I + H_k U_k^T E V_k \right] U_k^T \qquad (3.4) \\
&= (U_k V_k^T) V_k \left[I + H_k U_k^T E V_k \right] U_k^T \qquad (3.5) \\
&= (U_k V_k^T) \left[I + V_k H_k U_k^T E \right] V_k U_k^T \qquad (3.6) \\
&= U_k V_k^T G_k^{-1} V_k U_k^T,
\end{aligned}$$

where we have used the relation (3.3) in going from (3.4) to (3.5). □

Proposition 3.2 along with the expressions (2.12) of the preconditioner and (3.1) for H_k lead to yet another way of expressing the preconditioner.

PROPOSITION 3.4. *Under the same assumptions as those of Proposition 3.3, the preconditioner M given by equation (2.12) satisfies the relation:*

$$M = B - BU_k U_k^T B \tag{3.7}$$

Proof. We need to invert the matrix M given in the above expression in order to compare it with (2.12). Using Sherman-Morrison formula leads to the expression,

$$(B - (BU_k)(BU_k)^T)^{-1} = B^{-1} + U_k(I - U_k^T BU_k)^{-1}U_k^T$$

Using the relation $U_k BU_k^T = U_k^T EV_k$ from Proposition 3.2 and the expression of H_k obtained from Lemma 3.1 leads to the same expression as (2.12) for the inverse of M. □

The expression of the preconditioner given by the above proposition provides some insight as to the nature of the preconditioner. Consider the extreme situation when we use the full decomposition $B^{-1}E = U_k V_k^T$, so $k = n_x$ and $Z = 0$ in (3.3). Then, $E = BU_k V_k^T$ and hence we will have $BU_k = EV_k$. Since V_k is now a square (unitary) matrix, therefore,

$$M = B - EV_k V_k^T E^T = B - EE^T,$$

which is the original matrix per (2.6). Not surprisingly, we do indeed obtain an exact preconditioner when an exact decomposition $B^{-1}E = U_k V_k^T$ is used. When an inexact decomposition $B^{-1}E \approx U_k V_k^T$ is used, $(k < n_x, Z \neq 0)$ then we have $B^{-1}E = U_k V_k^T + Z$ and we obtain $(E - BZ) = BU_k V_k^T$. We can now ask what approximation is the preconditioner making on the original matrix in this situation. Remarkably, the preconditioner used simply corresponds to a modification of (2.6) in which E is perturbed by $-BZ$.

PROPOSITION 3.5. *Under the same assumptions as those of Proposition 3.3, the preconditioner M given by equation (2.12) satisfies the relation:*

$$M = B - (E - BZ)(E - BZ)^T, \tag{3.8}$$

where Z is given in (3.3).

Proof. The proof follows immediately from the above arguments, Proposition 3.4, and the equality

$$(E - BZ)(E - BZ)^T = BU_k V_k^T V_k U_k^T B = BU_k U_k^T B.$$

□

The final question we would like to answer now is: Under which condition is the preconditioner symmetric positive definite? The following result gives a necessary and sufficient condition. In the following we will say that the preconditioner (2.12) is *well-defined* when H_k exists, i.e., when $I - U_k^T EV_k$ is nonsingular.

THEOREM 3.6. *Let the assumptions of Proposition 3.2 be satisfied. Then the preconditioner given by (2.12) is well defined and symmetric positive definite if and only if $\rho(U_k^T EV_k) < 1$.*

Proof. Recall from Proposition (3.2) that the matrix $U_k^T EV_k$ is a symmetric positive definite matrix. If $\rho(U_k^T EV_k) < 1$ then clearly the eigenvalues of $I - U_k^T EV_k$ are all positive. Therefore, the matrix $I - U_k^T EV_k$ is symmetric positive definite and

it is nonsingular so the preconditioner M is well defined. Its inverse, H_k is SPD. As a result the matrix M in (2.12) is also SPD.

To prove the converse, we consider expression (3.7) of the preconditioner and make the assumption that it is positive definite. Under this assumption, $U_k^T M U_k$ is SPD since U_k is of full rank. Now observe that if we set $S \equiv U_k^T B U_k$ then

$$U_k^T M U_k = U_k^T B U_k - U_k^T B U_k U_k^T B U_k = S - S^2.$$

The eigenvalues of $S - S^2$ are positive. Since S is symmetric positive definite any eigenvalue λ of S is positive. Therefore λ is positive and such that $\lambda - \lambda^2$ is also positive. This implies that $0 < \lambda < 1$ and the proof is complete since $U_k^T E V_k = U_k^T B U_k = S$ (Proposition 3.2). □

Finally, the above theorem can be extended to the recursive multilevel preconditioner which is defined by (2.17). The following result gives a sufficient condition of M_i to be symmetric positive definite. We define $s(i)$ to be the set consisting of i and all its non-leaf descendants, i.e.,

$$s(i) = \{x \mid x = i \text{ or } x \text{ is a descendant of } i \text{ and } x \text{ is not a leaf node}\},$$

and $t(i)$ to be the set consisting of all the leaf descendants of i, i.e.,

$$t(i) = \{x \mid x \text{ is a descendant of } i \text{ and } x \text{ is a leaf node}\}.$$

COROLLARY 3.7. *Assume that the approximation $U_i V_i^T$ is the best 2-norm rank k approximation to $\tilde{B}_i^{-1} E_i$ as obtained from the SVD where $\tilde{B}_i$ is defined in* (2.19) *and that we have $V_i^T V_i = I$ and the matrices A_j for all $j \in t(i)$ are symmetric positive definite. Then the preconditioner M_i given by* (2.17) *is well defined and symmetric positive definite if $\rho(U_j^T E_j V_j) < 1$ for all $j \in s(i)$.*

Proof. The proof consists of a simple inductive argument which exploits the previous result which is valid for two levels. □

4. Numerical Experiments. A preliminary implementation of the above preconditioning method has been written and tested in MATLAB® [23]. This section first provides numerical results when solving linear systems from two-dimensional elliptic partial differential equations (PDEs) using the RMLR method along with Krylov subspace methods. In these problems, a recursive geometric bisection is used for the domain decomposition in the RMLR approach. A 2D regular grid is simply cut in half along either x or y dimension, depending on which dimension of the grid is shorter. We then extend the RMLR method to solve a problem from a three-dimensional PDE, in which a similar 3D recursive geometric bisection is used. Finally, the RMLR method is tested for solving a sequence of general symmetric linear systems. For these problems, a graph partitioning algorithm `PartGraphRecursive` from METIS [13] is used to bisect the graph as needed in the RMLR method.

The RMLR strategy is compared against the incomplete Cholesky factorization with threshold dropping (ICT) or the incomplete LU factorization with threshold dropping and partial pivoting (ILUTP) by using the functions `cholinc` or `luinc` from MATLAB. The accelerators used in the following runs include the conjugate gradient (CG) method, the minimal residual (MINRES) method and the restarted generalized minimal residual (GMRES) method with a restart dimension of 40. The iteration is stopped whenever the residual norm has been reduced by 8 orders or the maximum number of iterations allowed, which is 400, is exceeded.

In all the following experimental results, we only list and compare the number of iterations and the fill-ratio, i.e., the ratio of the number of nonzeros required to store a preconditioner to the number of nonzeros in the original matrix. The CPU time for building the RMLR preconditioner and applying it in the iterations is not presented since the current implementation cannot fairly represent the time cost of the RMLR strategy, especially for the aimed multi-core CPU or many-core GPU platforms. This comparison is left for future work in which efficient implementations of the RMLR techniques in C or CUDA for NVIDIA GPUs will be tested.

4.1. 2D elliptic PDE. We will examine a two-dimensional elliptic partial differential equation

$$-a\frac{\partial^2 u}{\partial x^2} - b\frac{\partial^2 u}{\partial y^2} = -\left(bx^2 + ay^2\right) e^{xy} \text{ in } \Omega, \tag{4.1}$$

subject to the Dirichlet boundary condition $u = e^{xy}$ on $\partial\Omega$, where $\Omega = (0,1) \times (0,1)$. So the exact solution of (4.1) is $u = e^{xy}$. We solve an anisotropic problem with $a = 20$ and $b = 1$ and we take the 5-point centered difference approximation on an $n_x \times n_y$ grid. The coefficient matrix in the linear system from (4.1) is symmetric positive definite. Recall from Corollary 3.7 in Section 3, that if we have $\rho(U_i^T EV_i) < 1$ for all i, then for an SPD matrix A, the RMLR preconditioner defined in (2.17) is also SPD. Therefore, we can use the RMLR-E preconditioning method along with the CG method for solving the above problem. The Lanczos bidiagonalization method is used to compute the low-rank approximations.

In this example, we take a grid of size $n_x = n_y = 130$, so the coefficient matrix is of order $16,384 \times 16,384$. Numerical experiments were carried out to compare the performance of the RMLR-E method with that of the ICT method along with the CG method. Figure 4.1 shows the convergence profile for ICT-CG and RMLR-E-CG. The y-axis shows the residual norm (in log scale) and the x-axis shows the iteration count. The number at the end of each curve shows the fill-ratio. For the RMLR-E method, the number of levels is 2 and the rank k is set to 2, 3 or 5. The dropping tolerance for ICT is 10^{-3} or 2×10^{-3}. The results indicate that the RMLR-E preconditioner can be used along with the CG method to solve this problem successfully. Furthermore, it performs better than the ICT preconditioner in that it achieves a faster convergence at a lower memory cost, as indicated by the lower fill-ratio.

Table 4.1 presents results of the RMLR-E preconditioning method along with the CG method for solving the same problem with different ranks k and numbers of levels $nlev$. The table shows the number of iterations (the first number) and the fill-ratio (the second number). From results in the table, first we can see that the RMLR-E method with a smaller $nlev$ or a larger k exhibits a faster convergence rate. Second, we find that fill-ratios grow as k increases and they also grow as $nlev$ increases because more levels of low-rank approximation matrices are saved.

4.2. 2D Helmholtz-type equation. Experiments were conducted for solving the following 2D Helmholtz-type equation,

$$-\frac{\partial^2 u}{\partial x^2} - \frac{\partial^2 u}{\partial y^2} - \rho u = -6 - \rho\left(2x^2 + y^2\right) \text{ in } \Omega, \tag{4.2}$$

subject to the Dirichlet boundary condition $u = 2x^2 + y^2$ on $\partial\Omega$, where $\Omega = (0,1) \times (0,1)$. The exact solution of (4.2) is $u = 2x^2 + y^2$. We take the 5-point centered difference approximation on an $n_x \times n_y$ grid. The test matrix originates from the

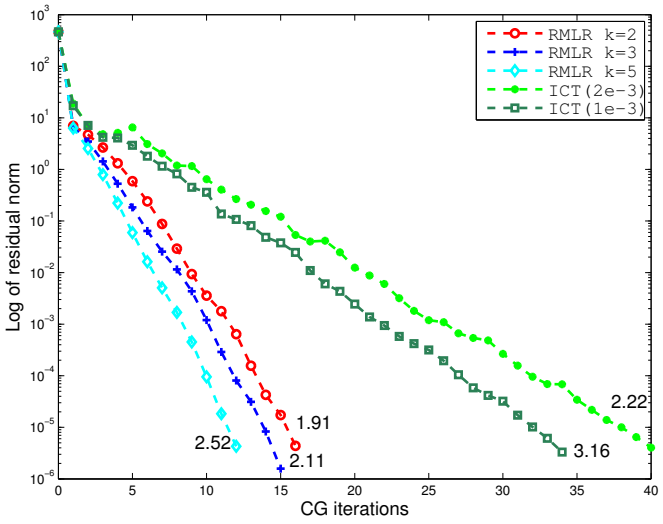

FIG. 4.1. *Comparison between the* ICT *method and the* RMLR *method for solving a 2D elliptic equation in* (4.1) *along with the* CG *method.*

TABLE 4.1
Experimental results for solving a 2D elliptic equation in (4.1)*:* CG *iteration counts with the* RMLR-E *method and fill ratios.*

k	nlev=4		nlev=3		nlev=2	
2	69	2.51	59	2.23	16	1.91
3	61	3.12	53	2.64	15	2.11
4	55	3.72	49	3.04	13	2.32
5	50	4.33	45	3.44	12	2.52
6	46	4.94	41	3.85	12	2.72

discretization of a negative Laplacean, shifted by $\rho h^2 I$, in which the same mesh size $h = \frac{1}{n_x - 1}$ is used in both x and y directions. The shift can make the problem substantially indefinite. Hence we use the RMLR method along with GMRES(40) to solve the problem and then compare its performance with that of the nonsymmetric ILUTP preconditioner.

In the first example, we take $n_x = n_y = 66$ and $\rho = 845$, so the dimension of the coefficient matrix is 4096×4096 and the corresponding shift is $0.2I$. First, we present numerical results of the RMLR-E method by using the three different approaches discussed in Section 2.4 to compute the low-rank approximations. The three approaches are referred to as the `exact-svd` approach, the `approx-svd` approach and the `approx-lan`(m) approach respectively. The `exact-svd` approach computes matrix $B_i^{-1}E$ exactly while the `approx-svd` approach computes its approximation $\tilde{B}_i^{-1}E$. Then both methods compute the partial SVD using `svds` from MATLAB and select the singular vectors associated with the largest k singular values. On the other hand, the `approx-lan`(m) approach, without computing matrix $\tilde{B}_i^{-1}E_i$, applies the Lanczos bidiagonalization method implicitly to $\tilde{B}_i^{-1}E_i$, where m is the number of Lanczos steps. This approach can be much more efficient since it usually can provide

results that are as accurate as the `approx-svd` approach with a small number of Lanczos steps. As was seen above, the `exact-svd` approach is not computationally feasible for large problems and it is used here only as a reference for the other two methods. Figure 4.2 compares the performance of the RMLR-E methods using these three approaches for solving the linear system along with GMRES(40). The y-axis shows the residual norm (in log scale) and the x-axis shows the iteration count. The fill-ratio is 5.38. As shown in Figure 4.2, the `approx-svd` approach exhibits a performance that is similar to that of the `exact-svd` approach. Furthermore, when approximating 3 largest singular values, `approx-lan(5)` can provide a result that is closed to that of the `approx-svd` approach, but `approx-lan(10)` is accurate enough to provide the same convergence. Therefore, in what follows, we use the Lanczos bidiagonalization method to compute the low-rank approximations.

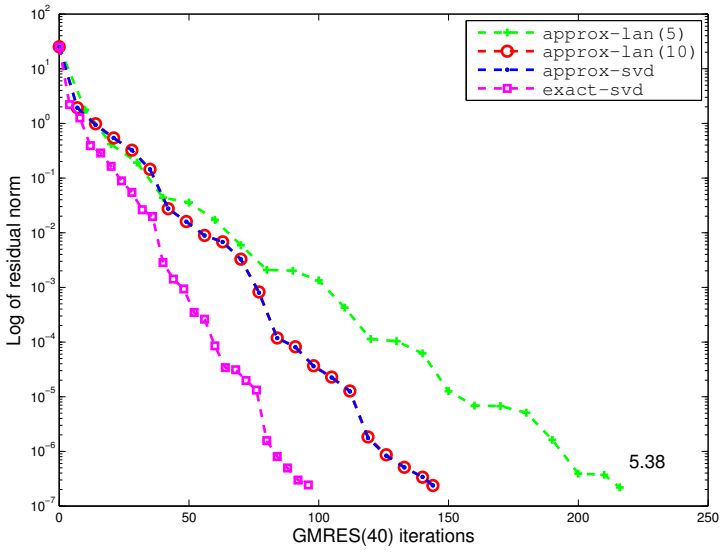

FIG. 4.2. *Computing the low-rank approximations in* RMLR-E ($k = 3$) *by the* SVD *vs. the Lanczos bidiagonalization method.*

Next, we compare the performance of the RMLR-E method with that of the ILUTP method along with GMRES(40) for solving the above problem. Figure 4.3 shows the convergence profile of ILUTP-GMRES and RMLR-E-GMRES. For the RMLR-E preconditioner, the size of the last-level matrices is 64 so that the number of levels is 7 and the rank k is set to 2, 3 or 5. The dropping tolerance for ILUTP is 10^{-2}. From Figure 4.3, we can see that ILUTP-GMRES does not converge with a fill-ratio as high as 8.47, while the RMLR-E method starts converging for a rank k as small as 2 and the corresponding fill-ratio is 4.36. The storage required increases with the rank k and it remains reasonable for $k = 3$. In addition, the low-rank approximation matrices to be stored are dense and there are possibilities that can be explored to compress them.

Table 4.2 presents results of the RMLR-E preconditioning method for different ranks k and numbers of levels $nlev$. The table shows the number of iterations (the first number) and the fill-ratio (the second number). Several observations can be made

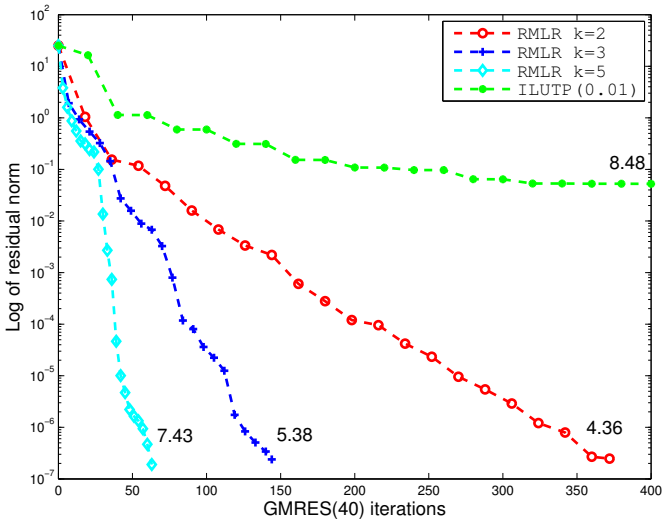

FIG. 4.3. *Comparison between the* ILUTP *method and the* RMLR-E *method for solving a 2D Helmholtz-type equation in* (4.2) *along with* GMRES(40).

from this table:

1. The fill-ratios grow as k increases.
2. When k is small, say $k = 2, 3$, the fill-ratio increases as $nlev$ decreases, or in other words, as the size of the last-level matrix increases.
3. In contrast, when k is large, say $k = 5, 6$, the fill-ratio, decreases as $nlev$ decreases.
4. The RMLR method with a smaller $nlev$ or a larger k performs better in terms of the convergence rate.

Table 4.3 presents the memory requirement for the RMLR-E method with $k = 2$ and $k = 6$. 'Lrk' indicates the memory cost for the low-rank approximations. The number of nonzeros in low-rank approximation matrices and its percentage of the total number of nonzeros in the preconditioner are shown. In addition, 'Fact' indicates the number of nonzeros in the factors from the last-level factorizations. Results of the table can explain the observations 2 and 3 above. In the cases when k is small, the overall fill-ratio is dominated by the number of nonzeros in last-level factorizations, which will be higher, in general, when last-level matrices become larger. Therefore, the overall fill-ratio increases as $nlev$ decreases. On the other hand, when k is large, the overall fill-ratio is dominated by the memory cost of the low-rank approximations which increases with $nlev$.

In fact, we found that when $nlev = 7$, all matrices A_i at the last level are SPD, so the factorizations are performed by IC, while ILU is used for $nlev \leq 6$ instead. Note that the coefficient matrix from this problem is indefinite but from (2.15), we can see that A_{i_1} and A_{i_2} are perturbed and as a result become less indefinite than A_i. Therefore, when an RMLR preconditioner has enough levels, matrices at the last level may be positive definite even though the original problem is indefinite. This makes the incomplete factorizations at the last level more robust. Moreover, we showed in Section 3 that if all matrices at the last level are SPD and $\rho(U_i^T E_i V_i) < 1$ is satisfied

TABLE 4.2
Experimental results for solving a 2D Helmholtz-type equation in (4.2)*:* GMRES(40) *iteration counts with the* RMLR-E *method and fill ratios.*

k	nlev=7		nlev=6		nlev=5		nlev=4		nlev=3	
2	318	3.56	372	4.36	261	4.77	183	4.80	47	5.53
3	192	4.78	144	5.38	144	5.59	102	5.41	38	5.94
4	181	6.03	132	6.41	74	6.41	45	6.02	35	6.35
5	75	7.20	63	7.43	39	7.22	33	6.63	31	6.76
6	45	8.52	41	8.46	35	8.04	29	7.24	28	7.16

TABLE 4.3
Memory requirement for the RMLR-E method with $k = 2$ *and* $k = 6$.

k	nlev=7		nlev=6		nlev=5		nlev=4		nlev=3	
	Lrk	Fact	Lrk	Fact	Lrk	Fact	Lrk	Fact	Lrk	Fact
2	49k (69%)	23k	41k (47%)	47k	33k (34%)	64k	24k (25%)	72k	16k (15%)	96k
6	149k (87%)	23k	124k (72%)	47k	99k (61%)	64k	74k (51%)	72k	49k (34%)	96k

for every non-leaf node i, then the RMLR preconditioner is SPD. This implies that, in this case, we can use the RMLR method along with iterative methods for symmetric indefinite systems, for instance, the MINRES method or the SYMMLQ method. An example of using the RMLR-E method along with the MINRES method for solving a indefinite linear system from a Helmholtz-type problem will be shown later in this section. Another advantage of factoring matrices at the last level by IC is that we can save about half the memory needed for storing the factors.

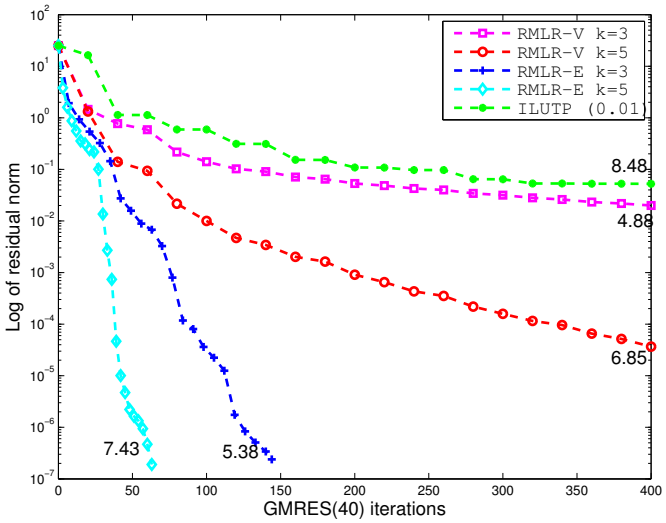

FIG. 4.4. *Comparison between the* RMLR-E *method and* RMLR-V *method for solving a 2D Helmholtz-type equation in* (4.2).

TABLE 4.4

Experimental results for solving a 2D Helmholtz-type equation in (4.2)*: ρ_{max},* MINRES *iteration counts with the* RMLR *method and fill ratios.*

k	nlev=7			nlev=8			nlev=9		
2	0.9414	103	3.56	0.8771	103	3.72	0.7900	102	4.30
3	1.0118		F	0.9164	96	5.17	0.9186	98	5.71
4	1.0594		F	0.9437	83	6.63	0.9437	80	7.41
5	1.2350		F	2.3190		F	2.3190		F
6	1.3571		F	1.0396		F	1.0396		F

Finally in this example, we compare the RMLR-E method and the RMLR-V method. We find that for all tested cases, the RMLR-E method performs much better than the RMLR-V method. Figure 4.4 gives a comparison between the performance of these two methods with $nlev = 6$ and $k = 3, 5$ for solving the above problem along with GMRES(40). As shown, for both cases, the RMLR-V method failed to converge. From (2.26) and (2.31), we can see that the rank of the matrix $B^{-1}E$ in the RMLR-V method is at least twice as that in the RMLR-E method. As a result, the low-rank approximation in the RMLR-V method is less accurate than that in the RMLR-E method. Therefore, in what follows, we will only test the RMLR-E method and refer to the RMLR-E method as the RMLR method for convenience.

In the second example, we take $n_x = n_y = 66$ and $\rho = 211.25$, so the dimension of the coefficient matrix is 4096×4096 and the corresponding shift is $0.05I$. Accordingly, the linear system is less indefinite and the RMLR preconditioner built for this problem is SPD. Therefore, we can use it along with the MINRES method for solving this symmetric indefinite linear system. In Table 4.4, we show the performance of the RMLR method with different k and $nlev$. For $nlev = 7, 8, 9$, all matrices at the last level are SPD and they are factored by IC. Table 4.4 lists ρ_{max} (the first number), where $\rho_{max} = \max_i \{\rho(U_i^T E_i V_i)\}$, the number of iterations (the second number) and the fill-ratio (the third number). The symbol F indicates that MINRES failed due to the preconditioner not being SPD (MATLAB convergence flag = 5). We showed in Section 3 that in a case when all matrices at the last level are SPD, the RMLR preconditioner is SPD if $\rho(U_i^T E_i V_i) < 1$ for every non-leaf node i. This agrees with our numerical results since in all failed cases, we find that $\rho_{max} > 1$. However, the RMLR method works for all cases when using along with the restarted GMRES method. In addition, results from the table also indicate that k and $nlev$ have an effect on positive definiteness of the preconditioner. The preconditioner tends to be indefinite as k increases. However, $nlev$ has the opposite effect, i.e., increasing it makes the preconditioner more likely to be positive definite.

4.3. 3D Helmholtz-type equation. In this section, we present several numerical results for solving a three-dimensional elliptic partial differential equation,

$$-\frac{\partial^2 u}{\partial x^2} - \frac{\partial^2 u}{\partial y^2} - \frac{\partial^2 u}{\partial z^2} - \rho u = -6 - \rho\left(x^2 + y^2 + z^2\right) \text{ in } \Omega, \tag{4.3}$$

subject to the Dirichlet boundary condition $u = x^2 + y^2 + z^2$ on $\partial\Omega$, where $\Omega = (0,1) \times (0,1) \times (0,1)$. So the exact solution of (4.3) is $u = x^2 + y^2 + z^2$. We take the 7-point centered difference approximation on an $n_x \times n_y \times n_z$ grid. In a case when $n_x = n_y = n_z$, the matrix originating from the discretization is a negative Laplacean matrix shifted by $\rho h^2 I$, where $h = \frac{1}{n_x - 1}$. We use the RMLR method along

TABLE 4.5

Experimental results for solving a 3D Helmholtz-type equation in (4.3)*:* GMRES(40) *iteration counts with the* RMLR *method and the* ILUTP *method, and fill ratios.*

RMLR							ILUTP			
k	nlev=6		nlev=5		nlev=4		tol=5.8e-3		tol=6e-3	
2	377	5.49	177	6.66	114	8.46	F	14.64	F	11.60
4	293	6.97	138	7.84	88	9.35				
6	187	8.46	101	9.03	73	10.23				
8	116	9.95	78	10.22	51	11.12				

with GMRES(40) to solve this indefinite problem and then compare the results with those of the ILUTP preconditioner. In 3D problems, interface nodes contain a larger percentage of the total nodes than 2D problems. Therefore, higher ranks might be needed in the approximations. In the following example, we take $n_x = n_y = n_z = 26$ and $\rho = 312.5$ so that correspondingly, the dimension of the coefficient matrix is $13,824 \times 13,824$ and the shift is $0.5I$. Table 4.5 presents numerical results of the RMLR method for different ranks k and numbers of levels $nlev$ and the ILUTP method. The dropping tolerance for ILUTP is 5.8×10^{-3} or 6×10^{-3}. In the table, we show the number of iterations (the first number) and the fill-ratio (the second number). The results in the table indicate that fill-ratios grow as k increases. On the other hand, fill-ratios grow as $nlev$ decreases since in this 3-D problem, for all k, the overall fill-ratio is dominated by the number of nonzeros in last-level factorizations. Therefore, the fill-ratio increases as sizes of the last-level matrices increase. In addition, similar to results shown in previous experiments, the RMLR method with a smaller $nlev$ or a larger k performs better in terms of the convergence rate. At last, by comparing results of the RMLR method with those of the ILUTP method in the table, we can see that the RMLR method exhibits superior performance for this problem. For all tests, the RMLR method can achieve convergence, whereas the ILUTP method failed even though it has higher fill-ratios.

4.4. General matrices. We selected 17 matrices from the University of Florida sparse matrix collection [8] and a matrix from a structural shell problem for the following tests. Among these 7 are real symmetric positive definite matrices and 11 are real symmetric indefinite matrices. Table 4.6 lists the name, order (N), number of nonzeros (NNZ), the positive definiteness, and a short description for each matrix. If the actual right-hand-side is not provided, the linear system is obtained by creating the artificial right-hand-side $b = Ae$, where e is the vector of all ones.

For solving the SPD systems, the RMLR method is used along with the CG method and compared with the ICT method, whereas for solving the indefinite systems, it is used along with GMRES(40) and compared with the ILUTP method. Table 4.7 presents performance results for these methods. In this table, we show the number of levels $nlev$, the rank k for the RMLR method, fill-ratios and the numbers of iterations. `F` indicates non-convergence in the maximum allowed number of steps. We can see that the RMLR method performs better than the ICT method or the ILUTP method for all cases. It requires lower fill-ratios but can achieve convergence in fewer iterations. In many cases, ICT or ILUTP failed to converge.

We note here that the ranks k used for many problems are much larger than the ones for the above structured problems. This is because in general graphs, the ratio

TABLE 4.6
Name, order (N), number of nonzeros (NNZ) and positive definiteness of the test matrices.

MATRIX	N	NNZ	SPD	DESCRIPTION
FIDAP/ex10	2,410	54,840	yes	CFD problem
FIDAP/ex10hs	2,548	57,308	yes	CFD problem
HB/bcsstk24	3,562	159,910	yes	Structural engineering
HB/bcsstk28	4,410	219,024	yes	Solid element model
Cylshell/s3rmt3m1	5,489	217,669	yes	FEM, cylindrical shells
Cylshell/s3rmt3m3	5,357	207,123	yes	FEM, cylindrical shells
Boeing/bcsstk38	8,032	355,460	yes	Stiffness matrix
HB/bcsstm27	1,224	56,126	no	Buckling analysis
HB/bcspwr06	1,454	5,300	no	Power network problem
HB/bcspwr07	1,612	5,824	no	Power network problem
HB/bcspwr08	1,624	6,050	no	Power network problem
HB/blckhole	2,132	14,872	no	Structural Engineering
HB/jagmesh3	1,089	7,361	no	FEM, model problem
Boeing/nasa1824	1,824	39,208	no	Structural problem
AG-Monien/3elt1_dual	9,000	26,556	no	2D finite element problem
AG-Monien/airfoil1_dual	8,034	23,626	no	2D finite element problem
AG-Monien/ukerbe1_dual	1,866	7,076	no	2D finite element problem
SHELL/COQUE8E3	8,073	196,295	no	Structural problem (shell)

TABLE 4.7
Experimental results for solving general symmetric linear systems: the number of levels (nlev), rank k, fill ratios and CG/GMRES(40) *iteration counts with the* RMLR *method and the* ICT/ILUTP *method.*

MATRIX	RMLR				ICT/ILUTP	
	nlev	k	fill-ratio	#its	fill-ratio	#its
FIDAP/ex10	3	4	0.7	220	1.4	F
FIDAP/ex10hs	3	4	0.7	151	1.2	F
HB/bcsstk24	3	50	2.6	149	4.2	348
HB/bcsstk28	3	60	2.5	127	2.5	204
Cylshell/s3rmt3m1	3	50	2.6	213	2.8	F
Cylshell/s3rmt3m3	4	50	2.9	127	3.2	249
Boeing/bcsstk38	3	40	2.6	112	2.6	F
HB/bcsstm27	4	50	1.8	26	2.3	73
HB/bcspwr06	4	5	3.1	6	5.2	F
HB/bcspwr07	5	5	3.2	6	4.8	F
HB/bcspwr08	4	5	2.1	17	5.8	F
HB/blckhole	5	50	12.8	32	21.8	F
HB/jagmesh3	4	5	5.9	30	9.7	111
Boeing/nasa1824	4	60	3.6	116	4.9	150
AG-Monien/3elt_dual	6	5	9.3	12	13.9	F
AG-Monien/airfoil1_dual	6	5	9.5	5	12.7	F
AG-Monien/ukerbe1_dual	4	5	9.1	25	10.5	F
SHELL/COQUE8E3	3	70	5.0	83	5.06	F

of the number of interface nodes to the number of the total nodes is usually much higher than that in regular grids. Therefore, we need a higher-rank approximation to reach a good enough accuracy.

5. Conclusions. Multilevel preconditioners based on recursive low-rank approximations can be quite effective alternatives to ILU preconditioners for highly indefinite linear systems. We focused on two procedures, referred to as 'two-sided low rank approximation' and 'one-sided low rank approximation' respectively. These two procedures were shown to be mathematically identical. The low-rank approximation is computed from singular vectors or Lanczos vectors. The RMLR preconditioner was shown to be SPD under certain condition even if the original matrix is indefinite and it was used along with the CG method or the MINRES method to solve linear systems successfully. Experimental results show that the proposed preconditioner is efficient and quite robust for indefinite 2D or 3D Helmholtz-type problems, in situations where the standard ILUTP preconditioner usually fails. This scheme has been generalized using a domain decomposition approach leading to two distinct schemes for general sparse matrices. Numerical results show that the RMLR scheme is effective for general symmetric indefinite linear systems. Though this paper compared only convergence rates and memory requirements from a serial MatLab implementation, the preconditioner presented in this paper is especially suitable for highly parallel platforms. Work remains to be done to test the class of preconditioners presented here in these environments.

REFERENCES

[1] Marco Ament, Gunter Knittel, Daniel Weiskopf, and Wolfgang Strasser, *A parallel preconditioned conjugate gradient solver for the poisson problem on a multi-GPU platform*, in PDP '10: Proceedings of the 2010 18th Euromicro Conference on Parallel, Distributed and Network-based Processing, Washington, DC, USA, 2010, IEEE Computer Society, pp. 583–592.

[2] Nathan Bell and Michael Garland, *Implementing sparse matrix-vector multiplication on throughput-oriented processors*, in SC '09: Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis, New York, NY, USA, 2009, ACM, pp. 1–11.

[3] Jeff Bolz, Ian Farmer, Eitan Grinspun, and Peter Schröoder, *Sparse matrix solvers on the GPU: conjugate gradients and multigrid*, ACM Trans. Graph., 22 (2003), pp. 917–924.

[4] Sabine Le Borne and Lars Grasedyck, *H-matrix preconditioners in convection-dominated problems*, SIAM. J. Matrix Anal. Appl., 27 (2006), pp. 1172–1183.

[5] E. Chow and Y. Saad, *Approximate inverse techniques for block-partitioned matrices*, SIAM Journal on Scientific Computing, 18 (1997), pp. 1657–1675.

[6] ———, *Approximate inverse preconditioners via sparse-sparse iterations*, SIAM Journal on Scientific Computing, 19 (1998), pp. 995–1023.

[7] Thomas H. Cormen, Clifford Stein, Ronald L. Rivest, and Charles E. Leiserson, *Introduction to Algorithms*, McGraw-Hill Higher Education, 2nd ed., 2001.

[8] Timothy A. Davis, *University of florida sparse matrix collection, na digest*, 1994.

[9] James Demmel, Jack Dongarra, Axel Ruhe, and Henk van der Vorst, *Templates for the solution of algebraic eigenvalue problems: a practical guide*, Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2000.

[10] Björn Engquist and Lexing Ying, *Sweeping preconditioner for the helmholtz equation: Hierarchical matrix representation*, Communications on Pure and Applied Mathematics, 64 (2011), pp. 697–735.

[11] Gene H. Golub and Charles F. Van Loan, *Matrix computations (3rd ed.)*, Johns Hopkins University Press, Baltimore, MD, USA, 1996.

[12] Joseph Frank Grcar, *Analyses of the lanczos algorithm and of the approximation problem in richardson's method*, PhD thesis, Champaign, IL, USA, 1981. AAI8203472.

[13] George Karypis and Vipin Kumar, Metis - *a software package for partitioning unstructured*

graphs, partitioning meshes, and computing fill-reducing orderings of sparse matrices, version 4.0, tech. report, University of Minnesota, Department of Computer Science / Army HPC Research Center, 1998.

[14] Ruipeng Li, Hector Klie, Hari Sudan, and Yousef Saad, *Towards realistic reservoir simulations on manycore platforms*, SPE Journal, (2010), pp. 1–23.

[15] R. Li and Y. Saad, *GPU-accelerated preconditioned iterative linear solvers*, Tech. Report umsi-2010-112, Minnesota Supercomputer Institute, University of Minnesota, Minneapolis, MN, 2010.

[16] NVIDIA, *CUSPARSE Library User Guide*, 2012.

[17] Beresford N. Parlett, *The Symmetric Eigenvalue Problem*, Society for Industrial and Applied Mathematics, Philadephia, PA, 1998.

[18] B. N. Parlett and D. S. Scott, *The lanczos algorithm with selective orthogonalization*, Mathematics of Computation, 33 (1979), pp. pp. 217–238.

[19] Y. Saad and B. Suchomel, *ARMS: An algebraic recursive multilevel solver for general sparse linear systems*, Numerical Linear Algebra with Applications, 9 (2002).

[20] Horst D. Simon, *The lanczos algorithm with partial reorthogonalization*, Mathematics of Computation, 42 (1984), pp. pp. 115–142.

[21] H. Sudan, H. Klie, R. Li, and Y. Saad, *High performance manycore solvers for reservoir simulation*, in 12th European Conference on the Mathematics of Oil Recovery, 2010.

[22] J. Tang and Y. Saad, *Domain-decomposition-type methods for computing the diagonal of a matrix inverse*, Tech. Report umsi-2010-114, Minnesota Supercomputer Institute, University of Minnesota, Minneapolis, MN, 2010. To appear in SISC.

[23] MatLab, *version 7.10.0 (R2010a)*, The MathWorks Inc., Natick, Massachusetts, 2010.

[24] Mingliang Wang, Hector Klie, Manish Parashar, and Hari Sudan, *Solving sparse linear systems on nvidia tesla GPUs*, in ICCS '09: Proceedings of the 9th International Conference on Computational Science, Berlin, Heidelberg, 2009, Springer-Verlag, pp. 864–873.

[25] Shen Wang, Maarten V. de Hoop, and Jianlin Xia, *On 3d modeling of seismic wave propagation via a structured parallel multifrontal direct helmholtz solver*, Geophysical Prospecting, 59 (2011), pp. 857–873.

[26] Jianlin Xia, Shivkumar Chandrasekaran, Ming Gu, and Xiaoye S. Li, *Fast algorithms for hierarchically semiseparable matrices*, Numerical Linear Algebra with Applications, 17 (2010), pp. 953–976.

[27] Jianlin Xia and Ming Gu, *Robust approximate Cholesky factorization of rank-structured symmetric positive definite matrices*, SIAM J. MATRIX ANAL. APPL., 31 (2010), p. 28992920.