SIAM J. SCI. COMPUT. © 2013 Society for Industrial and Applied Mathematics
Vol. 35, No. 4, pp. A2069–A2095

DIVIDE AND CONQUER LOW-RANK PRECONDITIONERS FOR SYMMETRIC MATRICES[*]

RUIPENG LI[†] AND YOUSEF SAAD[†]

Abstract. This paper presents a preconditioning method based on an approximate inverse of the original matrix, computed recursively from a multilevel low-rank (MLR) expansion approach. The basic idea is to recursively divide the problem in two and apply a low-rank approximation to a matrix obtained from the Sherman–Morrison formula. The low-rank expansion is computed by a few steps of the Lanczos bidiagonalization procedure. The MLR preconditioner has been motivated by its potential for exploiting different levels of parallelism on modern high-performance platforms, though this feature is not yet tested in this paper. Numerical experiments indicate that, when combined with Krylov subspace accelerators, this preconditioner can be efficient and robust for solving symmetric 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

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

DOI. 10.1137/120872735

1. Introduction.

Krylov subspace methods preconditioned with a form of incomplete LU (ILU) factorization can be quite effective in solving symmetric linear systems but there are situations where these preconditioners will not perform well. For instance when the matrix is highly indefinite, an ILU-type 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, these preconditioners are ruled out for computers equipped with massively parallel coprocessors like GPUs.

These situations have motivated the development of a class of approximate inverse preconditioners in the 1990s as alternatives. 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 into two subdomains, then one can get the inverse of the matrix associated with the whole domain by the inverses of the matrices associated with both subdomains plus a low-rank correction. In fact, the rank needed to obtain

[*]Submitted to the journal's Methods and Algorithms for Scientific Computing section April 9, 2012; accepted for publication (in revised form) June 12, 2013; published electronically August 13, 2013. This work was supported by DOE under grant DOE/DE-FG-08ER25841, by NSF under grant NSF/DMS-1216366, and by the Minnesota Supercomputing Institute.
http://www.siam.org/journals/sisc/35-4/87273.html

[†]Computer Science & Engineering, University of Minnesota, Minneapolis, MN 55414 (rli@cs.umn.edu, saad@cs.umn.edu).

the exact inverse is equal to the number of interface nodes in a subdomain, i.e., the nodes which are connected to some node in the other subdomain. However, a decent approximation suitable for preconditioning can often be obtained from a much smaller rank. The idea then is to exploit this method in a multilevel fashion, wherein a domain is recursively divided into two for the next level until either a certain number of levels is reached or small enough subdomains are obtained. We refer to a preconditioner obtained by this approach as a recursive multilevel low-rank (MLR) 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 [23]. On the negative side, this new approach may lead to some obvious difficulties in implementation and higher construction cost. Exploiting recursivity as much as possible simplifies the implementation. On the other hand, this method appears to be simpler than those based on hierarchical matrices ($\mathcal{H}$-matrices), where approximations to LU factorizations or inverse matrices are obtained by similarly representing certain off-diagonal blocks by low-rank matrices; see, e.g., [17, 18].

The potential advantages of MLR preconditioners outnumber its disadvantages. First, the preconditioner is very suitable for single-instruction-multiple-data (SIMD) calculations, as when using GPUs; see, e.g., [5, 6, 27]. In fact one of the main motivations for developing a strategy of this type is the difficulty encountered when one attempts to obtain good performance from sparse matrix computations on SIMD-type parallel machines such as those equipped with GPUs [1, 19, 25]. 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 these processors. In addition, the domain decomposition (DD) framework lends itself easily to macro-level parallelism. 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 MLR preconditioners 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.

This paper considers only symmetric matrices. Extensions to the nonsymmetric case are possible and will be explored in our future work. It should also be emphasized that even though the MLR method is developed with parallel platforms in mind, all the testing and implementations discussed in paper have been carried out on scalar machines.

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 two-dimensional (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 (2.1) the matrix A has been split as

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

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 $\alpha \in \{1, 2, \dots, n_y - 1\}$. 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 and E defined above. A formalism similar to the one described above was exploited in [26] 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 $X \in \mathbb{R}^{n_x \times n_x}$ 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 $\tilde{X}$, and products with E and E^T. However, in a multilevel scheme of this sort, it can be verified that applying the preconditioner which requires two solves with B will require 4^{nlev-1} solves at the last level and the operation count for these solves will increase as $\mathcal{O}(2^{nlev-1} \cdot n)$, where $nlev$ denotes the number of levels. So the computational 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., [8]. Sparse approximate inverse methods have been used in a context that is somewhat related in [7]. 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 instance, LU factorizations or inverse matrices approximated by $\mathcal{H}$-matrices rely on low-rank approximations [17, 18]. A similar method has been introduced for solving the Helmholtz problem [12]. The main idea of this method is based on the observation that intermediate Schur complement matrices in an LDLT factorization of a specific order have numerically low-rank off-diagonal blocks, which makes them highly compressible in the $\mathcal{H}$-matrix framework. Similar ideas were exploited in the context of so-called hierarchically semiseparable (HSS) matrices [29, 28]. There are similarities between our work and the HSS work but the main difference is that we do not resort to an LU-type factorization.

Our starting point is the relation (2.7). Assume that we have the best 2-norm rank-k approximation to $B^{-1}E$ as obtained from the SVD 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 $V_k^T V_k = I$. Then, equality (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 systems with it can be solved 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 explode 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 matrix H_k can be computed in a simpler way than by the expression above. Specifically, it will be shown in section 3 that

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

and H_k is symmetric. The alternative expression (2.12) avoids the use of V_k explicitly. Hence, an application of the preconditioner requires one solve with B and a low-rank correction carried out by U_k and H_k. For now, we assume that a system with B can be solved in some unspecified manner.

Approximating both $B^{-1}E$ and its transpose when deriving (2.11) from (2.7), may appear to be a poor choice as it may sacrifice accuracy. Instead, a middle ground approach which approximates $B^{-1}E$ on one side only of the expression, will lead to the following,

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

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 preconditioners based on the two-sided low-rank approximations defined by (2.12).

2.3. Recursive multilevel low-rank approximation.

This section describes a framework for constructing the structure of 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 ith node. Recall that B_i is a 2×2 block diagonal matrix. If we label its two diagonal blocks as B_{i_1} and B_{i_2}, respectively, then we can write

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

To build another level from A_i, by letting $A_{i_1} = B_{i_1}$ and $A_{i_2} = B_{i_2}$, we can repeat the same process on A_{i_1} and A_{i_2}. Otherwise, when a certain number of levels is reached, then A_i is factored by an incomplete Cholesky factorization (IC) as $A_i \approx L_i L_i^T$ if it is symmetric positive definite (SPD), or an incomplete LDLT factorization (ILDLT) as $A_i \approx L_i D_i L_i^T$ if it is symmetric indefinite. The process of building a 4-level structure can be represented by a binary tree displayed in Figure 2.1, where nodes i_1 and i_2 are viewed as the children of node i.

Next, we derive a recursive definition of an MLR preconditioner. For node i which is a nonleaf node, $(U_i)_k (V_i)_k^T$ is the best 2-norm 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. For each node j, we denote by M_j the MLR preconditioning matrix of A_j. Suppose M_{i_1} and M_{i_2} have been obtained from the lower level. If approximate inverses in the form

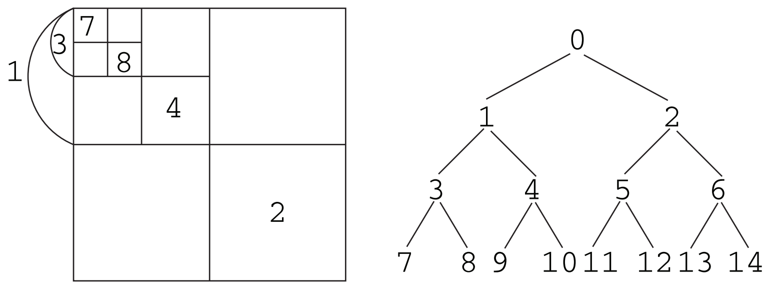

FIG. 2.1. *Matrix partitioning corresponding to a recursive domain bisection (left) and the binary tree structure in an MLR preconditioner (right).*

(2.12) are used, then we can derive M_i^{-1} as

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

On the other hand, if node i is a leaf node, then M_i is given by $M_i = L_i L_i^T$ or $M_i = L_i D_i L_i^T$. Putting these together gives a recursive definition of M_i,

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

Accordingly, the preconditioning operation is a recursive multilevel solve with low-rank approximation, which can be performed by Function. `MLRSolve` shown below. Specifically, the preconditioning operation of M_i, $x = M_i^{-1} b$ can be computed as $x =$. `MLRSolve(`i`,`b`)`. In lines 5 and 6 of Function. `MLRSolve`, vectors $b^{[i_1]}$ and $b^{[i_2]}$ are defined by $b = \left(b^{[i_1]}, b^{[i_2]}\right)^T$, corresponding to the row partitioning of node i into nodes i_1 and i_2.

Then, we analyze the computational cost of performing Function. `MLRSolve(`i`,`b`)`, which mainly lies in the solves at the last level (line 2 of Function. `MLRSolve`) and the rank-k corrections in all the levels except the last one (line 8 of Function. `MLRSolve`). In particular,. `MLRSolve(`0`,`b`)` performs the application of an MLR preconditioner of $A \in \mathbb{R}^{n\times n}$. Suppose the preconditioner has $nlev$ levels. Then, there are 2^{nlev-1} matrices at the last level and the average size of each is $\frac{n}{2^{nlev-1}}$. Applying the preconditioner requires one solve of each matrix at the last level. Therefore, the total operation count of these solves will be $\mathcal{O}(2^{nlev-1} \cdot \frac{cn}{2^{nlev-1}}) = \mathcal{O}(cn)$, where c is the average number of nonzeros per row in the factors. On the other hand, the total operation count of the rank-k corrections will be $\mathcal{O}(2(nlev-1)kn + (2^{nlev-1}-1)k^2)$, where the first and second terms are associated with the U_i's and H_i's, respectively, and typically the second term is negligible. Putting these together, the operation count of applying the MLR preconditioner is $\mathcal{O}((2k(nlev-1)+c)n)$.

FUNCTION. MLRSolve(i, b).

```
Data: $U_j$ and $H_j$ for the low-rank approximation for each nonleaf descendant
      $j$ of $i$; Factors $L_j$, or $L_j$, $D_j$ 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 $x = L_i^{-T}L_i^{-1}b$ or $x = L_i^{-T}D_i^{-1}L_i^{-1}b$
3  else
4      $i_1 \leftarrow i$'s left child;    $i_2 \leftarrow i$'s right child
5      $y_1 =$ MLRSolve($i_1$, $b^{[i_1]}$)
6      $y_2 =$ MLRSolve($i_2$, $b^{[i_2]}$)
7      $y = (y_1, y_2)^T$
8      $x = y + U_i H_i U_i^T b$
```

2.4. Computation of low-rank approximations. In an MLR preconditioner, for each nonleaf node i, the best 2-norm rank-k approximation to matrix $B_i^{-1}E_i$, as obtained from the SVD, is required, which is in the form $B_i^{-1}E_i \approx U_i V_i^T$ with $V_i^T V_i = I$. In this section, we describe an approach to compute this low-rank approximation. Let

$$Q_i = B_i^{-1}E_i. \tag{2.18}$$

Then for the 2-D model problem (2.1), we have $Q_i \in \mathbb{R}^{n_i \times n_x}$. Therefore, computing Q_i requires a solve with B_i and n_x columns of E_i, which will be inefficient if only a few large singular values and the associated singular vectors are wanted, as is the case when the rank $k \ll n_x$. However, the Lanczos bidiagonalization method [14] (see also [20, 4]), can be invoked since it can approximate large singular values and the associated singular vectors without the requirement of forming the matrix explicitly.

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. In our approach, we simply use the full reorthogonalization scheme, in which the orthogonality of a new computed Lanczos vector against all previous ones is reinstated at each step. The cost of this step will not be an issue to the overall performance since we only perform a small number of steps to approximate a few singular values. More efficient reorthogonalization schemes have been proposed; for details see, e.g., [15, 21, 24], but these will not be considered here.

To monitor convergence of the computed singular values we adopt the approach used in [13]. Let $\theta_j^{(m-1)}$ and $\theta_j^{(m)}$ be the singular values of the bidiagonal matrices in two consecutive steps, $m-1$ and m, of the Lanczos procedure. Assume that we want to approximate k largest singular values of A and $k < m$. Then with a preselected tolerance ϵ, the desired singular values are considered to have converged if

$$\left|\frac{\sigma_m - \sigma_{m-1}}{\sigma_{m-1}}\right| < \epsilon, \text{ where } \sigma_{m-1} = \sum_{j=1}^{k}\theta_j^{(m-1)} \text{ and } \sigma_m = \sum_{j=1}^{k}\theta_j^{(m)}. \tag{2.19}$$

The Lanczos bidiagonalization method requires the matrix only in the form of matrix-vector products and matrix-transpose-vector products. This is quite appealing to our case, since the matrix Q_i is implicitly available and recursively defined. Recall that $B_i = \begin{pmatrix} A_{i_1} & \\ & A_{i_2} \end{pmatrix}$ and $Q_i \equiv B_i^{-1}E_i$. Then it follows that applying the Lanczos algorithm to Q_i will require solving linear systems with A_{i_1} and A_{i_2}. However, it appears expensive to perform these solves, even inexactly, for instance, via incomplete factorizations. Obviously, this cannot be a practical approach since the solves will be required for all A_i's except A_0. Alternatively, we perform the Lanczos algorithm on another matrix $\tilde{Q}_i$ with an underlying assumption that its large singular values and the associated singular vectors are close to those of Q_i. Let

$$\tilde{B}_i = \begin{pmatrix} M_{i_1} & \\ & M_{i_2} \end{pmatrix} \tag{2.20}$$

be an approximation to B_i, where M_{i_1} and M_{i_2} are the MLR preconditioning matrices of A_{i_1} and A_{i_2} defined in (2.17). Then we define $\tilde{Q}_i$ to be,

$$\tilde{Q}_i \equiv \tilde{B}_i^{-1} E_i. \tag{2.21}$$

Hence when performing the Lanczos algorithm with $\tilde{Q}_i$, the matrix-vector product and the matrix-transpose-vector product can be carried out by

$$y = \tilde{Q}_i x = \left(\tilde{B}_i^{-1} E_i\right) x = \tilde{B}_i^{-1} w, \text{ where } w = E_i x \tag{2.22}$$

and

$$y = \tilde{Q}_i^T x = \left(\tilde{B}_i^{-1} E_i\right)^T x = E_i^T z, \text{ where } z = \tilde{B}_i^{-1} x. \tag{2.23}$$

With $w = (w_1, w_2)^T$ associated with the row partitioning of node i, $y = \tilde{B}_i^{-1} w$ in (2.22) is given by

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

where e and f can be computed by the recursive solve on nodes i_1 and i_2, namely, $e =.$ `MLRSolve(`i_1, w_1`)` and $f =.$ `MLRSolve(`i_2, w_2`)`. Moreover, $z = \tilde{B}_i^{-1} x$ in (2.23) can be computed likewise.

Note that in this approach, a recursive solve with $\tilde{B}_i$ requires low-rank approximations on the nonleaf nodes and factorizations on the leaf nodes of the subtree rooted at i. This requires building the binary tree structure of an MLR preconditioner in a *postorder traversal* (for details of postorder traversal of a binary tree, see, e.g., [9]).

Next, we analyze the storage requirement for an MLR preconditioner. Suppose the rank used in the MLR preconditioner is k and the number of levels is *nlev*. Then for a nonleaf node i, matrices U_i and H_i are stored for the low-rank approximation. Therefore, the memory requirement for storing these matrices is $\mathcal{O}((nlev-1)kn + (2^{nlev-1}-1)\frac{k^2}{2})$, where the first and second terms are associated with the U_i's and H_i's, respectively. Typically, the second term is negligible relative to the total memory requirement. However, storing the U_i's would require a great deal of memory, especially for large problems, where a large number of levels is needed to make the matrices at the last level small. On the other hand, for a leaf node j, IC or ILDLT factors are stored at a memory cost that depends on the number of the nonzeros in the matrix A_j and the fill-ins in the factors. In general, with a given dropping tolerance in the factorizations, this memory cost increases with the sizes of the matrices at the last level. The matrices stored at all levels of an MLR preconditioner are illustrated schematically in Figure 2.2.

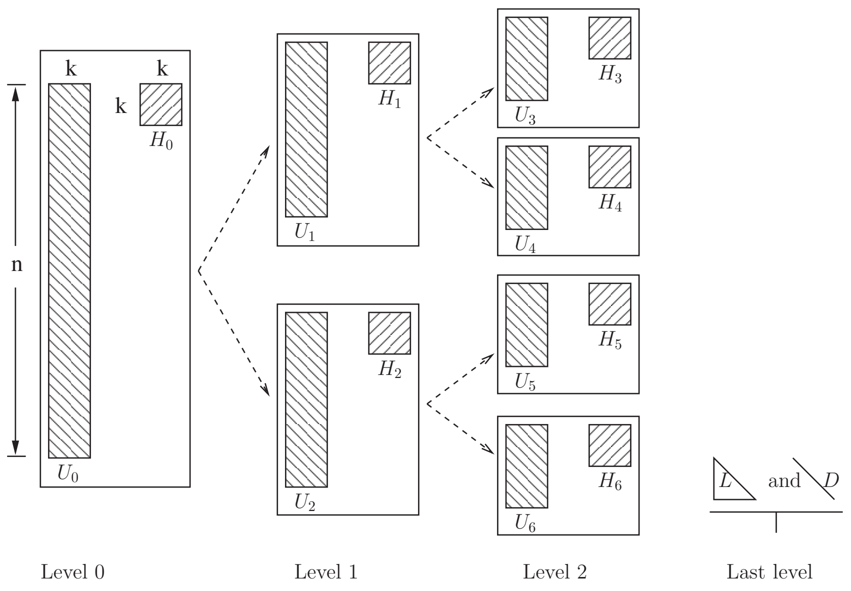

FIG. 2.2. *The matrices stored at all levels of an MLR preconditioner.*

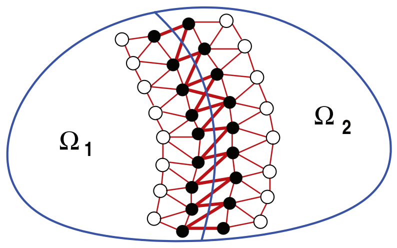

FIG. 2.3. *Illustration of a domain partitioned into two subdomains (DD with an edge separator).*

2.5. Generalization via domain decomposition. To generalize the above scheme to unstructured matrices, we take a DD viewpoint. We consider the situation illustrated in Figure 2.3, in which a domain Ω is partitioned into two subdomains Ω_1 and Ω_2. The interface variables (the filled circles) and the *edge separator* (edges connected to these filled circles in different subdomains) are shown in the figure. We refer to all those variables excluding interface variables as interior variables. Two layers of the interior variables (the open circles) in the two subdomains are shown as well in Figure 2.3.

Then the variables in Ω are naturally partitioned into 4 parts: the interior variables in Ω_1, the interface variables in Ω_1, the interior variables in Ω_2, and the interface variables in Ω_2. We denote by m_i the number of interface variables in Ω_i, for $i = 1, 2$. Therefore, if we assume that the interface variables in each subdomain are listed last, then the $n \times n$ matrix A associated with the domain in Figure 2.3 can be written as,

$$\left(\begin{array}{cc|cc} \hat{B}_1 & \hat{F}_1 & & \\ \hat{F}_1^T & C_1 & & -W \\ \hline & & \hat{B}_2 & \hat{F}_2 \\ & -W^T & \hat{F}_2^T & C_2 \end{array}\right). \tag{2.24}$$

In matrix (2.24), for $i = 1, 2$, the matrices $\hat{B}_i$ and C_i are associated with the interior and interface variables in Ω_i, respectively, and $\hat{F}_i$ corresponds to the connections between the interior and interface variables in Ω_i. In addition, the matrix $W \in \mathbb{R}^{m_1 \times m_2}$ represents the coupling between the interface variables in Ω_1 and those in Ω_2, which is essentially the edge separator composed of the edges between the two subdomains.

Consider the following matrix $E \in \mathbb{R}^{n \times m_1}$ defined with respect to the above partitioning of variables,

$$E = \begin{pmatrix} 0 \\ I \\ 0 \\ W^T \end{pmatrix}. \tag{2.25}$$

Then, we obtain the following generalization of (2.6):

$$A = \begin{pmatrix} B_1 & \\ & B_2 \end{pmatrix} - EE^T \text{ with } B_i = \begin{pmatrix} \hat{B}_i & \hat{F}_i \\ \hat{F}_i^T & C_i + D_i \end{pmatrix} \text{ and } \begin{cases} D_1 = I, \\ D_2 = W^T W. \end{cases} \tag{2.26}$$

For a model problem, e.g., the one discussed at the beginning of section 2, with a careful ordering of the interface variables of Ω_1 and Ω_2, we have $W = I$, which yields $D_1 = D_2 = I$. Here, however, there could be a big imbalance between the expressions of B_1 and B_2. One way to mitigate the imbalance is by weighting the two nonzero terms in (2.25). The matrix E can then be redefined as

$$E_\alpha = \begin{pmatrix} 0 \\ \alpha I \\ 0 \\ \frac{W^T}{\alpha} \end{pmatrix}, \tag{2.27}$$

which yields $D_1 = \alpha^2 I$ and $D_2 = \frac{1}{\alpha^2} W^T W$. In order to make the spectral radius of D_1 equal to that of D_2, we take α as the square root of the largest singular value of W.

The above method can be generalized in a number of ways by considering any factorization of W, which can be a rectangular matrix. Consider any factorization $W = X_1 X_2$. Looking only at the relevant blocks of the matrix (2.24), we can write

$$\begin{pmatrix} 0 & W \\ W^T & 0 \end{pmatrix} = \begin{pmatrix} X_1 \\ X_2^T \end{pmatrix} \begin{pmatrix} X_1^T & X_2 \end{pmatrix} - \begin{pmatrix} X_1 X_1^T & 0 \\ 0 & X_2^T X_2 \end{pmatrix}.$$

This means that for any factorization $W = X_1 X_2$ we can replace E in (2.25) and D_1, D_2 in (2.26) by:

$$E = \begin{pmatrix} 0 \\ X_1 \\ 0 \\ X_2^T \end{pmatrix}, \quad D_1 = X_1 X_1^T, \quad D_2 = X_2^T X_2. \tag{2.28}$$

With this, the next simplest generalization of (2.25), is to take X_1 to be a diagonal matrix, instead of αI. In this case, $X_2 = X_1^{-1} W$, and X_1 is some nonsingular diagonal matrix to be selected. We need to balance between X_1^2 and $X_2^T X_2 = W^T X_1^{-2} W$. Note that these two matrices can be of (slightly) different dimensions. Multiplying the matrix W to the left by X_1^{-1} corresponds to scaling the rows of W. We will *scale each row by the square root of its norm*, so

$$X_1 = \text{Diag}\,\{\sqrt{\omega_i}\}, \quad \omega_i = \|e_i^T W\|_2.$$

In this way,

$$X_1^2 = \text{Diag}\,\{\omega_i\} \quad \text{and} \quad X_2^T X_2 = W^T \text{Diag}\,\{\omega_i^{-1}\} W.$$

Although these two matrices can be of different dimensions, note that the norm of the ith row of X_1 (which is $\sqrt{\omega_i}$) is the same as the norm of the ith row of X_2. Of course, $X_1 X_1^T$ is diagonal, and $X_2^T X_2$ is not. But in terms of their general magnitude these terms will be well balanced in most cases.

Finally, we also considered another balancing option based on an LU factorization of W, which may not have full rank, $PWQ = LU$, where $L \in \mathbb{R}^{m_1 \times l}$, $U \in \mathbb{R}^{l \times m_2}$, and $l = \min(m_1, m_2)$, and P and Q are permutation matrices chosen to reduce fill-in and to perform pivoting. We then balance the LU factors by a diagonal matrix $\Lambda \in \mathbb{R}^{l \times l}$ as $W = (P^T L \Lambda)(\Lambda^{-1} U Q^T) \equiv \tilde{L}\tilde{U}$. Here Λ is selected so that $\|\tilde{l}_i\|_2 = \|\tilde{u}_i^T\|_2$ for $i = 1, \ldots l$, where $\tilde{l}_i$ is the ith column of $\tilde{L}$ and $\tilde{u}_i^T$ is the ith row of $\tilde{U}$. Then formulas (2.28) are used with $X_1 = \tilde{L}, X_2 = \tilde{U}$. This strategy was explored but did not yield results as good as the ones discussed above.

2.6. Updating an MLR preconditioner.

MLR preconditioners have the appealing quality that they can be easily updated. By this we mean that if the performance of the iterative procedure is not satisfactory, one can compute and use an improved version of the preconditioner without foregoing work previously performed. This is a quality shared by all approximate inverse preconditioners but the multilevel nature of MLR preconditioners requires particular considerations.

With a rank-k MLR preconditioner already computed, an incremental rank-$(k+1)$ MLR preconditioner can be obtained by resorting to a form of deflation. For each nonleaf node i, we perform the Lanczos algorithm on the "deflated" matrix $K_i \equiv \tilde{Q}_i - U_i V_i^T$ to compute the best 2-norm rank-1 approximation of K_i. Although V_i is not necessarily available, by assuming that $V_i = \tilde{Q}_i^T U_i$, we have $K_i = \tilde{Q}_i - U_i U_i^T \tilde{Q}_i = (I - U_i U_i^T)\tilde{Q}_i$, where U_i is available from the rank-k MLR preconditioner. In the Lanczos algorithm, the matrix-vector operation, $y = K_i x$, can be performed as $y = (I - U_i U_i^T)(\tilde{Q}_i x)$, where the term $\tilde{Q}_i x$ is computed as in (2.22). Moreover, $y = K_i^T x$ can be computed likewise. Suppose the rank-1 approximation to K_i is in the form $K_i \approx uv^T$. Then we can update U_i and H_i from the rank-k MLR preconditioner into $\bar{U}_i = (U_i, u)$ and $\bar{H}_i = I - (U_i, u)^T E_i (V_i, v)$ which satisfies

$$\bar{H}_i = I - \begin{pmatrix} U_i^T E_i V_i & U_i^T E_i v \\ u^T E_i V_i & u^T E_i v \end{pmatrix} = \begin{pmatrix} H_i & U_i^T E_i v \\ u^T E_i V_i & 1 - u^T E_i v \end{pmatrix}. \tag{2.29}$$

Under a few assumptions, the matrix $\bar{H}_i$ can be shown to be also symmetric. Specifically, assume that $V_i^T v = 0$ and $K_i v = u$, and that $\tilde{B}_i^{-1} E_i V_i = U_i$. Then:

$$\begin{aligned} u^T E_i V_i &= v^T K_i^T E_i V_i \\ &= v^T (\tilde{Q}_i - U_i V_i^T)^T E_i V_i \\ &= v^T \tilde{Q}_i^T E_i V_i \\ &= v^T \tilde{Q}_i^T \tilde{B}_i U_i \\ &= v^T (E_i^T \tilde{B}_i^{-1}) \tilde{B}_i U_i \\ &= v^T E_i^T U_i. \end{aligned} \tag{2.30}$$

Symmetry is important because the (2,1) term $u^T E_i V_i$ in (2.29) is not available in practice since V_i is not necessarily available. This term can be obtained by symmetry, i.e., as $v^T E_i^T U_i$. Note that the assumptions made above can be enforced, but they are satisfied approximately in practice and this should be enough to exploit the symmetry relation above since we are only building an approximate inverse for preconditioning. Finally, as in the process of building an MLR preconditioner, the incremental update is also performed in a postorder traversal of the MLR tree structure.

3. Analysis.

This section will examine a few questions which have not been addressed so far, regarding the preconditioner defined in previous sections. First, we will explore a number of algebraic relationships already mentioned, which lead to a few simplifications of the algorithm. Second, we will attempt to explain on a model problem, why a small rank approximation is sufficient to capture the difference $A^{-1} - B^{-1}$ in (2.7).

3.1. Algebraic relationships and equivalences.

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 refer to this as assumption (H).

(H) $U_k V_k^T$ in (2.9) is the best 2-norm rank-k approximation to $B^{-1}E$, and $V_k^T V_k = I$.

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 and that we have $V_k^T V_k = I$. 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

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

This completes the proof. $\square$

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 $\sum_{i\le k}\sigma_i u_i v_i^T$ 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* (H) *is satisfied and that B is SPD. 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 SPD.*

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* (H) *is satisfied 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 $B^{-1}EG_k^{-1}V_k U_k^T = U_k V_k^T G_k^{-1} V_k U_k^T$. The proof requires the expression for the inverse of G_k which is provided by (3.1) of Lemma 3.1, which is valid under the assumption that $I - U_k^T E V_k$ is nonsingular. From this it follows that

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

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* (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 the Sherman–Morrison formula leads to the expression

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

Using the relation $U_k B U_k^T = U_k^T E V_k$ 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* (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

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

The final question we would like to answer now is, Under which condition is the preconditioner SPD? 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 $I - U_k^T E V_k$ 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 SPD if and only if $\rho(U_k^T E V_k) < 1$.*

Proof. Recall from Proposition (3.2) that the matrix $U_k^T E V_k$ is an SPD matrix. If $\rho(U_k^T E V_k) < 1$ then clearly the eigenvalues of $I - U_k^T E V_k$ are all positive. Therefore, the matrix $I - U_k^T E V_k$ is SPD 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 SPD, 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). $\square$

Finally, the above theorem can be extended to the recursive multilevel preconditioner which is defined by (2.17). The following result shows a sufficient condition for the preconditioning matrix M_i to be SPD. If we denote by $st(i)$ the subtree rooted at i, then in the following, we let $l(i)$ be a set consisting of all the leaf nodes of $st(i)$ and $nl(i)$ be a set consisting of all the nonleaf nodes of $st(i)$. We will refer to (H_i) as the assumption (H) above at node i: $B^{-1}E$ in (H) is replaced by $\tilde{B}_i^{-1}E_i$, where $\tilde{B}_i$ is defined in (2.20), and $U_k V_k^T$ is now $U_i V_i^T$ and V_k becomes V_i.

COROLLARY 3.7. *Assume that* (H_i) *is satisfied and that for all $j \in l(i)$ the matrices M_j are SPD. Then the preconditioning matrix M_i given by* (2.17) *is well defined and SPD if $\rho(U_j^T E_j V_j) < 1$ for all $j \in nl(i)$.*

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

3.2. Two-level analysis for a 2-D model problem.

The analysis of the multilevel preconditioner proposed in this paper is difficult for general problems. In the simplest case when the matrix A originates from a 2-D Laplacian discretized by centered differences, we can easily perform a spectral analysis of the preconditioner to provide some insight. Specifically, one of the questions which we wish to address is why a low-rank approximation, sometimes very low, yields a reasonable preconditioner. Looking at (2.7) this can be understood only by a rapid decay of the eigenvalues of the difference between A^{-1} and B^{-1}. This leads us to a spectral analysis of the matrix $B^{-1}EX^{-1}E^TB^{-1}$ in (2.7). Note that the method essentially approximates this matrix with a low-rank approximation, which is itself obtained from approximating $B^{-1}E$. As will be seen, $B^{-1}E$ itself may not show as good a decay in its singular values as does $B^{-1}EX^{-1}E^TB^{-1}$.

Assume that $-\Delta$ is discretized on a grid of size $(n_x+2)\times(n_y+2)$, with Dirichlet boundary conditions and that the ordering is major along the x direction. Call T_x the tridiagonal matrix of dimension $n_x \times n_x$ which discretizes $\partial^2/\partial x^2$, and similarly T_y the tridiagonal matrix of dimension $n_y \times n_y$ which discretizes $\partial^2/\partial y^2$. The scaling term $1/h^2$ is omitted so these matrices have the constant 2 on their main diagonal and -1 on their codiagonals. Finally, we will call I_x, I_y the identity matrices of size n_x, n_y, respectively. Then, the matrix A which results from discretizing $-\Delta$ can be written using Kronecker products as follows:

$$A = T_y \otimes I_x + I_y \otimes T_x. \tag{3.9}$$

In this section, we will make extensive use of Kronecker products. We need to recall a few basic rules when working with such products. First, for any $A \in \mathbb{R}^{m\times n}$, $B \in \mathbb{R}^{p\times q}$, $C \in \mathbb{R}^{n\times s}$, $D \in \mathbb{R}^{q\times t}$ we have

$$(A \otimes B)\ (C \otimes D) = (AC) \otimes (BD)\ . \tag{3.10}$$

This is valid in particular when $s = t = 1$, i.e., when B and D are vectors, which we denote by $u \in \mathbb{R}^n, v \in \mathbb{R}^q$:

$$(A \otimes B)(u \otimes v) = (Au) \otimes (Bv)\ . \tag{3.11}$$

Another simple rule involves the transposition operation:

$$(A \otimes B)^T = A^T \otimes B^T. \tag{3.12}$$

Eigenvalues and eigenvectors of A in (3.9) can be easily obtained from the expression (3.9) by using the relation (3.11). The eigenvectors of A are of the form $u_i \otimes v_j$, where u_i is an eigenvector of T_y associated with an eigenvalue μ_i and v_j is an eigenvector of T_x associated with an eigenvalue λ_i. The eigenvalues of A are $\mu_i + \lambda_j$.

Consider now the case when B is given by (2.6). In terms of the grid, the domain is separated horizontally, keeping $n_y/2$ horizontal lines in one half of the domain and the remaining lines in the other half. The matrix E itself can be written as a Kronecker product. If e is the vector of $\mathbb{R}^{n_y}$ which has zero entries except in locations $n_y/2$ and $n_y/2+1$, where the entries are 1, then clearly $E = e \otimes I_x$ and from (3.12) and (3.10)

$$EE^T = (e \otimes I_x)(e \otimes I_x)^T = (e \otimes I_x)(e^T \otimes I_x) = (ee^T) \otimes I_x,$$

so

$$B = A + EE^T = T_y \otimes I_x + I_y \otimes T_x + (ee^T) \otimes I_x = (T_y + ee^T) \otimes I_x + I_y \otimes T_x. \tag{3.13}$$

The eigenvalues and eigenvectors of B can be obtained in a similar way to those of A. The only difference is that T_y is now perturbed by the rank-1 matrix ee^T into

$\tilde{T}_y = T_y + ee^T$. In fact this perturbation corresponds to splitting the one-dimensional domain (y direction) in two and applying the Neumann boundary condition at the interface. As it turns out the eigenvalues of this matrix are simply related to those of T_y. Details can be found in the appendix.

We will denote by $\tilde{\mu}_i$ and $\tilde{u}_i$ the corresponding eigenvalues and vectors, noting in passing that, due to the structure of $\tilde{T}_y$, all eigenvalues are double. It is easy to invert B by using its spectral decomposition:

$$B^{-1} = \sum_{i=1}^{n_y}\sum_{j=1}^{n_x} \frac{1}{\tilde{\mu}_i + \lambda_j}(\tilde{u}_i \otimes v_j)(\tilde{u}_i \otimes v_j)^T = \sum_{i,j} \frac{1}{\tilde{\mu}_i + \lambda_j}(\tilde{u}_i\tilde{u}_i^T) \otimes (v_j v_j^T), \tag{3.14}$$

which is a *linear combination of Kronecker products of eigenprojectors*. Thus,

$$B^{-1}E = \sum_{i,j} \frac{1}{\tilde{\mu}_i + \lambda_j}\left[(\tilde{u}_i\tilde{u}_i^T) \otimes (v_j v_j^T)\right](e \otimes I_x) = \sum_{i,j} \frac{\tilde{u}_i^T e}{\tilde{\mu}_i + \lambda_j}\tilde{u}_i \otimes (v_j v_j^T)\,. \tag{3.15}$$

We are interested in the difference $A^{-1} - B^{-1}$ which is the very last term in (2.7), i.e., the matrix $B^{-1}EX^{-1}E^TB^{-1}$, where $X = I - E^TB^{-1}E$. The nonzero eigenvalues of this matrix are the same as those of $X^{-1}E^TB^{-2}E$. First, consider the matrix $B^{-2}E$ which can be expanded in a way similar to that of $B^{-1}E$ above:

$$B^{-2}E = \sum_{i,j} \frac{\tilde{u}_i^T e}{(\tilde{\mu}_i + \lambda_j)^2}\tilde{u}_i \otimes (v_j v_j^T). \tag{3.16}$$

Next, for the matrix X^{-1}, we have from (3.15),

$$E^TB^{-1}E = \sum_{i,j} \frac{\tilde{u}_i^T e}{\tilde{\mu}_i + \lambda_j}(e^T \otimes I_x)(\tilde{u}_i \otimes (v_j v_j^T))$$

$$= \sum_{i,j} \frac{(\tilde{u}_i^T e)^2}{\tilde{\mu}_i + \lambda_j}(v_j v_j^T) \equiv \sum_{j=1}^{n_x} \alpha_j v_j v_j^T, \tag{3.17}$$

where we have denoted by α_j the sum

$$\alpha_j = \sum_{i=1}^{n_y} \frac{(\tilde{u}_i^T e)^2}{\tilde{\mu}_i + \lambda_j}\ . \tag{3.18}$$

Equation (3.17) expresses $E^TB^{-1}E$ as a linear combination of the eigenprojectors $v_j v_j^T$ of T_x. The matrix X^{-1} can be readily obtained from this since $X = I - \sum \alpha_j v_j v_j^T = \sum (1 - \alpha_j) v_j v_j^T$:

$$X^{-1} = \sum_{j=1}^{n_x} \frac{1}{1 - \alpha_j} v_j v_j^T. \tag{3.19}$$

Finally, a spectral expansion for $E^TB^{-2}E$ can be obtained in a very similar way to (3.17):

$$E^TB^{-2}E = \sum_{j=1}^{n_x} \beta_j v_j v_j^T, \quad \beta_j = \sum_{i=1}^{n_y} \frac{(\tilde{u}_i^T e)^2}{(\tilde{\mu}_i + \lambda_j)^2}. \tag{3.20}$$

In the end, the eigenvalues of $X^{-1}E^TB^{-2}E$ are given by

$$\gamma_j = \frac{\beta_j}{1 - \alpha_j}, \quad j = 1, \ldots, n_x\ . \tag{3.21}$$

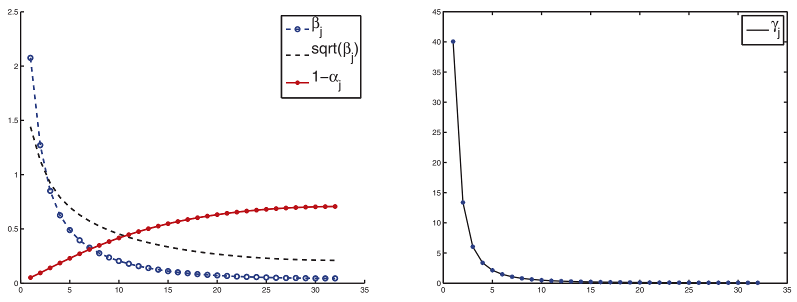

FIG. 3.1. *Illustration of the decay of the eigenvalues of $B^{-1}EX^{-1}E^TB^{-1}$ for the case when $nx = ny = 32$. The left panel shows the coefficients $\beta_j, 1-\alpha_j$ and the square roots of β_j's, which are the singular values of $B^{-1}E$. The right panel shows the ratios γ_j, the eigenvalues of $B^{-1}EX^{-1}E^TB^{-1}$. In this particular case 3 eigenvectors will capture 92% of the inverse whereas 5 eigenvectors will capture 97% of the inverse.*

We can now have an explicit expression of these eigenvalues since the quantities above are all known. From the appendix, we see that $e^T\tilde{u}_i = 0$ for i odd, and so we set $i = 2k$, for $k = 1, \ldots, n_y/2$ in (3.20) and (3.18), to obtain

$$\beta_j = \sum_{k=1}^{n_y/2} \frac{\sin^2 \frac{n_y k\pi}{n_y+1}}{4\left(\sin^2 \frac{k\pi}{n_y+1} + \sin^2 \frac{j\pi}{2(n_x+1)}\right)^2}, \quad \alpha_j = \sum_{k=1}^{n_y/2} \frac{\sin^2 \frac{n_y k\pi}{n_y+1}}{\sin^2 \frac{k\pi}{n_y+1} + \sin^2 \frac{j\pi}{2(n_x+1)}} .$$

An illustration of these two functions along with the eigenvalues γ_j is shown in Figure 3.1. The scalars β_j decay from the largest value β_1. Note that the singular values of $B^{-1}E$ are the square roots of the β_j's and these do not decay particularly fast. The scalars β_j's, their squares, decay somewhat rapidly. However, the more interesting observation is the very rapid decay of the ratios (3.21). While the decay of the singular values of $B^{-1}E$ is not sufficient by itself to explain a good approximation of $A^{-1}-B^{-1}$ by a low-rank matrix, *what makes a difference is the squaring of $B^{-1}E$ and the strong damping effect from the term X^{-1}, which contributes the divisions by $1-\alpha_j$.*

Because $B^{-1}E$ does not necessarily have fast-decaying singular values, it may be argued that the approach based on extracting a low-rank approximation from it may have problems. However, the above expansion shows why this is not an issue. For example, assume that we use, say 15 singular vectors when approximating $B^{-1}E$, while only 3 are needed to approximate $A^{-1} - B^{-1}$ well. Then in the related approximation of $B^{-1}EX^{-1}E^TB^{-1}$, the resulting rank-1 terms beyond the third rank will be damped by the factors γ_j which are relatively small, and so they will contribute very little to the resulting preconditioner.

4. Numerical experiments.

The experiments were conducted on a machine equipped with an Intel Xeon X5675 processor (12 MB Cache, 3.06 GHz, 6-core) and 96 GiB of main memory. A preliminary implementation of MLR preconditioners has been written in C/C++ and compiled by g++ using -O2 optimization level. BLAS and LAPACK routines from Intel Math Kernel Library (MKL, version 10.2) are used to enhance the performance on multiple cores. We should emphasize that while our ultimate goal is to implement this class of methods on machines equipped with GPUs, the present experiments were not performed on such platforms. Implementations and tests of MLR preconditioners on GPUs will require time and are left as future work. Based on experimental results on current platforms with our current code, we can state that, in general, building an MLR preconditioner requires more CPU time than an IC or an ILDLT preconditioner requiring similar storage. Nevertheless, MLR preconditioners achieved great CPU time savings in the iterative process. In addition, we should add that this is a first implementation and that improvements are possible for the preprocessing stage. Finally, note that the preconditioner construction itself offers far more parallelism than standard ILU-type preconditioners, a feature which will again be tested in our future work.

In this section, we first report on results of solving symmetric linear systems from a 2-D and a three-dimensional (3-D) elliptic partial differential equation (PDE) on regular grids using MLR preconditioners combined with Krylov subspace methods. For these problems, a recursive geometric bisection is used for the domain decomposition in an MLR preconditioner. Specifically, a 2-D regular grid is cut in half along the shorter side, while a 3-D regular grid is cut in half along the face of the smallest area. Next, MLR preconditioners were tested for solving a sequence of general symmetric linear systems. For these problems, a graph partitioning algorithm `PartGraphRecursive` from METIS [16] was used for the domain decomposition.

Three types of preconditioners are compared in our experiments: (1) MLR preconditioners, (2) incomplete Cholesky factorization with threshold dropping (ICT), for SPD cases, and (3) ILDLT with threshold dropping, for indefinite cases. The accelerators used are the conjugate gradient (CG) method for SPD cases and the generalized minimal residual (GMRES) method with a restart dimension of 40 for indefinite cases. For all cases, iterations were stopped whenever the residual norm has been reduced by 8 orders of magnitude or the maximum number of iterations allowed, which is 500, was exceeded. The results are summarized in Tables 4.2–4.7 where all CPU times are reported in seconds. When comparing preconditioners, the following factors are considered: (1) 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, (2) time for building preconditioners, (3) the number of iterations, and (4) time for the iterations. In all tables, "`F`" indicates nonconvergence within the maximum allowed number of steps.

In the Lanczos bidiagonalization method, the convergence is checked every 10 iterations and the tolerance ϵ used for convergence in (2.19) is set to 10^{-3}. In addition, we set the maximum number of Lanczos steps as ten times the number of requested singular values. At the last level of an MLR preconditioner, we preorder the matrices by the approximate minimum degree ordering (AMD) [2, 3, 11] to reduce fill-ins. Furthermore, for most cases, we are able to choose a smaller tolerance in the factorization than that used in an ICT or an ILDLT preconditioner and ensure that both preconditioners have similar memory requirements. In addition, when using an ICT or ILDLT factorization as a global preconditioner, we also preorder the matrix by the AMD ordering.

4.1. 2-D/3-D model problems. We will examine a 2-D elliptic PDE,

$$\begin{gathered} -\frac{\partial^2 u}{\partial x^2} - \frac{\partial^2 u}{\partial y^2} - cu = -\left(x^2 + y^2 + c\right) e^{xy} \ \text{ in } \Omega, \\ u = e^{xy} \text{ on } \partial\Omega, \end{gathered} \tag{4.1}$$

and a 3-D elliptic PDE as well,

$$\begin{gathered} -\frac{\partial^2 u}{\partial x^2} - \frac{\partial^2 u}{\partial y^2} - \frac{\partial^2 u}{\partial z^2} - cu = -6 - c\left(x^2 + y^2 + z^2\right) \text{ in } \Omega, \\ u = x^2 + y^2 + z^2 \text{ on } \partial\Omega. \end{gathered} \tag{4.2}$$

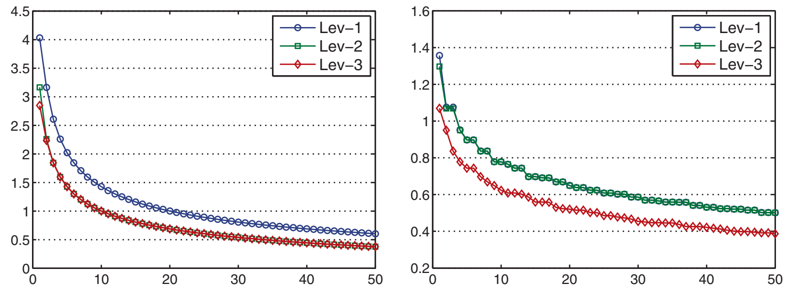

FIG. 4.1. *Illustration of the decay of singular values of Q_i for $i = 0, 1, 3$, for a 2-D model problem (left) and a 3-D model problem (right).*

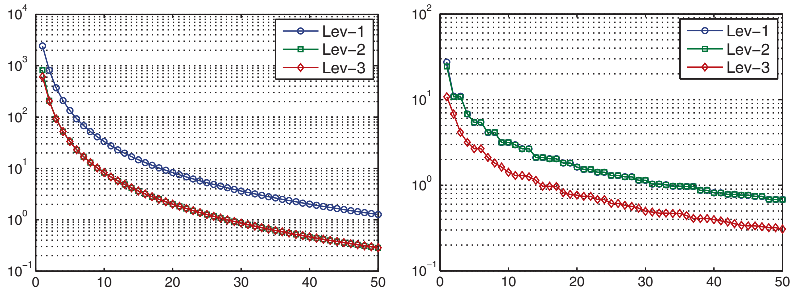

FIG. 4.2. *Illustration of the decay of eigenvalues of $B_i^{-1}E_iX^{-1}E_i^TB_i^{-1}$ (log scale) for $i = 0, 1, 3$, for a 2-D model problem (left) and a 3-D model problem (right).*

The domain is $\Omega = (0, 1) \times (0, 1)$ for the 2-D problem and for the 3-D problem, is $\Omega = (0, 1) \times (0, 1) \times (0, 1)$, while $\partial\Omega$ is the boundary. The exact solutions of (4.1) and (4.2) are $u = e^{xy}$ and $u = x^2 + y^2 + z^2$, respectively. We take the 5-point (or 7-point) centered difference approximation on regular 2-D (or 3-D) grids.

To begin with, we examine the singular values of Q_i and $\tilde{Q}_i$ and the eigenvalues of $B_i^{-1}E_iX_i^{-1}E_i^TB_i^{-1}$ for the above 2-D and 3-D model problems on a 256×256 grid and a $32 \times 32 \times 64$ grid, respectively, where Q_i and $\tilde{Q}_i$ were defined in (2.18) and (2.21). For both problems, we set $c = 0$ in (4.1) and (4.2). The first 50 singular values of Q_i and the first 50 eigenvalues of $B_i^{-1}E_iX_i^{-1}E_i^TB_i^{-1}$, for $i = 0, 1, 3$, from the first three levels of the MLR preconditioners are depicted in Figures 4.1 and 4.2 (note log scale on the y-axis of Figure 4.2). As shown, compared to the singular values of Q_i, the eigenvalues of $B_i^{-1}E_iX_i^{-1}E_i^TB_i^{-1}$ decay more rapidly due to the damping effect of X_i^{-1}.

Recall that when building an MLR preconditioner, instead of Q_i, we compute approximate singular values and vectors of $\tilde{Q}_i$ defined in (2.21) by the Lanczos algorithm. In Table 4.1, we compare the singular values of $\tilde{Q}_i$ to those of Q_i at the first three levels of the MLR preconditioners which have 5 levels. For a rank-k MLR preconditioner, only the k largest singular values of $\tilde{Q}_i$ are required. In this table, we tabulate the first eight singular values of Q_i and the k largest singular values computed for $\tilde{Q}_i$ in the MLR preconditioners with rank $k = 2, 4, 8$ for $i = 0, 1, 3$. From the results in Table 4.1, we make several observations: (1) for a certain Q_i and rank k, smaller singular values computed for $\tilde{Q}_i$ approximate those corresponding ones of Q_i better; (2) for a certain Q_i, among different ranks, singular values computed for $\tilde{Q}_i$

Table 4.1

Approximation to singular values of Q_i, for $i = 0, 1, 3$, at the first 3 levels of MLR preconditioners by the Lanczos bidiagonalization method for a 2-D model problem (top) and a 3-D model problem (bottom).

	Rank	σ_1	σ_2	σ_3	σ_4	σ_5	σ_6	σ_7	σ_8
Q_0	-	4.029	3.166	2.609	2.260	2.021	1.845	1.707	1.596
$\tilde{Q}_0$	$k = 2$	2.873	2.605	-	-	-	-	-	-
	$k = 4$	3.381	2.926	2.507	2.260	-	-	-	-
	$k = 8$	3.494	2.996	2.555	2.260	1.987	1.813	1.679	1.596
Q_1	-	3.166	2.260	1.845	1.596	1.427	1.301	1.203	1.123
$\tilde{Q}_1$	$k = 2$	2.847	2.260	-	-	-	-	-	-
	$k = 4$	2.931	2.260	1.813	1.596	-	-	-	-
	$k = 8$	2.987	2.260	1.834	1.596	1.415	1.301	1.191	1.123
Q_3	-	2.849	2.238	1.843	1.596	1.427	1.301	1.203	1.123
$\tilde{Q}_3$	$k = 2$	2.777	2.238	-	-	-	-	-	-
	$k = 4$	2.824	2.238	1.813	1.596	-	-	-	-
	$k = 8$	2.841	2.238	1.833	1.596	1.415	1.300	1.191	1.123

	Rank	σ_1	σ_2	σ_3	σ_4	σ_5	σ_6	σ_7	σ_8
Q_0	-	1.357	1.075	1.075	0.951	0.898	0.898	0.836	0.836
$\tilde{Q}_0$	$k = 2$	1.130	1.012	-	-	-	-	-	-
	$k = 4$	1.161	1.034	1.012	0.951	-	-	-	-
	$k = 8$	1.193	1.053	1.034	0.951	0.839	0.824	0.812	0.799
Q_1	-	1.297	1.069	1.069	0.950	0.897	0.897	0.836	0.836
$\tilde{Q}_1$	$k = 2$	1.114	1.007	-	-	-	-	-	-
	$k = 4$	1.167	1.024	1.003	0.951	-	-	-	-
	$k = 8$	1.184	1.045	1.017	0.950	0.840	0.824	0.811	0.788
Q_3	-	1.069	0.950	0.836	0.778	0.743	0.743	0.697	0.669
$\tilde{Q}_3$	$k = 2$	1.021	0.950	-	-	-	-	-	-
	$k = 4$	1.026	0.950	0.797	0.749	-	-	-	-
	$k = 8$	1.039	0.950	0.805	0.760	0.743	0.673	0.642	0.610

with a higher rank k approximate those corresponding ones of Q_i better; (3) among different Q_i's, singular values computed for $\tilde{Q}_i$ at a lower level approximate those corresponding ones of Q_i better.

Then we report on the performance of MLR preconditioners when solving the 2-D and 3-D model problems in (4.1) and (4.2). First, we set $c = 0$, so that the coefficient matrices are SPD. Recall from Corollary 3.7 in section 3, that if we have $\rho(U_i^T E V_i) < 1$ for all i for an SPD matrix, then the MLR preconditioner defined in (2.17) is also SPD. Therefore, for these cases, we use an MLR preconditioner along with the CG method. Numerical experiments were carried out to compare the performance of MLR preconditioners and ICT preconditioners. We solved each problem on 3 different grids. The size of each grid and the order of the matrix (N) are shown in Table 4.2. The fill ratios (fill), the numbers of iterations (its), the time for building preconditioners (p-t) and iterations (i-t) for both preconditioners are also tabulated. In the experiments, the fill ratios of both preconditioners were controlled to be roughly equal. For the 2-D problems, the number of levels of MLR preconditioners is fixed at 5. On the other hand, for the 3-D problems, we increase the number of levels by one as the size of the grid doubles. In this way, the sizes of the matrices at the last level of the MLR preconditioners on the 3 grids will be kept about the same. Specifically, we set the number of levels to 5 for the first problem and use 7 and 10 levels for the other two problems, respectively. For all the problems, the rank used in MLR preconditioners is 2.

Table 4.2
Comparison between ICT *preconditioners and* MLR *preconditioners for solving SPD linear systems from* 2*-D/*3*-D elliptic equations in* (4.1) *and* (4.2) *along with the* CG *method.*

Grid	N	ICT-CG				MLR-CG			
		fill	p-t	its	i-t	fill	p-t	its	i-t
256×256	65,536	3.1	0.08	69	0.19	3.2	0.45	84	0.12
512×512	262,144	3.2	0.32	133	1.61	3.5	1.57	132	1.06
1024×1024	1,048,576	3.4	1.40	238	15.11	3.5	4.66	215	9.77
$32 \times 32 \times 64$	65,536	2.9	0.14	33	0.10	3.0	0.46	43	0.08
$64 \times 64 \times 64$	262,144	3.0	0.66	47	0.71	3.1	3.03	69	0.63
$128 \times 128 \times 128$	2,097,152	3.0	6.59	89	13.47	3.2	24.61	108	10.27

Table 4.3
Performance of solving a 2*-D SPD problem in* (4.1) *by* MLR *preconditioners of different ranks and numbers of levels along with the* CG *method.*

rank	nlev	fill	mem.lrk		mem.fact		p-t	its	i-t
2	5	3.5	45%	(8.4m)	55%	(10.1m)	4.66	215	9.77
2	6	3.9	52%	(10.5m)	48%	(9.7m)	5.40	251	11.06
2	7	4.2	58%	(12.6m)	42%	(9.3m)	5.80	230	9.86
2	8	4.5	63%	(14.7m)	37%	(8.8m)	6.65	254	10.89
2	9	4.8	67%	(16.8m)	33%	(8.4m)	7.26	250	11.20
3	5	4.3	56%	(12.6m)	44%	(10.1m)	6.23	192	10.00
4	5	5.1	62%	(16.8m)	38%	(10.1m)	7.60	185	10.11
5	5	5.9	68%	(21.0m)	32%	(10.1m)	9.81	168	11.06
6	5	6.7	71%	(25.2m)	29%	(10.1m)	12.13	167	11.98

The results in Table 4.2 first indicate that an MLR preconditioner can be used along with the CG method to solve the above problems. Second, compared to an ICT preconditioner, building an MLR preconditioner requires more CPU time, 4 times more on average in this set of experiments. Finally, for the 2-D problems, the MLR-CG method achieved convergence in slightly fewer iterations than with the ICT preconditioners, whereas for the 3-D problems, it required more iterations. For all cases, we obtained performance gain using MLR preconditioners in terms of reduced iteration times. The CPU time for building an MLR preconditioner is typically dominated by the cost of matrix-vector operations in (2.22) and (2.23) when performing the Lanczos algorithm on $\tilde{Q}_i$. Furthermore, this cost is actually governed by the cost of the triangular solves at the last level. Different layers of parallelism exist in building an MLR preconditioner. For instance, in a matrix-vector operation of each Q_i, recursive calls in lines 5 and 6 of Function. `MLRSolve` are independent. Moreover the Lanczos algorithm can be performed on all $\tilde{Q}_i$'s within the same level in parallel. These features have not yet been implemented in our current code.

In the next set of experiments, we examine the behavior of MLR preconditioners with different ranks and numbers of levels. We solved the same 2-D problem on a 1024×1024 regular grid. We present the results in Table 4.3, where "mem.lrk" stands for the memory requirement for storing the matrices for the low-rank approximations and "mem.fact" indicates the memory requirement for storing the factors at the last level. For each case, the percentage and the number of nonzeros (measured in millions) are reported. Here are a few observations from Table 4.3. First, we fix the rank to $k = 2$ but increase the number of levels. As a result the fill ratio grows and the MLR preconditioner requires more CPU time to build. Note that the number of nonzeros in the factors decreases since the orders of the matrices at the last level become smaller. In addition, more iterations are required for convergence and the total iteration time

Table 4.4
Comparison between ILDLT *preconditioners and* MLR *preconditioners for solving symmetric indefinite linear systems from* 2*-D/*3*-D elliptic equations in* (4.1) *and* (4.2) *along with the* GMRES(40) *method.*

Grid	ILDLT-GMRES				MLR-GMRES			
	fill	p-t	its	i-t	fill	p-t	its	i-t
256×256	6.5	0.16	F	2.57	6.0	0.39	84	0.30
512×512	8.4	1.25	F	14.17	8.2	2.24	246	6.03
1024×1024	10.3	10.09	F	82.03	9.0	15.05	F	46.06
$32 \times 32 \times 64$	5.6	0.25	61	0.38	5.4	0.98	62	0.22
$64 \times 64 \times 64$	7.0	1.33	F	15.64	6.6	6.43	224	5.43
$128 \times 128 \times 128$	8.8	15.35	F	176.11	6.5	28.08	F	102.56

increases as well. However, the CPU time cost per iteration turns out to be almost the same. Second, we fix the number of levels to $nlev = 5$, but increase the rank. In this case, the fill ratio and the time for building the preconditioner grow as before, whereas the number of iterations drops. In spite of the fewer iterations required, the total iteration time still increases with the rank since performing an iteration is now more expensive.

Next, we solve the indefinite problems by setting $c > 0$ in (4.1) and (4.2), which shifts the discretized negative Laplacian (a positive definite matrix) by subtracting the matrix sI for a certain s. In this set of experiments, we solve the 2-D problems with $s = 0.01$ and the 3-D problems with $s = 0.05$. MLR preconditioners are compared to ILDLT preconditioners along with the GMRES(40) method. For the 2-D problems, the number of levels is set to 4, whereas for the 3-D problems, the number of levels is set to 5. Moreover, for both sets of 2-D and 3-D problems, we use an MLR preconditioner of rank 5 for the first problem, but a rank of 7 for the other two. Results are shown in Table 4.4. For most problems, the ILDLT-GMRES method failed even though higher fill ratios were used compared to the SPD cases. In contrast, MLR preconditioners appear to be more effective. Except for the two largest cases in Table 4.4, the MLR-GMRES method achieved convergence and great savings in CPU time.

A few difficulties were encountered for large indefinite problems. Typically, an MLR preconditioner with many levels (e.g., 7 or 8) will not lead to convergence for these problems and this makes matrices at the last level still large. As a result, factoring them can be inefficient in terms of both CPU time and memory requirement. Furthermore, approximations of higher ranks were required compared to those for the SPD cases. This increases the memory requirement and also the CPU time of building an MLR preconditioner as well as the computational cost of applying it in iterations.

4.2. General matrices.

We selected 10 matrices from the University of Florida sparse matrix collection [10] for the following tests. Among these 7 are SPD matrices and 3 are symmetric indefinite. Table 4.5 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 an artificial one as $b = Ae$, where e is the vector of all ones.

MLR preconditioners were generalized via domain decomposition. In the current approach, for each nonleaf node i, we bisect the graph associated with matrix A_i by calling a graph partitioning algorithm `PartGraphRecursive` from Metis [16]. In the results shown in Tables 4.6–4.7, we include the CPU time for the domain decompositions for building an MLR preconditioner.

Table 4.5
Name, order (N), number of nonzeros (NNZ) and positive definiteness of the test matrices.

MATRIX	N	NNZ	SPD	DESCRIPTION
Andrews/Andrews	60,000	760,154	yes	computer graphics problem
Williams/cant	62,451	4,007,383	yes	FEM cantilever
UTEP/Dubcova2	65,025	1,030,225	yes	2-D/3-D PDE problem
Rothberg/cfd1	70,656	1,825,580	yes	CFD problem
Schmid/thermal1	82,654	574,458	yes	thermal problem
Rothberg/cfd2	123,440	3,085,406	yes	CFD problem
Schmid/thermal2	1,228,045	8,580,313	yes	thermal problem
Cote/vibrobox	12,328	301,700	no	vibroacoustic problem
Cunningham/qa8fk	66,127	1,660,579	no	3-D acoustics problem
Koutsovasilis/F2	71,505	5,294,285	no	structural problem

Table 4.6
Comparison between ICT *or* ILDLT *preconditioners and* MLR *preconditioners for solving general symmetric linear systems along with the* CG *or* GMRES(40) *method. Scalar balancing is used.*

MATRIX	ICT/ILDLT				MLR-CG/GMRES					
	fill	p-t	its	i-t	rank	nlev	fill	p-t	its	i-t
Andrews	2.6	0.44	32	0.16	2	6	2.3	1.32	29	0.08
cant	4.3	2.47	F	19.01	10	5	4.3	8.09	380	7.56
Dubcova2	1.4	0.14	42	0.21	4	4	1.4	0.65	46	0.10
cfd1	2.8	0.56	314	3.42	5	5	2.5	3.65	322	1.92
thermal1	3.1	0.15	108	0.51	2	5	3.1	0.75	110	0.32
cfd2	3.6	1.14	F	12.27	5	4	3.1	6.52	396	6.57
thermal2	5.3	4.11	148	20.45	5	5	5.3	15.42	175	14.63
vibrobox	3.3	0.19	F	1.06	10	4	2.9	0.50	259	0.32
qa8fk	1.8	0.58	56	0.60	2	8	1.6	2.53	61	0.29
F2	2.3	1.37	F	13.94	5	5	2.4	4.77	289	5.68

In section 2.5, we discussed three balancing options: the first one with a scalar, the second one with a diagonal matrix and the third one with an LU factorization. The third option did not yield results as good as with the other two and the related experimental results are omitted. For the first option, the scalar α defined in (2.27) is estimated by a power iteration.

Table 4.6 shows the performance of MLR preconditioners with the first balancing method, along with the number of levels and the rank used for each problem. The GMRES method with MLR preconditioners achieved convergence for all cases, whereas for many cases, it failed to converge with ICT or ILDLT preconditioners. Similar to the experiments with the model problems, MLR preconditioners required more CPU time to build than their ICT and ILDLT counterparts but achieved significant CPU time savings in the iteration phase. We note here that for two problems (`cant` and `vibrobox`), a higher rank was required than with the other problems. This might be due to the fact that in these problems the ratio of the number of interface nodes to the number of total nodes is higher than for the others, which results in the need of higher rank approximations to reach enough accuracy. Experimental results using smaller ranks did not lead to convergence within the maximum number of iterations for these two problems.

The last experiment we carried out was to test the MLR preconditioners with the second balancing method for the same set of matrices. The results are reported in Table 4.7. The ranks and the numbers of levels used here are the same as the ones displayed in Table 4.6. Compared to the MLR preconditioners with the first

TABLE 4.7
Comparison between ICT *or* ILDLT *preconditioners and* MLR *preconditioners for solving general symmetric linear systems along with the* CG *or* GMRES(40) *method. Diagonal matrices used for balancing.*

MATRIX	MLR-CG/GMRES			
	fill	p-t	its	i-t
Andrews	2.3	1.38	27	0.08
cant	4.3	7.89	253	5.30
Dubcova2	1.5	0.60	47	0.09
cfd1	2.3	3.61	244	1.45
thermal1	3.2	0.69	109	0.33
cfd2	3.1	4.70	312	4.70
thermal2	5.4	15.15	178	14.96
vibrobox	3.0	0.45	183	0.22
qa8fk	1.6	2.33	75	0.36
F2	2.5	4.17	371	7.29

balancing method, these preconditioners require almost the same amount of memory and similar CPU time to build. However, in six out of the ten cases this option led to fewer steps to converge, significantly so (at least 21% fewer) in four of the cases.

5. Conclusions.

This paper presented a preconditioning method for solving symmetric sparse linear systems, based on a recursive multilevel low-rank approximation approach. Experimental results indicate that for SPD systems, the proposed preconditioner can be a more efficient alternative to one based on incomplete factorization in terms of the iteration time. Moreover, this preconditioner appears to be more robust than incomplete factorization-based methods for indefinite problems. Recall that incomplete factorization-based methods often deliver unstable factors when the original matrix is indefinite which renders them ineffective for such cases. In contrast, MLR preconditioners compute an approximate inverse to A and as such they are not prone to any form of instability. On the negative side, building an MLR preconditioner can be time consuming, although two mitigating factors should be taken into account. First, what was demonstrated in the experiments is a first real implementation and there are many possibilities for further improvements. Second, we note that the proposed method is especially targeted at massively parallel machines since the computation required for building and for applying the preconditioner can be easily vectorized. As such, the set-up phase is likely to be far more advantageous than a factorization-based one which tends to be much more sequential; see, e.g., [19]. Work remains to be done to test this preconditioner in such environments.

Appendix. Eigenvalues of $\boldsymbol{T_y}$.

We consider the $n \times n$ matrix

$$T = \text{Tridiag}[-1, 2, -1] \tag{A.1}$$

whose main diagonal entries are all 2 and its codiagonals entries are all -1. It is assumed that n is even and we set $m = n/2$. As is well known [22, section 13.2], the eigenvalues of T are given by

$$\mu_k = 4 \sin^2 \frac{\theta_k}{2} \quad \text{with} \quad \theta_k = \frac{k\pi}{n+1}, \quad k = 1, 2, \ldots, n, \tag{A.2}$$

and each corresponding eigenvector u_k has components

$$u_{k,i} = \sin(i\ \theta_k), \quad i = 1, \ldots, n. \tag{A.3}$$

We are interested in the eigenvalues and eigenvectors of $\tilde{T} = T + ee^T$, where e was defined in section (3.2) as the vector of $\mathbb{R}^n$ that has entries equal to 1 in locations m and $m+1$ and zero elsewhere. This matrix is shown below for the case $n = 8$:

$$\tilde{T} = T + ee^T = \left[\begin{array}{cccc|cccc} 2 & -1 & & & & & & \\ -1 & 2 & -1 & & & & & \\ & -1 & 2 & -1 & & & & \\ & & -1 & 3 & & & & \\ \hline & & & & 3 & -1 & & \\ & & & & -1 & 2 & -1 & \\ & & & & & -1 & 2 & -1 \\ & & & & & & -1 & 2 \end{array}\right]. \tag{A.4}$$

As it turns out, the eigenvalues and eigenvectors of $\tilde{T}$ can be readily obtained from those of T. Consider an eigenvalue μ_k of T when k is even, which we write as $k = 2k'$.

In this situation, an eigenvector of T associated with the eigenvalue μ_k can be written in the form

$$u_k = \begin{bmatrix} v \\ -v^b \end{bmatrix},$$

where $.^b$ indicates a backward ordering, i.e., a permutation of a vector with the permutation $m, m-1, \ldots, 1$, so $v^b(i) = v(n+1-i)$ for $i = 1, \ldots, n$. This is because if we set $i = (n+1) - j$ in (A.3), we get

$$u_{k,i} = \sin\left(\frac{2k'(n+1-j)\pi}{n+1}\right) = -\sin\left(\frac{2k'j\pi}{n+1}\right) = -\sin\left(kj\theta_k\right) = -u_{k,j}$$

which shows that the first half of the components of u_k are the same as those of the second half, listed backward and with a negative sign. In particular note that $u_{k,m} = -u_{k,m+1}$ and so $e^T u_k = 0$. As a result, $(T - \mu_k I)u_k = 0$ implies that $(T - \mu_k I + ee^T)u_k = 0$, i.e., *each eigenpair* μ_k, u_k *of* T *for an even* k *is also an eigenpair of* $\tilde{T}$. This accounts for half of the eigenpairs of $\tilde{T}$. To get the other half (odd k), observe that the particular structure of $\tilde{T}$ shows that its eigenvalues are all double. The eigenvectors corresponding to the odd eigenvalues ($\mu_{k-1} = \mu_k$ for k even), can be easily seen to be given by

$$u_{k-1} = \begin{bmatrix} v \\ v^b \end{bmatrix}.$$

The whole set defined in this way does indeed form an orthonormal system of eigenvectors. This way of defining the set of eigenvectors is of course not unique. In the end we have the following eigenvalues defined for $k = 2, 4, \ldots, n-2, n$:

$$\mu_{k-1} = \mu_k = 4\sin^2\frac{k\pi}{2(n+1)} \ . \tag{A.5}$$

For the associated eigenvectors, we will now use the same notation as that of section 3.2, and, for a more convenient notation, we will swap the odd and even vectors,

$$\tilde{u}_{k-1} = \begin{bmatrix} v_k \\ -v_k^b \end{bmatrix}, \quad \tilde{u}_k = \begin{bmatrix} v_k \\ v_k^b \end{bmatrix}, \quad v_{k,i} = \sin\left(i\,\frac{k\pi}{n+1}\right), \quad i = 1\ldots, m \ . \tag{A.6}$$

Note that with this change of notation, $e^T\tilde{u}_k = 0$ for odd values of k, whereas $e^T\tilde{u}_k = 2\sin(m\theta_k)$ when k is even.

Acknowledgments. The authors would like to thank the anonymous reviewers for their valuable comments and suggestions. The second author acknowledges fruitful discussions with Lin Lin and Chao Yang which prompted the exploration of the model problem in section 3.2.

REFERENCES

[1] M. Ament, G. Knittel, D. Weiskopf, and W. 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, IEEE Computer Society, Los Alamitos, CA, 2010, pp. 583–592.

[2] P. R. Amestoy, T. A. Davis, and I. S. Duff, *An approximate minimum degree ordering algorithm*, SIAM J. Matrix Anal. Appl., 17 (1996), pp. 886–905.

[3] P. R. Amestoy, T. A. Davis, and I. S. Duff, *Algorithm* 837: *An approximate minimum degree ordering algorithm*, ACM Trans. Math. Software, 30 (2004), pp. 381–388.

[4] Z. Bai, J. Demmel, J. Dongarra, A. Ruhe, and H. van der Vorst, *Templates for the Solution of Algebraic Eigenvalue Problems: A Practical Guide*, Software Environ. Tools, SIAM, Philadelphia, 2000.

[5] N. Bell and M. 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, ACM, New York, 2009 pp. 1–11.

[6] J. Bolz, I. Farmer, E. Grinspun, and P. Schröoder, *Sparse matrix solvers on the GPU: Conjugate gradients and multigrid*, ACM Trans. Graph., 22 (2003), pp. 917–924.

[7] E. Chow and Y. Saad, *Approximate inverse techniques for block-partitioned matrices*, SIAM J. Sci. Comput., 18 (1997), pp. 1657–1675.

[8] E. Chow and Y. Saad, *Approximate inverse preconditioners via sparse-sparse iterations*, SIAM J. Sci. Comput., 19 (1998), pp. 995–1023.

[9] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*, 2nd ed., McGraw-Hill College, 2001.

[10] T. A. Davis, *University of Florida sparse matrix collection*, NA digest, 1994.

[11] T. A. Davis, *Direct Methods for Sparse Linear Systems*, Fundamentals of Algorithms, SIAM, Philadelphia, 2006.

[12] B. Engquist and L. Ying, *Sweeping preconditioner for the Helmholtz equation: Hierarchical matrix representation*, Comm. Pure Appl. Math., 64 (2011), pp. 697–735.

[13] H.-R. Fang and Y. Saad, *A filtered Lanczos procedure for extreme and interior eigenvalue problems*, SIAM J. Sci. Comput., 34 (2012), pp. A2220–A2246.

[14] G. H. Golub and C. F. Van Loan, *Matrix computations*, 3rd ed., Johns Hopkins University Press, Baltimore, MD, 1996.

[15] J. F. Grcar, *Analyses of the Lanczos Algorithm and of the Approximation Problem in Richardson's Method*, Ph.D. thesis, University of Illinois at Urbana-Champaign, Champaign, IL, 1981.

[16] G. Karypis and V. Kumar, *Metis - A Software Package for Partitioning Unstructured Graphs, Partitioning Meshes, and Computing Fill-Reducing Orderings of Sparse Matrices, Version* 4.0, Technical report, University of Minnesota, Department of Computer Science/Army HPC Research Center, 1998.

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

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

[19] R. Li and Y. Saad, *GPU-accelerated preconditioned iterative linear solvers*, J. Supercomput., 63 (2013), pp. 443–466.

[20] B. N. Parlett, *The Symmetric Eigenvalue Problem*, Classics Appl. Math., SIAM, Philadephia, 1998.

[21] B. N. Parlett and D. S. Scott, *The Lanczos algorithm with selective orthogonalization*, Math. Comp., 33 (1979), pp. 217–238.

[22] Y. Saad, *Iterative Methods for Sparse Linear Systems*, 2nd ed., SIAM, Philadelpha, 2003.

[23] Y. Saad and B. Suchomel, *ARMS: An algebraic recursive multilevel solver for general sparse linear systems*, Numer. Linear Algebra Appl., 9 (2002), pp. 359–378.

[24] H. D. Simon, *The Lanczos algorithm with partial reorthogonalization*, Math. Comp., 42 (1984), pp. 115–142.

[25] 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, Oxford, UK, 2010.

[26] J. M. Tang and Y. Saad, *Domain-decomposition-type methods for computing the diagonal of a matrix inverse*, SIAM J. Sci. Comput., 33 (2011), pp. 2823–2847.

[27] M. Wang, H. Klie, M. Parashar, and H. Sudan, *Solving sparse linear systems on Nvidia Tesla GPUs*, in ICCS '09: Proceedings of the 9th International Conference on Computational Science, Springer-Verlag, Berlin, Heidelberg, 2009, pp. 864–873.

[28] J. Xia, S. Chandrasekaran, M. Gu, and X. S. Li, *Fast algorithms for hierarchically semiseparable matrices*, Numer. Linear Algebra Appl., 17 (2010), pp. 953–976.

[29] J. Xia and M. Gu, *Robust approximate Cholesky factorization of rank-structured symmetric positive definite matrices*, SIAM J. Matrix Anal. Appl., 31 (2010), pp. 2899–2920.