LOW-RANK CORRECTION METHODS FOR ALGEBRAIC DOMAIN DECOMPOSITION PRECONDITIONERS *

RUIPENG LI † AND YOUSEF SAAD†

Abstract. This paper presents a parallel preconditioning method for distributed sparse linear systems, based on an approximate inverse of the original matrix, that adopts a general framework of distributed sparse matrices and exploits the domain decomposition method and low-rank corrections. The domain decomposition approach decouples the matrix and once inverted, a low-rank approximation is applied by exploiting the Sherman-Morrison-Woodbury formula, which yields two variants of the preconditioning methods. The low-rank expansion is computed by the Lanczos procedure with reorthogonalizations. Numerical experiments indicate that, when combined with Krylov subspace accelerators, this preconditioner can be efficient and robust for solving symmetric sparse linear systems. Comparisons with other distributed-memory preconditioning methods are presented.

Key words. Sherman-Morrison-Woodbury formula, low-rank approximation, distributed sparse linear systems, parallel preconditioner, incomplete LU factorization, Krylov subspace method, domain decomposition

1. Introduction. Preconditioning distributed sparse linear system remains a challenging problem in high-performance multi-processor environments. Simple algorithms such as the Additive Schwarz method are widely used and they usually yield good parallelism. A well-known problem with these preconditioners is that they often require a large number of iterations when the number of domains used is large. As a result, the benefits of increased parallelism is often outweighed by the increased number of iterations. Algebraic MultiGrid (AMG) methods have achieved a good success and can be extremely fast when they work. However, their success is still somewhat restricted to certain types of problems. Methods based on the global Schur complement technique such as the Parallel Algebraic Recursive Solver (pARMS) [26], which consist of eliminating interior variables first and then focus on solving in some ways the interface variables are designed to be general-purpose methods. The difficulty here is to find effective and efficient preconditioners for the distributed global Schur complement system. In the approach proposed in the present work, we do not try to solve the global Schur complement system exactly or even form it. Instead, we exploit the Sherman-Morrison-Woodbury (SMW) formula and a low-rank property to define an approximate inverse type preconditioner.

Low-rank approximations have recently gained popularity as a means to compute preconditioners. For instance, LU factorizations or inverse matrices using the $\mathcal{H}$-matrix format or the closely related Hierarchically Semi-Separable (HSS) matrix format rely on representing certain off-diagonal blocks by low-rank matrices [12, 22, 23, 36, 37]. The main idea of this work is inspired by the recursive Multilevel Low-Rank (MLR) preconditioner [24] targeted at SIMD-type parallel machines such as those equipped with Graphic Processing Units (GPUs), where traditional ILU-type preconditioners have difficulty reaching good performance [25]. Here, we adapt and extend this idea to the framework of distributed sparse matrices via domain decomposition methods. We refer to a preconditioner obtained by this approach as a Domain Decomposition based Low-Rank (DD-LR) preconditioner. This paper considers only symmetric matrices. Extensions to the nonsymmetric case are possi-

*This work was supported by NSF under grant NSF/DMS-1216366.

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

ble and will be explored in our future work. The paper is organized as follows: In Section 2, we briefly introduce the distributed sparse linear systems and discuss the domain decomposition framework. Section 3 presents the two proposed strategies for using low-rank approximations in the SMW formula. Parallel implementation details are presented in Section 4. Numerical results of model problems and general symmetric linear systems are presented in Section 5, and we conclude in Section 6.

2. Background: Distributed sparse linear systems. The parallel solution of a linear systems of the form

$$Ax = b, \tag{2.1}$$

where A is an $n \times n$ large sparse *symmetric* matrix, typically begins by subdividing the problem into p parts with the help of a graph partitioner [7, 16, 18, 20, 29, 30]. Generally, this consists of assigning sets of equations along with the corresponding right-hand side values to subdomains. If equation number i is assigned to a given subdomain, then it is common to also assign unknown number i to the same subdomain. Thus, each process holds a set of equations (rows of the linear system) and vector components associated with these rows. This viewpoint is prevalent when taking a purely algebraic viewpoint for solving systems of equations that arise from Partial Differential Equations (PDEs) or general unstructured sparse matrices.

2.1. The local systems. In this paper we partition the problem using an *edge separator* as is done in the pARMS method for example. As shown in Figure 2.1, once a graph is partitioned, three types of unknowns appear: (1) Interior unknowns that are coupled only with local unknowns; (2) Local interface unknowns that are coupled with both non-local (external) and local variables; and (3) External interface unknowns that belong to other subdomains and are coupled with local interface variables. The

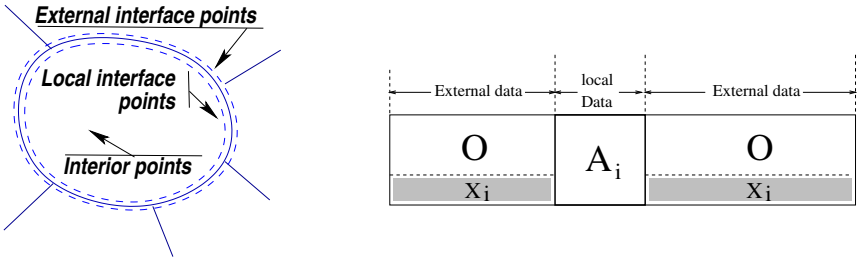

FIG. 2.1. *A local view of a distributed sparse matrix (left) and its matrix representation (right).*

rows of the matrix assigned to subdomain i can be split into two parts: a local matrix A_i which acts on the local variables and an interface matrix X_i which acts on the external interface variables. Local variables in each subdomain are often reordered so that the interface variables are listed after the interior variables. Thus, each vector of local unknowns x_i is split into two parts: subvector u_i of the internal components followed by subvector y_i of the local interface components. The right-hand-side vector b_i is conformingly split into the subvectors f_i and g_i. When the blocks are partitioned according to this splitting, the local system of equations can be written as

$$\underbrace{\begin{pmatrix} B_i & E_i \\ E_i^T & C_i \end{pmatrix}}_{A_i} \underbrace{\begin{pmatrix} u_i \\ y_i \end{pmatrix}}_{x_i} + \begin{pmatrix} 0 \\ \sum_{j \in N_i} E_{ij} y_j \end{pmatrix} = \underbrace{\begin{pmatrix} f_i \\ g_i \end{pmatrix}}_{b_i}. \tag{2.2}$$

Here, N_i is a set of the indices of the subdomains that are neighbors to subdomain i. The term $E_{ij}y_j$ is a part of the product which reflects the contribution to the local equations from the neighboring subdomain j. The result of this multiplication affects only the local interface equations, which is indicated by the zero in the top part of the second term of the left-hand side of (2.2).

2.2. The interface and Schur complement matrices. The local system (2.2) is naturally split in two parts: the first part represented by the term $A_i x_i$ involves only the local variables and the second part contains the couplings between the local interface variables and the external interface variables. Furthermore, the second row of the equations in (2.2)

$$E_i^T u_i + C_i y_i + \sum_{j \in N_i} E_{ij} y_j = g_i, \tag{2.3}$$

defines both the inner-domain and the inter-domain couplings. It couples the interior variables u_i with the local interface variables y_i and the external ones, y_j. An alternative way to order a global system is to group the interior variables of all the subdomains together and all the interface variables together as well. The action of the operation on the left-hand side of (2.3) on the vector of all interface variables, i.e., the vector $y^T = [y_1^T, y_2^T, \cdots, y_p^T]$, can be gathered into the following matrix C,

$$C = \begin{pmatrix} C_1 & E_{12} & \dots & E_{1p} \\ E_{21} & C_2 & \dots & E_{2p} \\ \vdots & & \ddots & \vdots \\ E_{p1} & E_{p,2} & \dots & C_p \end{pmatrix}. \tag{2.4}$$

Thus, if we reorder the equations so that the u_i's are listed first followed by the y_i's, we obtain a global system which has the following form:

$$\left(\begin{array}{cccc|c} B_1 & & & & \hat{E}_1 \\ & B_2 & & & \hat{E}_2 \\ & & \ddots & & \vdots \\ & & & B_p & \hat{E}_p \\ \hline \hat{E}_1^T & \hat{E}_2^T & \dots & \hat{E}_p^T & C \end{array}\right) \begin{pmatrix} u_1 \\ u_2 \\ \vdots \\ u_p \\ y \end{pmatrix} = \begin{pmatrix} f_1 \\ f_2 \\ \vdots \\ f_p \\ g \end{pmatrix}, \tag{2.5}$$

where $\hat{E}_i$ is expanded from E_i by adding zeros and on the right-hand side, $g^T = [g_1^T, g_2^T, \cdots, g_p^T]$. Writing the system in the form (2.2) is commonly adopted in practice when solving parallel/distributed sparse systems, while the form (2.5) is more convenient for analysis. In what follows, we will assume that the global matrix is put in the form of (2.5). The form (2.2) will return in the discussions of Section 4, which deal with parallel implementations.

We will assume that each subdomain i has d_i interior variables and s_i interface variables, i.e., the length of u_i is d_i and that of y_i is s_i. We will denote by s the size of y, i.e., $s = s_1 + s_2 + \cdots + s_p$. With this notation, each E_i is a matrix of size $d_i \times s_i$. The expanded version of this matrix, $\hat{E}_i$ is of size $d_i \times s$ and its columns outside of those corresponding to the variables in y_i are zero. An illustration for 4 subdomains is shown in Figure 2.2.

A popular way of solving a global system put into the form of (2.5) is to exploit Schur complement techniques which eliminate the interior variables u_i first and then

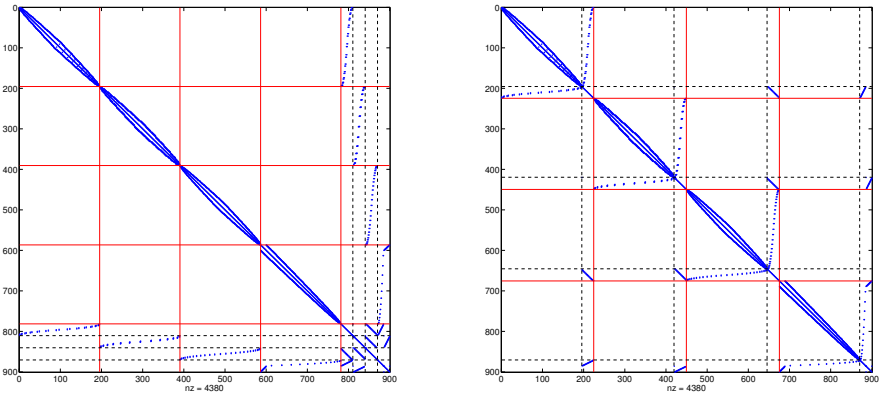

FIG. 2.2. *An example of a 2-D Laplacian matrix which is partitioned into* 4 *subdomains and reordered according to (2.5) (left) and (2.2) (right) respectively.*

focus on solving in some way for the interface variables. The interior variables can then be easily recovered by back substitution. Assuming that B_i is nonsingular, u_i in (2.2) can be eliminated by means of the first equation: $u_i = B_i^{-1}(f_i - E_i y_i)$ which yields, upon substitution in (2.3),

$$S_i y_i + \sum_{j \in N_i} E_{ij} y_j = g_i - E_i^T B_i^{-1} f_i \equiv g_i', \tag{2.6}$$

in which S_i is the *local* Schur complement,

$$S_i = C_i - E_i^T B_i^{-1} E_i.$$

When written for each subdomain i, (2.6) yields the *global* Schur complement system that involves only the interface unknown vectors y_i and the reduced system has a natural block structure,

$$\begin{pmatrix} S_1 & E_{12} & \dots & E_{1p} \\ E_{21} & S_2 & \dots & E_{2p} \\ \vdots & & \ddots & \vdots \\ E_{p1} & E_{p,2} & \dots & S_p \end{pmatrix} \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_p \end{pmatrix} = \begin{pmatrix} g_1' \\ g_2' \\ \vdots \\ g_p' \end{pmatrix}. \tag{2.7}$$

Each of the diagonal blocks in this system is the local Schur complement matrix S_i, which is dense in general. The off-diagonal blocks E_{ij} are identical with those of the local system (2.4) and are sparse. A key idea here is to (approximately) solve the reduced system (2.7) efficiently. For example, in pARMS [26] efficient preconditioners are developed based on forming an approximation to the Schur complement system and then approximately solving (2.7) and then extracting the internal variables u_i. This defines a preconditioning operation for the global system. In the method proposed in this paper we do not try to solve the global Schur complement system or even form it. Instead, an approximate inverse preconditioner to the original matrix is obtained by exploiting a low-rank property and the SMW formula.

3. Domain Decomposition with local low-rank corrections. The coefficient matrix of the system (2.5) is of the form

$$A \equiv \begin{pmatrix} B & \hat{E} \\ \hat{E}^T & C \end{pmatrix}, \tag{3.1}$$

where $B \in \mathbb{R}^{m\times m}$, $\hat{E} \in \mathbb{R}^{m\times s}$ and $C \in \mathbb{R}^{s\times s}$. Here, we abuse notation by using the same symbol A to represent the permuted version of the matrix in (2.1). The goal of this section is to build a preconditioner for the matrix (3.1).

3.1. Splitting. We begin by splitting matrix A as follows

$$A = \begin{pmatrix} B & \hat{E} \\ \hat{E}^T & C \end{pmatrix} = \begin{pmatrix} B & \\ & C \end{pmatrix} + \begin{pmatrix} & \hat{E} \\ \hat{E}^T & \end{pmatrix}, \tag{3.2}$$

and define the $n \times s$ matrix,

$$E \equiv \begin{pmatrix} \alpha^{-1}\hat{E} \\ -\alpha I \end{pmatrix}, \tag{3.3}$$

where I is the $s \times s$ identity matrix and α is a parameter. Then from (3.2) we immediately get the identity,

$$\left[\begin{array}{c|c} B & \hat{E} \\ \hline \hat{E}^T & C \end{array}\right] = \left[\begin{array}{c|c} B + \alpha^{-2}\hat{E}\hat{E}^T & 0 \\ \hline 0 & C + \alpha^2 I \end{array}\right] - EE^T. \tag{3.4}$$

A remarkable property is that the operator $\hat{E}\hat{E}^T$ is *local* in that it does not involve inter-domain couplings. Specifically, we have the following proposition.

PROPOSITION 3.1. *Consider the matrix $X = \hat{E}\hat{E}^T$ and its blocks X_{ij} associated with the same blocking as for the matrix in (2.5). Then, for $1 \le i, j \le p$ we have:*

$$\begin{aligned} X_{ij} &= 0, \quad \text{for} \quad i \neq j \\ X_{ii} &= E_i E_i^T. \end{aligned} \tag{3.5}$$

Proof. This follows from the fact that the columns of $\hat{E}$ associated with different subdomains are structurally orthogonal. This is illustrated on the left side picture of Figure 2.2. ∎

Thus, we can write

$$A = A_0 - EE^T, \quad A_0 = \begin{pmatrix} B + \alpha^{-2}\hat{E}\hat{E}^T & \\ & C + \alpha^2 I \end{pmatrix} \in \mathbb{R}^{n\times n}, \tag{3.6}$$

with E defined in (3.3). From (3.6) and the SMW formula, we can derive the expression for the inverse of A. First define,

$$G = I - E^T A_0^{-1} E. \tag{3.7}$$

Then, we have

$$A^{-1} = A_0^{-1} + A_0^{-1}E(\underbrace{I - E^T A_0^{-1} E}_{G})^{-1} E^T A_0^{-1} \equiv A_0^{-1} + A_0^{-1} E G^{-1} E^T A_0^{-1}. \tag{3.8}$$

Note that the matrix C is often strongly diagonally dominant and the parameter α can serve to improve diagonal dominance in the indefinite cases.

3.2. Low-rank approximation to the G matrix. In this section we will only consider the cases when A is symmetric positive definite (SPD). A preconditioner of the form

$$M^{-1} = A_0^{-1} + (A_0^{-1}E)\tilde{G}^{-1}(E^T A_0^{-1}) \tag{3.9}$$

can be readily obtained from (3.8) if we had an approximation $\tilde{G}^{-1}$ to G^{-1}. Note that the application of this preconditioner will involve two solves with A_0 instead of only one. It will also involve a solve with $\tilde{G}$ which operates on the interface variables. Let us, at least formally, assume that we know the spectral factorization of $E^T A_0^{-1} E$

$$H \equiv E^T A_0^{-1} E = U \Lambda U^T, \tag{3.10}$$

where $H \in \mathbb{R}^{s\times s}$, U is unitary, and Λ is diagonal. From (3.6) we have $A_0 = A + EE^T$, and since A is SPD, A_0 is also SPD. Then, H in (3.10) is at least symmetric positive semidefinite (SPSD) and the following lemma shows that its eigenvalues are less than one.

LEMMA 3.2. *Let $H = E^T A_0^{-1} E$ and assume that A is SPD and the matrix $I - H$ is nonsingular. Then we have $0 \le \lambda < 1$, for each eigenvalue λ of H.*

Proof. From (3.8), we have,

$$E^T A^{-1} E = H + H(I-H)^{-1}H = H\left(I + (I-H)^{-1}H\right) = H(I-H)^{-1}.$$

Since A is SPD, so $E^T A^{-1} E$ is SPSD. Thus, the eigenvalues of $H(I-H)^{-1}$ are nonnegative, i.e., $\lambda/(1-\lambda) \ge 0$. So, we have $0 \le \lambda < 1$. □

The goal now is to see what happens if we replace Λ by a diagonal matrix $\tilde{\Lambda}$. This will include the situation when a low-rank approximation is used for G but it can also include other possibilities. Suppose that H is approximated as follows:

$$H \approx U \tilde{\Lambda} U^T. \tag{3.11}$$

Then, from the SMW formula, the corresponding approximation to G^{-1} is:

$$G^{-1} \approx \tilde{G}^{-1} \equiv (I - U\tilde{\Lambda}U^T)^{-1} = I + U[(I-\tilde{\Lambda})^{-1} - I]U^T. \tag{3.12}$$

Note in passing that the above expression can be simplified to $U(I-\tilde{\Lambda})^{-1}U^T$. However, we keep the above form because it will still be valid when U has only k ($k < s$) columns and $\tilde{\Lambda}$ is $k \times k$ diagonal, in which case we denote by G_k^{-1} the approximation in (3.12). At the same time, the exact G can be obtained as a special case of (3.12), where $\tilde{\Lambda}$ is simply equal to Λ. Then we have

$$A^{-1} = A_0^{-1} + (A_0^{-1}E)G^{-1}(E^T A_0^{-1}) \tag{3.13}$$

and

$$M^{-1} = A_0^{-1} + (A_0^{-1}E)G_k^{-1}(E^T A_0^{-1}), \tag{3.14}$$

from which it follows by subtraction that

$$A^{-1} - M^{-1} = (A_0^{-1}E)[G^{-1} - G_k^{-1}](E^T A_0^{-1}),$$

and therefore,

$$AM^{-1} = I - A(A_0^{-1}E)[G^{-1} - G_k^{-1}](E^T A_0^{-1}). \tag{3.15}$$

A first consequence of (3.15) is that there will be at lease m eigenvalues of AM^{-1} that are equal to one, where $m = n - s$ is the dimension of matrix B in (3.1) or in other words, the total number of the interior variables. From (3.12) we obtain

$$G^{-1} - G_k^{-1} = U[(I - \Lambda)^{-1} - (I - \tilde{\Lambda})^{-1}]U^T. \tag{3.16}$$

The simplest selection of $\tilde{\Lambda}$ is the one that ensures that the k largest eigenvalues of $(I - \tilde{\Lambda})^{-1}$ match the largest eigenvalues of $(I - \Lambda)^{-1}$. This simply minimizes the 2-norm of (3.16) under the assumption that the approximation in (3.11) is of rank k. Assume that the eigenvalues of H are $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_s$, which means that the diagonal entries $\tilde{\lambda}_i$ of $\tilde{\Lambda}$ are selected such that

$$\tilde{\lambda}_i = \begin{cases} \lambda_i & \text{if} \quad i \leq k \\ 0 & \text{otherwise} \end{cases}. \tag{3.17}$$

Observe that from (3.16) the eigenvalues of $G^{-1} - G_k^{-1}$ are

$$\begin{cases} 0 & \text{if} \quad i \leq k \\ (1 - \lambda_i)^{-1} - 1 & \text{otherwise} \end{cases}.$$

Thus, from (3.15) we can infer that k more eigenvalues of AM^{-1} will take the value one in addition to the existing m ones revealed above independently of the choice of $\tilde{G}^{-1}$. Noting that $(1 - \lambda_i)^{-1} - 1 = \lambda_i/(1 - \lambda_i) \geq 0$, since $0 \leq \lambda_i < 1$ and we can say that the remaining $s-k$ eigenvalues of AM^{-1} will be between 0 and 1. Therefore, the result in this case is that the preconditioned matrix AM^{-1} in (3.15) will have $m + k$ eigenvalues equal to one, and $s - k$ other eigenvalues between 0 and 1.

From an implementation point of view, it is clear that a full diagonalization of H is not needed. All we need is U_k the $s \times k$ matrix consisting of the first k columns of U, along with the diagonal matrix Λ_k of the corresponding eigenvalues $\lambda_1, \cdots, \lambda_k$. Then, noting that (3.12) is still valid with U replaced by U_k and Λ replaced by Λ_k, we can get the approximation G_k and its inverse directly:

$$G_k = I - U_k \Lambda_k U_k^T, \quad G_k^{-1} = I + U_k[(I - \tilde{\Lambda}_k)^{-1} - I]U_k^T. \tag{3.18}$$

It may have become clear to the reader that it is possible to select $\tilde{\Lambda}$ so that AM^{-1} will have eigenvalues larger than one. Consider defining $\tilde{\Lambda}$ such that

$$\tilde{\lambda}_i = \begin{cases} \lambda_i & \text{if} \quad i \leq k \\ \theta & \text{if} \quad i > k \end{cases},$$

and denote by $G_{k,\theta}^{-1}$ the related analogue of (3.18). Then, from (3.16) the eigenvalues of $G^{-1} - G_{k,\theta}^{-1}$ are

$$\begin{cases} 0 & \text{if} \quad i \leq k \\ (1 - \lambda_i)^{-1} - (1 - \theta)^{-1} & \text{if} \quad i > k \end{cases}. \tag{3.19}$$

Note that for $i > k$,

$$\frac{1}{1 - \lambda_i} - \frac{1}{1 - \theta} = \frac{\lambda_i - \theta}{(1 - \lambda_i)(1 - \theta)},$$

and these eigenvalues can be made negative by selecting $\lambda_{k+1} \leq \theta < 1$ and the choice that yields the smallest 2-norm is $\theta = \lambda_{k+1}$. The earlier definition of Λ_k in (3.17) which truncates the eigenvalues of H to zero corresponds to selecting $\theta = 0$.

THEOREM 3.3. *Assume that A is SPD and θ is selected so that $\lambda_{k+1} \leq \theta < 1$. Then the eigenvalues η_i of AM^{-1} are such that,*

$$1 \leq \eta_i \leq 1 + \frac{1}{1-\theta} \|A^{1/2} A_0^{-1} E\|_2^2. \tag{3.20}$$

Furthermore, the term $\|A^{1/2} A_0^{-1} E\|_2^2$ is bounded from above by a constant:

$$\|A^{1/2} A_0^{-1} E\|_2^2 \leq \frac{1}{4}. \tag{3.21}$$

Proof. We rewrite (3.15) as $AM^{-1} = I + A(A_0^{-1}E)[G_k^{-1} - G^{-1}](E^T A_0^{-1})$ or upon applying a similarity transformation with $A^{1/2}$

$$A^{1/2} M^{-1} A^{1/2} = I + (A^{1/2} A_0^{-1} E)[G_k^{-1} - G^{-1}](E^T A_0^{-1} A^{1/2}). \tag{3.22}$$

From (3.19) we see that for $j \leq k$ we have $\lambda_j(G_k^{-1} - G^{-1}) = 0$ and for $j > k$,

$$0 \leq \lambda_j(G_k^{-1} - G^{-1}) = (1-\theta)^{-1} - (1-\lambda_j)^{-1} \leq (1-\theta)^{-1}.$$

This is because $1/(1-t)$ is an increasing function and for $j > k$, $0 \leq \lambda_j \leq \lambda_{k+1} \leq \theta$. The rest of the proof follows by taking the Rayleigh quotient of an arbitrary vector x and utilizing (3.22).

For the second part, first note that $\|A^{1/2} A_0^{-1} E\|_2^2 = \rho\left(E^T A_0^{-1} A A_0^{-1} E\right)$, where $\rho(\cdot)$ denotes the spectral radius of a matrix. Then, from $A = A_0 - EE^T$, we have

$$E^T A_0^{-1} A A_0^{-1} E = E^T A_0^{-1} E - \left(E^T A_0^{-1} E\right)\left(E^T A_0^{-1} E\right) \equiv H - H^2.$$

Lemma 3.2 states that each eigenvalue λ of H is such that $0 \leq \lambda < 1$. Hence, an eigenvalue μ of $E^T A_0^{-1} A A_0^{-1} E$ satisfies $\mu = \lambda - \lambda^2$, which is between 0 and 1/4 for $\lambda \in [0, 1)$. This gives the desired bound $\|A^{1/2} A_0^{-1} E\|_2^2 \leq 1/4$. ∎

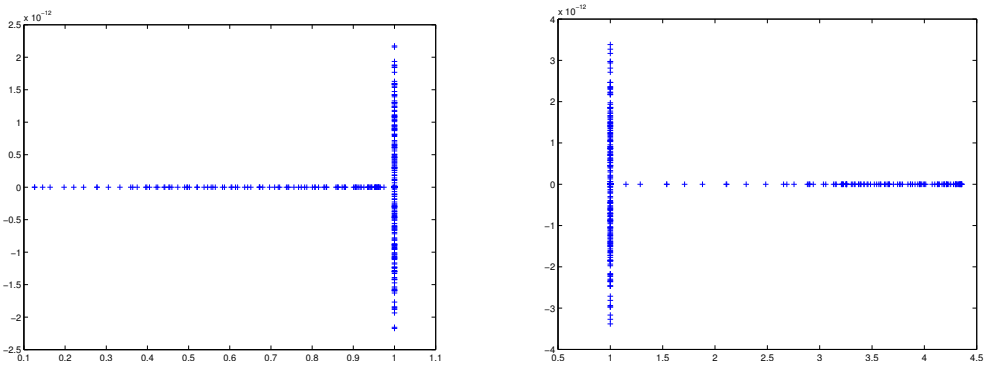

FIG. 3.1. *Eigenvalues of AM^{-1} for the cases when $\theta = 0$ (left) and $\theta = \lambda_{k+1}$ (right) using $k = 5$ eigenvectors for a 900×900 2-D Laplacian matrix, where $\alpha = 1$.*

The term $\|A^{1/2} A_0^{-1} E\|_2^2$ does not exceed 1/4, regardless of the mesh size and regardless of α, and it is close to 1/4 for Laplacian matrices. An illustration of

the spectra of AM^{-1} for the two cases when $\theta = 0$ and $\theta = \lambda_{k+1}$ with $k = 5$ is shown in Figure 3.1. Note that the tiny imaginary parts are due to the fact that the matrix AM^{-1} is not symmetric (but has real eigenvalues). The original matrix is a 900×900 2-D Laplacian obtained from a finite difference discretization of a square domain using 30 mesh points in each direction. The number of the subdomains used is four, resulting in 119 interface variables. The reordered matrices associated with this example are shown in Figure 2.2. For the case with $\alpha = 1$, we have $\theta = \lambda_6 \approx 0.93492$ and $\|A^{1/2}A_0^{-1}E\|_2^2 \approx 0.24996$, and thus the bound of the eigenvalues of AM^{-1} given by (3.20) is 4.8413, which is fairly close to the largest eigenvalue, which is 4.3581 (cf. the figure on the right). In the case when $\alpha = 2$, we have $\theta = \lambda_6 \approx 0.93987$ and $\|A^{1/2}A_0^{-1}E\|_2^2 \approx 0.25000$, and accordingly the eigenvalue bound given by (3.20) is 5.1575 whereas the largest eigenvalue is 4.6724. If $\alpha = 0.5$ and $\theta = \lambda_6 \approx 0.96945$, then $\|A^{1/2}A_0^{-1}E\|_2^2 \approx 0.24999$, and the bound is 9.1840, compared with the largest eigenvalue 8.6917. Therefore, setting $\alpha = 1$ gives the best eigenvalues clustering for the preconditioned matrix, which is typical for positive definite problems.

We now address the implementation of the preconditioner related to this second choice. Again all that is needed are U_k, Λ_k and the $(k+1)$-st eigenvalue λ_{k+1}. We can show an analogue to the expression (3.18).

PROPOSITION 3.4. *The following expression for $G_{k,\theta}^{-1}$ holds:*

$$G_{k,\theta}^{-1} = \frac{1}{1-\theta} I \; + \; U_k \left[(I - \Lambda_k)^{-1} - (1-\theta)^{-1} I \right] U_k^T. \tag{3.23}$$

Proof. We write $U = [U_k, W]$, where U_k is as before and W contains the remaining columns $u_{k+1}, \cdots, u_s$. Note that W is not available but we use the fact that $WW^T = I - U_k U_k^T$ for the purpose of this proof. With this, (3.12) becomes:

$$\begin{aligned}
G_{k,\theta}^{-1} &= I + [U_k, W] \begin{pmatrix} (I-\Lambda_k)^{-1} - I & 0 \\ 0 & ((1-\theta)^{-1} - 1) I \end{pmatrix} [U_k, W]^T \\
&= I + U_k \left[(I - \Lambda_k)^{-1} - I \right] U_k^T + \left[(1-\theta)^{-1} - 1 \right] WW^T \\
&= I + U_k \left[(I - \Lambda_k)^{-1} - I \right] U_k^T + \left[(1-\theta)^{-1} - 1 \right] (I - U_k U_k^T) \\
&= \frac{1}{1-\theta} I \; + \; U_k \left[(I - \Lambda_k)^{-1} - (1-\theta)^{-1} I \right] U_k^T.
\end{aligned}$$

□

PROPOSITION 3.5. *Let the assumptions of Lemma 3.2 be satisfied. Then the preconditioner given by* (3.14) *with the matrix $G_{k,\theta}^{-1}$ defined as in* (3.23) *is well-defined and SPD for the cases in which $0 \le \theta \le \lambda_{k+1}$.*

Proof. Recall from Lemma 3.2, we have $0 \le \lambda < 1$ for every eigenvalue of H and thus $G_{k,\theta}^{-1}$ is well-defined and SPD if $0 \le \theta \le \lambda_{k+1}$. Therefore, the preconditioner M^{-1} in (3.14) is also SPD. □

We refer to the preconditioner given by (3.14) with $G_k = G_{k,\theta}$ in (3.23) as a *one-sided* domain-decomposition based low-rank approximation preconditioner, abbreviated by the DD-LR-1 preconditioner.

3.3. Two-sided low-rank approximation. The method to be presented in this section will use low-rank approximations for more terms in (3.8) than DD-LR-1. This will result in a preconditioner that has a simpler form and is less expensive to apply but less accurate in general. Suppose we have the matrix $A_0^{-1}E$ factored as

$$A_0^{-1}E = UV^T, \tag{3.24}$$

with $U \in \mathbb{R}^{n\times s}$, $V \in \mathbb{R}^{s\times s}$ and $V^TV = I$ as obtained from the singular value decomposition (SVD). Then, for the matrix G in (3.7), we have the following lemma.

LEMMA 3.6. *Let $G = I - E^TA_0^{-1}E$ as defined by (3.7) be nonsingular. Then,*

$$G^{-1} = I + V\left(I - U^TEV\right)^{-1}U^TE. \tag{3.25}$$

Furthermore, the following relation holds,

$$V^TG^{-1}V = \left(I - U^TEV\right)^{-1}. \tag{3.26}$$

Proof. For G^{-1}, we can write

$$G^{-1} = \left(I - (E^TA_0^{-1})E\right)^{-1} = \left(I - VU^TE\right)^{-1} = I + V\left(I - U^TEV\right)^{-1}U^TE.$$

Relation (3.26) follows from

$$\begin{aligned} V^TG^{-1}V &= V^T\left(I + V\left(I - U^TEV\right)^{-1}U^TE\right)V = I + \left(I - U^TEV\right)^{-1}U^TEV \\ &= \left(I - U^TEV\right)^{-1}. \end{aligned} \tag{3.27}$$

This completes the proof. ∎

From (3.24), we can obtain the best 2-norm rank-k approximation to $A_0^{-1}E$ of the form

$$A_0^{-1}E \approx U_kV_k^T, \quad V_k^TV_k = I \tag{3.28}$$

where $U_k \in \mathbb{R}^{n\times k}$ and $V_k \in \mathbb{R}^{s\times k}$ consist of the first k columns of U and V respectively. For an approximation to G, we define the matrix G_k as

$$G_k = I - V_kU_k^TE\ . \tag{3.29}$$

Then, the expression of A^{-1} in (3.8) will yield the preconditioner:

$$M^{-1} = A_0^{-1} + U_k(V_k^TG_k^{-1}V_k)U_k^T, \tag{3.30}$$

which means that we can build an approximate inverse based on a low-rank correction of the form

$$M^{-1} = A_0^{-1} + U_kH_kU_k^T \quad \text{with} \quad H_k = V_k^TG_k^{-1}V_k. \tag{3.31}$$

Note that Lemma 3.6 will also hold if U and V are replaced by U_k and V_k. As a result, the matrix H_k has an alternative expression that is more amenable to computations. Specifically, we can show the following lemma.

LEMMA 3.7. *Let G_k be defined by (3.29) and assume that the matrix $I - U_k^TEV_k$ is nonsingular. Then,*

$$G_k^{-1} = I + V_k\hat{H}_kU_k^TE \quad \textit{with} \quad \hat{H}_k = (I - U_k^TEV_k)^{-1}. \tag{3.32}$$

Furthermore, the following relation holds:

$$V_k^TG_k^{-1}V_k = \hat{H}_k \tag{3.33}$$

i.e., the matrix H_k in (3.31) and the matrix $\hat{H}_k$ are equal.

Proof. A proof can be directly obtain from the proof of Lemma 3.6 by replacing matrices U,V and G with U_k,V_k and G_k respectively. ∎

The alternative expression (3.31) avoids the use of V_k. Hence, an application of the preconditioner requires one solve with A_0 and a low-rank correction carried out by using U_k and H_k. Since $A_0^{-1}E$ is approximated on both sides of G in (3.8), we refer to the preconditioner (3.31) as a *two-sided* domain-decomposition based low-rank approximation preconditioner and use the abbreviation DD-LR-2.

PROPOSITION 3.8. *Assume that $U_kV_k^T$ in* (3.28) *is the best 2-norm rank-k approximation to the matrix $A_0^{-1}E$ with $V_k^TV_k = I$, and that A_0 is SPD. Then the preconditioner given by* (3.30) *is well-defined and SPD if and only if $\rho(U_k^TEV_k) < 1$.*

Proof. The proof follows immediately from Proposition 3.2 in [24], which shows the symmetry of H_k, and Proposition 3.4 and Theorem 3.6 in [24], which show the if-and-only-if condition. ∎

Next, we will show that the eigenvalues of the preconditioned matrix AM^{-1} with M^{-1} defined by (3.30) are between zero and one. Suppose that $U_kV_k^T$ is obtained by (3.28), so that we have $\left(A_0^{-1}E\right)V_k = U_k$. Then, the preconditioner in (3.31) reads

$$\begin{aligned} M^{-1} &= A_0^{-1} + U_kH_kU_k^T = A_0^{-1} + \left(A_0^{-1}E\right)V_kH_kV_k^T\left(E^TA_0^{-1}\right) \\ &= A_0^{-1} + \left(A_0^{-1}E\right)V\begin{pmatrix} H_k & 0 \\ 0 & 0 \end{pmatrix}V^T\left(E^TA_0^{-1}\right), \end{aligned} \tag{3.34}$$

where U and V are defined in (3.24). We write $U = \begin{bmatrix} U_k, \bar{U} \end{bmatrix}$ and $V = \begin{bmatrix} V_k, \bar{V} \end{bmatrix}$, where $\bar{U}$ and $\bar{V}$ individually contain the remaining $s-k$ columns of U and V which are not in U_k and V_k. Recall that $H_k^{-1} = I - U_k^TEV_k$ and define $X = I - \bar{U}^TE\bar{V}$, $Z = -U_k^TE\bar{V}$. Then, from (3.13), we have

$$\begin{aligned} A^{-1} &= A_0^{-1} + \left(A_0^{-1}E\right)V\left(V^TG^{-1}V\right)V^T\left(E^TA_0^{-1}\right) \\ &= A_0^{-1} + \left(A_0^{-1}E\right)V\left(I - U^TEV\right)^{-1}V^T\left(E^TA_0^{-1}\right), \\ &= A_0^{-1} + \left(A_0^{-1}E\right)V\begin{pmatrix} H_k^{-1} & Z \\ Z^T & X \end{pmatrix}^{-1}V^T\left(E^TA_0^{-1}\right). \end{aligned} \tag{3.35}$$

Define the Schur complement,

$$S_k = X - Z^TH_kZ \in \mathbb{R}^{(s-k)\times(s-k)}, \tag{3.36}$$

and the matrix $\bar{S}_k \in \mathbb{R}^{2(s-k)\times 2(s-k)}$,

$$\bar{S}_k = \begin{pmatrix} S_k^{-1} & -I \\ -I & S_k \end{pmatrix}, \tag{3.37}$$

and we will show that S_k is SPD and $\bar{S}_k$ is SPSD.

LEMMA 3.9. *Assume that G defined by* (3.8) *is nonsingular as well as the matrix $I - U_k^TEV_k$. Then, the Schur complement S_k defined by* (3.36) *is SPD. Moreover, matrix $\bar{S}_k$ is SPSD with $s-k$ positive eigenvalues and $s-k$ zero eigenvalues.*

Proof. From Lemma 3.2, we can infer that the eigenvalues of G are all positive. Thus, G is SPD and so is matrix $V^TG^{-1}V$. In the end the Schur complement S_k is SPD when H_k is nonsingular. The signs of the eigenvalues of $\bar{S}_k$ can be easy revealed by a block LDLT factorization. ∎

Then, from (3.34) and (3.35), it follows by subtraction that

$$\begin{aligned} A^{-1} - M^{-1} &= \left(A_0^{-1}E\right) V \left[\begin{pmatrix} H_k^{-1} & Z \\ Z^T & X \end{pmatrix}^{-1} - \begin{pmatrix} H_k & 0 \\ 0 & 0 \end{pmatrix} \right] V^T \left(E^T A_0^{-1}\right), \\ &= \left(A_0^{-1}E\right) V \begin{pmatrix} H_k Z S_k^{-1} Z^T H_k & -H_k Z S_k^{-1} \\ -S_k^{-1} Z^T H_k & S_k^{-1} \end{pmatrix} V^T \left(E^T A_0^{-1}\right), \\ &= \left(A_0^{-1}E\right) V \begin{pmatrix} H_k Z & 0 \\ 0 & S_k^{-1} \end{pmatrix} \bar{S}_k \begin{pmatrix} Z^T H_k & 0 \\ 0 & S_k^{-1} \end{pmatrix} V^T \left(E^T A_0^{-1}\right), \quad (3.38) \end{aligned}$$

and therefore,

$$AM^{-1} = I - A\left(A_0^{-1}E\right) V \begin{pmatrix} H_k Z & 0 \\ 0 & S_k^{-1} \end{pmatrix} \bar{S}_k \begin{pmatrix} Z^T H_k & 0 \\ 0 & S_k^{-1} \end{pmatrix} V^T \left(E^T A_0^{-1}\right). \quad (3.39)$$

Thus, the preconditioned matrix AM^{-1} will have $(n-s+k)$ eigenvalues equal to one and for all the eigenvalues η_i's of AM^{-1}, we have $0 < \eta_i \leq 1$.

4. Implementation. In this section, we will address implementation details for building and applying a DD-LR preconditioner. We focus especially on the implementation in a parallel/distributed environment.

4.1. Building a DD-LR Preconditioner. The construction of a DD-LR preconditioner involves the following steps. In a first step, a graph partitioner is called on the underlying graph of the original global matrix to partition the problem. For each obtained subdomain, we separate the interior nodes and the interface nodes, and then reorder the matrix according to the domain decomposition either into the form of (2.2) or (2.5). The second step is to build a solver for each matrix $B_{i,\alpha} \equiv B_i + \alpha^{-2} E_i E_i^T$ from the *local* system, which can be done in parallel and independently. In our current implementation, this consists of an incomplete Cholesky (IC) or an incomplete LDLT (ILDLT) factorization. The third step is to build a solver for the *global* matrix,

$$C_\alpha \equiv C + \alpha^2 I, \quad (4.1)$$

which will require global communication. We will focus on the solves with C_α in Section 4.3. The last step, which is also the most expensive one, is to compute a low-rank approximation to the matrix $E^T A_0^{-1} E$ in DD-LR-1 or to the matrix $A_0^{-1}E$ in DD-LR-2. The low-rank approximation is obtained by performing the Lanczos procedure [21], which involves the local solves with $B_{i,\alpha}$ and the global solves with C_α. Details on the Lanczos procedure will be given in Section 4.4.

4.2. Applying the DD-LR Preconditioner. First, consider the DD-LR-1 preconditioner (3.14), which we rewrite as

$$M^{-1} = A_0^{-1} \left(I + E G_{k,\theta}^{-1} E^T A_0^{-1} \right). \quad (4.2)$$

The steps involved in applying M^{-1} to a vector x are listed in Algorithm 1. The vector u resulting from the last step, will be the desired vector $u = M^{-1}x$.

As indicated in the comments parts of Algorithm 1, the two solves with A_0 in steps 1 and 5, can in turn be viewed as consisting of p independent local solves with the $B_{i,\alpha}$'s and one global solve with C_α as is inferred from (3.6). Remember that the matrix C given by (2.4) is the matrix that couples all the interface variables,

1 Solve: $A_0 z = x$ /* $B_{i,\alpha}$ solves and C_α solve */
2 Compute: $y = E^T z$ /* Interior variables to interface neighbors */
3 Compute: $w = G_{k,\theta}^{-1} y$ /* Use (3.23) */
4 Compute: $v = Ew$ /* Interface variables to interior neighbors */
5 Solve: $A_0 u = x + v$ /* $B_{i,\alpha}$ solves and C_α solve */

Algorithm 1: Preconditioning operation of the DD-LR-1 preconditioner.

so communication will be required to solve systems with C_α if they are assigned to different processes. The multiplication with E^T in step 2 transforms the z vector, which is a vector of interior variables, into a vector y of interface variables. This can be likened to a 'descent' operation which moves objects from a 'fine' space (fine grid) to a 'coarse' space (coarse grid). The multiplication with E in step 4, performs the reverse operation, which can be termed 'ascent' operation, consisting of going from interface variables to interior variables. Finally, the operation with $G_{k,\theta}^{-1}$ in step 3, involves only the interface variables and yields a vector w of interface variables. This step will also require communication in general. In summary, there are essentially 4 types of operations: (1) solves with the $B_{i,\alpha}$'s; (2) solves with C_α; (3) products with E and E^T, which are dual of one another; and (4) applications of $G_{k,\theta}^{-1}$ to vectors.

1 Solve: $A_0 z = x$ /* $B_{i,\alpha}$ solves and C_α solve */
2 Compute: $u = z + U_k H_k U_k^T x$

Algorithm 2: Preconditioning operation of the DD-LR-2 preconditioner.

Next consider the DD-LR-2 preconditioner given by (3.31). The steps involved in applying the preconditioner, $u = M^{-1}x$, are listed in Algorithm 2, which are simpler than those of applying the DD-LR-1 preconditioner. They consist of one solve with A_0 in step 1 and a low-rank correction in step 2. Communication is required in step 2 as well in the computation of $U_k^T x$, where $U_k \in \mathbb{R}^{n\times k}$, since it involves all the interior and the interface variables. Here, we assume that $H_k \in \mathbb{R}^{k\times k}$ is stored on every process. Parallel implementations of the operations to apply the two types of preconditioners depend on the mappings used for the interface variables, a few of which are discussed in Section 4.5.

4.3. The global solves with $\mathbf{C}_\alpha$. This section addresses the solution methods used for solving systems with the matrix C_α. This is an essential computation required in both the DD-LR-1 and the DD-LR-2 algorithms whenever solving with the matrix A_0. It is an important part of the calculation, especially for the DD-LR-1 preconditioner as it takes place twice for each iteration. In addition, it is a non-local computation which can be costly due to communication. An important characteristic of the matrix C_α under consideration is that it can be made (strongly) diagonally dominant by selecting a proper scaling factor α. Therefore, the first approach one can think about is to use a Chebyshev scheme for solving linear systems with C_α. In our approach, we perform a few steps of the Chebyshev iterations preconditioned by a block Jacobi preconditioner denoted by D_α, where D_α consists of the local C_i's (see, e.g., [6, §2.3.9] for the preconditioned Chebyshev method). An appealing property of the Chebyshev method is that the computation of inner products is avoided. This avoids synchronization among processes, which makes the Chebyshev method efficient

in particular for distributed memory architectures [32]. The price one pays for avoiding communication in the Chebyshev iterations is that this method requires enough knowledge of the spectrum. It is well-known that the convergence of the Chebyshev method is governed by the bounds used for the largest and smallest eigenvalues of C_α. Therefore, prior to the Chebyshev iterations, we perform a few steps of the Lanczos iterations on the matrix $D_\alpha^{-1}C_\alpha$, which can give estimates (not bounds) of the smallest and the largest eigenvalues, namely θ_s and θ_1. In order to have bounds of the spectrum, safeguard terms are added to θ_1 and subtracted from θ_s (see [27, §13.2] for the definitions of these terms).

Another approach that can be used to solve linear systems with C_α is to resort to an approximate inverse, namely $X \approx C_\alpha^{-1}$, and then the solves with C_α will be reduced to the matrix vector products with X. A simple scheme known as the method of Hotelling and Bodewig [17] is given by $X_{k+1} = X_k(2I - C_\alpha X_k)$. In the absence of dropping, this scheme squares the residual $I - C_\alpha X_k$ from one step to the next. This method will converge (quadratically) provided the initial guess is such that $\|I - C_\alpha X_0\| < 1$ for some matrix norm. The global self-preconditioned Minimal Residual (MR) iteration method (in the presence of dropping) has been shown to have superior performance [8]. In this work, we adopt this method to build an approximate inverse of C_α. Given an initial X_0, the self-preconditioned MR iterations are obtained by the sequence of operations in Algorithm 3. The initial X_0 can be selected as the inverse of the diagonal part of C_α. The numerical dropping can be performed by a dual threshold strategy based on a drop tolerance and a maximum number of nonzeros per column.

1 Compute: $R_k = I - C_\alpha X_k$ `/* residual */`
2 Compute: $Z_k = X_k R_k$ `/* self-preconditioned residual */`
3 Apply numerical dropping to Z_k
4 Compute: $\beta_k = \mathrm{tr}(R_k^T C_\alpha Z_k)/\left\|C_\alpha Z_k\right\|_F^2$ `/* tr(·) denotes the trace */`
5 Compute: $X_{k+1} = X_k + \beta_k Z_k$

Algorithm 3: Self-preconditioned global MR iteration with dropping.

4.4. Computation of low-rank approximations. For the DD-LR-1 preconditioner, a rank-k approximation to the matrix $E^T A_0^{-1} E$ is required, which is of the form $U_k \Lambda_k U_k^T$. On the other hand, for the DD-LR-2 preconditioner, we compute a rank-k approximation to the matrix $A_0^{-1}E$ in the form of $U_k V_k^T$. In this section, we consider computing these low-rank approximations by the Lanczos algorithm.

When a few extreme eigenvalues and the associated eigenvectors of a matrix are wanted, the Lanczos algorithm [11, 14, 27] can efficiently approximate these without forming the matrix explicitly since the algorithm only requires the matrix in the form of matrix vector products. For the DD-LR-1 preconditioner, this means that we need to compute $y = E^T A_0^{-1} E x$ for a given x. On the other hand, for the DD-LR-2 preconditioner, we perform the Lanczos algorithm on the matrix $E^T A_0^{-2} E$. In this case, V_k is obtained from the approximated eigenvectors of $E^T A_0^{-2} E$, and U_k can be obtained by computing $U_k = A_0^{-1} E V_k$. Note that it is also possible to compute U_k and V_k at the same time using the Lanczos bidiagonalization method (for details, see, e.g., [14, §9.3.3]).

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, the partial reorthogonalization scheme [28, 35] is used. The cost of this step will not be an issue to the overall performance when a small number of steps are performed to approximate a few eigenpairs. To monitor convergence of the computed eigenvalues, we adopt the approach used in [13]. Let $\theta_j^{(m-1)}$ and $\theta_j^{(m)}$ be the Ritz values obtained in two consecutive Lanczos steps, $m-1$ and m. Assume that we want to approximate k largest eigenvalues and $k < m$. Then with a preselected tolerance ϵ, the desired eigenvalues are considered to have converged if

$$\left|\frac{\sigma_m - \sigma_{m-1}}{\sigma_{m-1}}\right| < \epsilon, \text{ where } \sigma_{m-1} = \sum_{j=1}^{k} \theta_j^{(m-1)} \text{ and } \sigma_m = \sum_{j=1}^{k} \theta_j^{(m)}. \tag{4.3}$$

In terms of the parallel implementation, the most computationally demanding operations involved in the Lanczos procedure include (1) solves with the $B_{i,\alpha}$'s; (2) solves with C_α; (3) products with E and E^T; (4) reorthogonalizations. Communication is required in (2), (3) and (4).

4.5. Parallel implementations: Standard mapping. Considerations of parallel implementations have been mentioned in the previous sections, which suggest several possible schemes for distributing the interface variables. Before discussing these schemes, it is helpful to overview the issues at hand. Operations involved in a DD-LR preconditioner are the following:

a. Solves with $B_{i,\alpha}$, which involve the local interior variables (local),
b. Solves with C_α, which involve all the interface variables (nonlocal),
c. Products with E^T and E, which involve the mapping from the local interface variables to the local interior variables and the reverse operation (local),
d1. For DD-LR-1, applying $G_{k,\theta}^{-1}$ using (3.23), which involves the interface variables (nonlocal), or,
d2. For DD-LR-2, products with U_k and U_k^T, which involve all the variables (nonlocal).

The most straightforward mapping we can consider might be to map each subdomain to a process. This means that the interior variables and the local interface variables of each subdomain are assigned to a different process. If p subdomains are used, the global matrices A and C_α or its approximate inverse X are partitioned and distributed among the p processes. So, process i will hold $d_i + s_i$ rows of A and s_i rows of either C_α or X. In the DD-LR-1 approach, $U_k \in \mathbb{R}^{s\times k}$ is distributed such that process i will keep s_i rows, while in the DD-LR-2 approach, $d_i + s_i$ rows of $U_k \in \mathbb{R}^{n\times k}$ will reside in process i. For all the nonlocal operations, communication is among all the p processes. The operations labeled by (b.) and (d1.) involve interface to interface communication, while the operation (d2.) involves communication among all the variables. From another perspective, the communication in (d1.) and (d2.) is of the all-reduction type required by the products with U_k^T, while (b.) involves point-to-point communication such as that in the rowwise distributed sparse matrix vector products. If an iterative process is used to solve systems with C_α, then there will be communication involved at each iteration. This is why it is important to select α carefully so as to reach a compromise between the number of inner iterations (each of which requires communication) and outer iterations (each of which involves solves with C_α). The scalar α also plays a role if the approximate inverse is used since the convergence of the MR iterations will be affected.

4.6. Unbalanced mapping: Interface variables together. Since the interface nodes require communication when solving systems with C_α and when applying

$G_{k,\theta}$ to a vector, an idea that comes to mind is to map the interior variables of a given subdomain to a process, and all the interface variables to a separate process. The global matrix A is partitioned and reordered to the form of (2.5). In a case of p subdomains, $p+1$ processes will be used and the global matrix A is distributed in such a way that process $i, i = 1, \cdots, p$ owns the d_i rows (which consist of $B_{i,\alpha}$ and E_i) corresponding to the local interior variables, while process $p+1$ will hold the s rows related to all the interface variables. Moreover, in this mapping the matrix C_α or X will reside entirely on the process $p+1$.

A clear advantage of this approach is that solving the systems with C_α requires no communication. However, the operations with E and E^T are no longer local. Indeed, E^T can be viewed as a restriction operator, which "scatters" interface data from process $p+1$ to the other p processes. Specifically, referring to (2.2) and the related Figure 2.1, the variables y_i will each be sent to process i from process $p+1$. Analogously, the product with E, as a prolongation, will perform a dual operation which "gathers" from processes $1, \cdots, p$ to process $p+1$. In Algorithm 1, the scatter operation goes before step 2 and the gather operation should be executed after step 4. Likewise, if we store the entire U_k on process $p+1$, applying $G_{k,\theta}$ will not require communication but another pair of the gather-and-scatter operations will be needed before and after step 3 respectively. Therefore, at each application of the DD-LR-1 preconditioner, *two* pairs of the scatter-and-gather operations on the vectors of the interface variables will be required. A middle ground approach is to distribute U_k in the same way as it is in the standard mapping. In this way, applying $G_{k,\theta}$ will require communication among the p processes but only *one* pair of the scatter-and-gather operations is necessary. On the other hand, in the DD-LR-2 approach, the distribution of U_k should be consistent with that of the matrix A.

The main issue with this approach is that it is hard to achieve load balancing in general. Indeed for a good balancing, we need to have the interior variables of each subdomain and the global interface variables of roughly the same size and this is difficult to achieve in practice. Furthermore, the load balancing issue is further complicated by the fact that the equations to solve on process $p+1$ are completely different from those on the other processes. A remedy to this is to use q processes dedicated to the global interface (a total of $p+q$ processes used in all) instead of just one, which will provide a compromise. Then, the communication for the solves with C_α and applying $G_{k,\theta}$ is confined within the q processes.

4.7. Improving a given preconditioner. One of the main weaknesses of standard, e.g., ILU-type, preconditioners is that they are difficult to update. For example, suppose we compute a preconditioner to a given matrix and find that it is not accurate enough to yield convergence. In the case of ILU we would have essentially to start from the beginning. For DD-LR, improving a given preconditioner is essentially trivial. For example, the heart of DD-LR-1 consists of obtaining a low-rank approximation the matrix G defined in (3.7). Improving this approximation would consist in merely adding a few more vectors (increasing k) and this can be easily achieved in a number of ways without having to throw away the vectors already computed.

5. Numerical Experiments. The experiments were conducted on Itasca, an HP ProLiant BL280c G6 Linux cluster at Minnesota Supercomputing Institute, which has 2,186 Intel Xeon X5560 processors. Each processor has four cores, 8 MB cache, and communicates with memory on a QuickPath Interconnect (QPI) interface. A preliminary implementation of the DD-LR preconditioners has been written in C/C++

with the Intel Math Kernel Library, the Intel MPI library and PETSc [3, 4, 5], compiled by the Intel MPI compiler using -O3 optimization level.

The accelerators used are the conjugate gradient (CG) method when both the matrix and the preconditioner are SPD, and the generalized minimal residual (GMRES) method with a restart dimension of 40, denoted by GMRES(40), for the indefinite cases. Three types of preconditioners were compared in our experiments: 1) the DD-LR preconditioners, 2) the pARMS method and 3) the restricted additive Schwarz (RAS) preconditioner (with overlapping). Recall that for an SPD matrix A, the DD-LR preconditioners given by (3.14) and (3.31) will also be SPD if the assumptions in Proposition 3.5 and 3.8 are satisfied. However, Proposition 3.5 and 3.8 will not hold when the solves with A_0 are approximate, which is typical in practice. The positive definiteness can be determined by checking if the largest eigenvalue, λ_1, is less than one for DD-LR-1 or by checking the positive definiteness of H_k for DD-LR-2. In the DD-LR-1 method, we set $\theta = \lambda_{k+1}$.

For each subdomain i, we reorder $B_{i,\alpha}$ by the approximate minimum degree ordering (AMD) [1, 2, 9] to reduce fill-ins and then simply use IC or ILDLT as a local solver, which is not necessarily the best choice. A more efficient and robust local solver, for example, the ARMS approach in [34], can lead to better performance in terms of both memory requirement and speed. However, this has not been implemented in our current code. A typical setting of the scalar α for C_α and $B_{i,\alpha}$ is $\alpha = 1$, which in general gives the best overall performance, the exceptions being the three cases in Section 5.2, for which choosing $\alpha > 1$ will help with the convergence. Regarding solves with C_α, using the approximate inverse is usually more efficient than performing Chebyshev iterations, especially in the iteration phase. The price to pay here is that the approximate inverse might be expensive to build, in particular for the indefinite 3-D cases. The standard mapping will be adopted unless specially stated, which in general gives a better performance than the unbalanced mapping, the behavior of which will be analyzed by the results in Table 5.3. In the Lanczos algorithm, the convergence is checked every 10 iterations and the tolerance ϵ used for convergence in (4.3) is set to 10^{-4}. In addition, we set the maximum number of Lanczos steps as five times the number of requested eigenvalues.

For the pARMS method, we select the ARMS method to be the local preconditioner and use the Schur complement method as the global preconditioner, where the global Schur complement systems are solved by a few inner Krylov subspace iterations preconditioned by a block-Jacobi preconditioner. For the details of these options of pARMS, we refer the readers to [33, 34]. We point out that when inner iterations are enabled in pARMS, some flexible Krylov subspace method will be required since the preconditioning will no longer be fixed from one step to the next. We always use pARMS in conjunction with the flexible version of GMRES [31]. The RAS preconditioner is obtained from PETSc, for which we set the incomplete LU factorization with level-based dropping, namely ILU(k), as the local solver and uses a one level overlapping between each pair of subdomains. Moreover, since the RAS preconditioner is nonsymmetric even for a symmetric matrix, GMRES will be used as well.

Based on the experimental results, we can state that in general, building a DD-LR preconditioner requires much more time than a pARMS preconditioner or an RAS preconditioner requiring similar storage. Nevertheless, experimental results indicate that the DD-LR-1 preconditioner is more robust and can achieve great time savings in the iterative process. In particular, in the cases of systems with a large number of right-hand sides, expensive but effective preconditioners may be justified because their

cost is amortized. In this section, we first report on results of solving symmetric linear systems from a 2-D and a 3-D elliptic PDE on regular meshes. Next, we will show the results for solving a sequence of general sparse symmetric linear systems. For all the problems, a parallel multilevel k-way graph partitioning algorithm from ParMetis [18, 19] was used for the domain decomposition. For all the cases, iterations were stopped whenever the residual norm has been reduced by 6 orders of magnitude or the maximum number of iterations allowed, which is 500, was exceeded. The results are summarized in Tables 5.1, 5.2 and 5.5, where all times are reported in seconds. When comparing the 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 non-convergence within the maximum allowed number of steps.

5.1. Model Problems. We will examine a 2-D and a 3-D elliptic PDE,

$$\begin{aligned} -\Delta u - cu &= f \ \text{ in } \Omega, \\ u &= 0 \text{ on } \partial\Omega, \end{aligned} \tag{5.1}$$

where $\Omega = (0,1) \times (0,1)$ and $\Omega = (0,1) \times (0,1) \times (0,1)$ are the domains, and $\partial\Omega$ is the boundary. We take the 5-point(or 7-point) centered difference approximation on the regular meshes. To begin with, we solve (5.1) with $c = 0$, so that the coefficient matrices are SPD and we use the DD-LR preconditioners along with the CG method. Numerical experiments were carried out to compare the performance of the DD-LR preconditioners with the pARMS method and the RAS preconditioner. The results are shown in Table 5.1. The sizes of the meshes, the orders of the matrices (N), the fill-ratios (fill), the numbers of iterations (its), the time for building the preconditioners (p-t) and the time for iterations (i-t) are tabulated. As shown, we tested problems on six 2-D meshes and six 3-D meshes of increasing sizes, where we keep roughly the same problem size on each process such that the number of processes used is growing proportional to the order of the matrix. This can serve as a *weak scaling* test. We increase the rank k used in the DD-LR preconditioner as the meshes sizes. The fill-ratios of the DD-LR-1 preconditioner and those of the pARMS method were controlled to be roughly equal, whereas the memory cost of the DD-LR-2 method is much higher, which comes mostly from the matrix $U_k \in \mathbb{R}^{n \times k}$ when k is large. For pARMS, the inner Krylov subspace dimension used is 3.

The time required to build a DD-LR preconditioner is much higher and it grows with the rank k. In contrast the time to build the pARMS and the RAS preconditioners remains roughly constant. This set-up time for DD-LR is typically dominated by the Lanczos algorithm, where solves with $B_{i,\alpha}$ and C_α are required at each iteration. Moreover, when k is large, the cost of reorthogonalization will become significant. A more efficient local solver for $B_{i,\alpha}$ and fine-grain parallelism (threading, or GPU computing) may help reduce the time of building the preconditioner, but in general building a DD-LR preconditioner is an expensive process. As shown in Table 5.1, the DD-LR-1 method and the pARMS method are more robust as they succeeded for all the 2-D and 3-D cases, while the DD-LR-2 method failed for the largest 2-D case and the RAS method failed for the three largest 2-D cases. For most of the 2-D problems, the DD-LR-1/CG method achieved convergence in fewer iterations than those required by the other three methods, and required less time for iterations. For the 3-D problems, DD-LR-1 required more iterations than pARMS and RAS to converge but a performance gain was still achieved for most cases in terms of reduced iteration

time. Exceptions are the two largest 3-D problems, where the RAS/GMRES method yielded the smallest iteration time.

TABLE 5.1

Comparison between the DD-LR *preconditioners, the* pARMS *method and the* RAS *preconditioner for solving SPD linear systems from the 2-D/3-D elliptic PDEs in* (5.1) *along with the* CG *method or the* GMRES(40) *method.*

Mesh	Np	DD-LR-1					DD-LR-2				
		rk	nz	its	p-t	i-t	rk	nz	its	p-t	i-t
128^2	2	8	6.6	15	.209	.027	8	8.2	30	.213	.031
256^2	8	16	6.6	34	.325	.064	16	9.7	69	.330	.083
512^2	32	32	6.8	61	.567	.122	32	13.0	132	.540	.194
1024^2	128	64	7.0	103	1.12	.218	64	19.3	269	1.03	.570
1448^2	256	91	7.2	120	1.67	.269	91	24.7	385	1.72	1.05
2048^2	512	128	7.6	168	3.02	.410	128	32.2	F	-	-
25^3	2	8	7.2	11	.309	.025	8	8.3	17	.355	.021
50^3	16	16	7.5	27	.939	.064	16	9.3	52	.958	.076
64^3	32	16	7.4	36	1.06	.089	16	9.2	67	1.07	.102
100^3	128	32	8.0	52	1.57	.136	32	11.5	101	1.48	.190
126^3	256	32	8.2	65	2.07	.178	32	12.5	126	1.87	.265
159^3	512	51	8.7	85	2.92	.251	51	14.2	156	2.50	.387

Mesh	Np	pARMS				RAS			
		nz	its	p-t	i-t	nz	its	p-t	i-t
128^2	2	6.7	15	.062	.037	2.7	40	.003	.032
256^2	8	6.7	30	.066	.082	2.7	102	.004	.072
512^2	32	6.9	52	.072	.194	2.7	212	.005	.157
1024^2	128	6.6	104	.100	.359	2.7	F	.008	-
1448^2	256	6.6	247	.073	.820	2.7	F	.011	-
2048^2	512	6.8	282	.080	1.06	2.7	F	.015	-
25^3	2	7.3	9	.100	.032	5.9	13	.004	.041
50^3	16	8.1	17	.179	.095	6.7	28	.006	.071
64^3	32	8.2	20	.142	.121	6.7	34	.007	.103
100^3	128	8.3	29	.170	.198	6.7	51	.011	.148
126^3	256	8.4	34	.166	.216	6.7	60	.014	.127
159^3	512	8.5	40	.179	.275	6.7	83	.019	.183

Next, we consider solving symmetric indefinite problems by setting $c > 0$ in (5.1), which corresponds to shifting the discretized negative Laplacian (a positive definite matrix) by subtracting σI with a certain $\sigma > 0$. In this set of experiments, we reduce the size of the shift, σ as the problem size increases in order to make the problems fairly difficult but not too difficult to solve for all the methods. We used a higher inner Krylov subspace dimension, which is 6, in the pARMS method. Results are reported in Table 5.2. From there we can first see that the DD-LR-2 method did not perform well as it failed for almost all the problems. Second, the RAS method failed for all the 2-D cases and three of the 3-D cases, but for the other three cases where it worked, it yielded the best iteration time. Third, the DD-LR-1 method achieved convergence in all the cases while the pARMS method failed for two 2-D cases. Comparison between DD-LR-1 and pARMS shows a similar result as in Table 5.1: for the 2-D cases, DD-LR-1 required smaller numbers of iteration and less iteration time, while for some 3-D

cases, it required a bit more iterations but still less iteration time.

TABLE 5.2

Comparison between the DD-LR *preconditioners, the* pARMS *method and the* RAS *preconditioner for solving symmetric indefinite linear systems from the 2-D/3-D elliptic PDEs in* (5.1) *along with the* GMRES(40) *method*

Mesh	Np	σ	DD-LR-1					DD-LR-2				
			rk	nz	its	p-t	i-t	rk	nz	its	p-t	i-t
128^2	2	1e-1	16	6.8	18	.233	.034	16	13.2	146	.310	.234
256^2	8	1e-2	32	6.8	38	.674	.080	16	13.0	F	1.01	-
512^2	32	1e-3	64	7.1	48	1.58	.105	64	19.4	F	1.32	-
1024^2	128	2e-4	128	7.6	68	4.15	.160	128	32.3	F	4.45	-
1448^2	256	5e-5	182	8.1	100	7.14	.253	182	43.2	F	7.77	-
2048^2	512	2e-5	256	8.8	274	12.6	.749	256	58.4	F	13.1	-
25^3	2	.25	16	8.3	29	.496	.099	16	9.6	62	.595	.130
50^3	16	7e-2	32	8.2	392	1.38	1.19	32	10.2	F	1.66	-
64^3	32	3e-2	64	8.9	201	2.26	.688	64	16.3	F	2.08	-
100^3	128	2e-2	128	11.4	279	5.17	1.08	128	28.7	F	5.29	-
126^3	256	7e-3	128	12.8	255	5.85	1.10	128	28.3	F	6.01	-
159^3	512	5e-3	160	13.5	387	8.60	1.71	160	33.0	F	8.33	-

Mesh	Np	σ	pARMS				RAS			
			nz	its	p-t	i-t	nz	its	p-t	i-t
128^2	2	1e-1	11.4	76	.114	.328	2.7	F	.003	-
256^2	8	1e-2	13.9	F	-	-	2.7	F	.004	-
512^2	32	1e-3	12.3	298	.181	1.53	2.7	F	.005	-
1024^2	128	2e-4	12.5	232	.230	1.46	2.7	F	.008	-
1448^2	256	5e-5	12.5	F	-	-	2.7	F	.011	-
2048^2	512	2e-5	12.6	314	.195	2.13	2.7	F	.015	-
25^3	2	.25	8.3	100	.156	.599	5.9	108	.004	.123
50^3	16	7e-2	8.9	448	.142	2.59	6.7	F	.006	-
64^3	32	3e-2	8.9	130	.115	.784	6.7	252	.007	.375
100^3	128	2e-2	9.4	187	.137	1.24	6.7	343	.011	.541
126^3	256	7e-3	10.6	340	.137	2.74	6.7	F	.014	-
159^3	512	5e-3	10.8	329	.148	2.85	6.7	F	.019	-

In all the previous tests, the DD-LR-1 preconditioner was used with the standard mapping. In the next set of experiments, we will examine the behavior of the DD-LR-1 method with the unbalanced mapping discussed in Section 4.6. In these experiments, we tested a 2-D 128×128 mesh and a 3-D $25 \times 25 \times 25$ mesh, which were divided into 128 subdomains ($p = 128$). Note here that the problem size per process is remarkably small. This was made on purpose since it will make the communication cost more significant (and likely to be dominant) in the overall cost in solving with C_α, such that it can make the effect of the unbalanced mapping more prominent. Table 5.3 lists the times for iterations for solving the two problems with the standard mapping and the unbalanced mapping with different settings. Two types of solution methods for the systems with C_α were tested, the one with the approximate inverse and the preconditioned Chebyshev iterations (5 iterations were used per solve). In the unbalanced mapping, q processes were used dedicated to the interface variables (and $p + q$ processes were used totally). The unbalanced mapping was tested with eight different q's from 1 to 96 and $q = 1$ is the special case where no communication is involved in the solve with C_α. In the DD-LR-1 preconditioner, U_k is stored on the p processes, such that only one pair of the scatter-and-gather communication is

required at each outer iteration as discussed in Section 4.6. The standard mapping is indicated by $q = 0$ in Table 5.3. As the results indicated, the iteration time keeps decreasing at first as the number of the interface processes q increases but after some point it will start to increase, which is a typical situation corresponding to the balance of communication and computation. When q is small, the amount of computation on each of the q processes is high, which dominates the overall cost, so the cost keeps being reduced as q increases until some point when the communication cost starts to affect the overall performance. The optimal number of the interface processes yields a balance of computation and communication, which is shown in bold in Table 5.3. For these cases, the optimal iteration time with the unbalanced mapping is slightly better than the iteration time with the standard mapping. However, we need to point out that this is not a typical case in practice and for all the other tests in this section, we adopt the standard mapping in the DD-LR-1 preconditioner.

TABLE 5.3

Comparison of the iteration time (measured in milliseconds) between the standard mapping and the unbalanced mapping in solving 2-D/3-D SPD problems in (5.1) *by the DD-LR-1 preconditioner along with the CG method. q is the number of processes dedicated to the interface variables in the unbalanced mapping. $q = 0$ indicates the standard mapping.*

Mesh	C_α^{-1}	$q = 0$	1	2	4	8	16	32	64	96
128^2	`AINV`	9.0	53.4	27.8	15.4	13.9	10.8	**9.1**	9.4	13.5
	`Cheb`	9.2	116.8	60.7	29.9	15.8	10.7	10.0	**8.3**	9.1
25^3	`AINV`	15.1	119.7	66.3	34.7	26.0	19.2	17.2	**14.8**	18.2
	`Cheb`	13.8	368.3	166.0	78.3	38.5	20.4	14.4	**12.3**	13.5

5.2. General Matrices. We selected 12 symmetric matrices from the University of Florida sparse matrix collection [10] for the following tests. Table 5.4 lists the name, order (N), number of nonzeros (NNZ), 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 a random vector.

TABLE 5.4

Names, orders (N), numbers of nonzeros (NNZ) and short descriptions of the test matrices from the University of Florida sparse matrix collection [10].

MATRIX	N	NNZ	DESCRIPTION
Andrews/Andrews	60,000	760,154	computer graphics problem
UTEP/Dubcova2	65,025	1,030,225	2-D/3-D PDE problem
Rothberg/cfd1	70,656	1,825,580	CFD problem
Schmid/thermal1	82,654	574,458	thermal problem
Rothberg/cfd2	123,440	3,085,406	CFD problem
UTEP/Dubcova3	146,689	3,636,643	2-D/3-D PDE problem
Botonakis/thermo_TK	204,316	1,423,116	thermal problem
Wissgott/para_fem	525,825	3,674,625	CFD problem
CEMW/tmt_sym	726,713	5,080,961	electromagnetics problem
McRae/ecology2	999,999	4,995,991	landscape ecology problem
McRae/ecology1	1,000,000	4,996,000	landscape ecology problem
Schmid/thermal2	1,228,045	8,580,313	thermal problem

Table 5.5 shows the performance of the DD-LR preconditioners along with the

rank and memory cost for each problem. The DD-LR-1 and the DD-LR-2 preconditioners were combined with GMRES(40) for three problems `tmt_sym`, `ecology1` and `ecology2`, where the obtained DD-LR preconditioners were found not to be SPD, while for the other problems the CG method was applied. We set the scalar $\alpha = 2$ in C_α and $B_{i,\alpha}$ for two problems `ecology1` and `ecology2`, where it turned out to reduce the numbers of iterations, but for elsewhere we set $\alpha = 1$. As indicated by the results, the DD-LR-1 preconditioner achieved convergence for all the cases, whereas the other three preconditioners all have some failures for a few cases. Similar to the experiments with the model problems, the DD-LR preconditioners required more time to construct, but the DD-LR-1 method achieved time savings in the iteration phase for seven (out of twelve) problems, whereas the DD-LR-2 method had a faster convergence for three cases, compared to their counterparts the pARMS and the RAS preconditioners.

TABLE 5.5
Comparison among the DD-LR*, the* pARMS *and the* RAS *preconditioners for solving general sparse symmetric linear systems along with the* CG *or the* GMRES(40) *method.*

Mesh	Np	DD-LR-1					DD-LR-2				
		rk	nz	its	p-t	i-t	rk	nz	its	p-t	i-t
Andrews	8	8	4.7	33	.587	.220	8	5.2	53	.824	.175
Dubcova2	8	16	3.5	18	.850	.054	16	4.5	44	.856	.079
cfd1	8	8	18.1	17	7.14	.446	8	18.4	217	6.44	2.97
thermal1	8	16	6.0	48	.493	.145	16	8.3	126	.503	.234
cfd2	16	8	13.2	12	4.93	.232	8	13.4	F	5.11	-
Dubcova3	16	16	2.6	16	1.70	.061	16	3.2	44	1.71	.107
thermo_TK	16	32	6.4	24	.568	.050	32	10.8	63	.537	.096
para_fem	16	32	7.8	59	4.02	.777	32	12.3	159	4.12	1.35
tmt_sym	16	16	7.3	33	5.56	.668	16	9.5	62	5.69	.790
ecology2	32	32	8.9	39	3.67	.433	32	15.2	89	3.79	.709
ecology1	32	32	8.8	40	3.48	.423	32	15.1	82	3.59	.656
thermal2	32	32	6.8	140	5.06	2.02	32	11.3	F	5.11	-

Mesh	Np	pARMS				RAS			
		nz	its	p-t	i-t	nz	its	p-t	i-t
Andrews	8	4.3	15	.217	.109	3.6	19	.010	.073
Dubcova2	8	3.5	25	.083	.090	3.5	43	.008	0.11
cfd1	8	16.1	F	.091	-	10.6	153	.013	3.55
thermal1	8	5.4	39	.089	.153	4.6	156	.006	.235
cfd2	16	26.0	F	.120	-	11.9	310	.012	3.26
Dubcova3	16	2.6	37	.130	.200	4.2	39	.013	.212
thermo_TK	16	4.9	16	.048	.035	5.5	34	.004	.067
para_fem	16	6.5	89	.586	1.36	5.1	247	.019	1.18
tmt_sym	16	6.9	16	.587	.361	3.7	26	.026	.222
ecology2	32	9.9	15	.662	.230	5.8	28	.017	.165
ecology1	32	10.0	14	.664	.220	5.8	27	.017	.161
thermal2	32	6.1	205	.547	3.70	4.7	F	.025	-

6. Conclusion. This paper presented a preconditioning method for solving distributed symmetric sparse linear systems, based on an approximate inverse of the

original matrix which exploits the domain decomposition method and low-rank approximations. Two low-rank approximation strategies are discussed, called DD-LR-1 and DD-LR-2. In terms of the number of iterations and iteration time, experimental results indicate that for SPD systems, the DD-LR-1 preconditioner can be an efficient alternative to other Domain Decomposition-type approaches such as one based on distributed Schur complements (as in pARMS), or on the RAS preconditioner. Moreover, this preconditioner appears to be more robust than the pARMS method and the RAS method for indefinite problems.

The DD-LR preconditioners require more time to build than other DD-based preconditioners. However, one must take a number of other factors into account. First, some improvements can be made to reduce the set-up time. For example, more efficient local solvers such as ARMS can be used instead of the current ILUs; vector processing such as GPU computing can accelerate the computations of the low-rank corrections; and more efficient algorithms than the Lanczos method, e.g., randomized techniques [15], can be exploited for computing the eigenpairs. Second, there are many applications in which many systems with the same matrix must be solved. In this case more expensive but more effective preconditioners may be justified because their cost can be amortized. Finally, another important factor touched upon briefly in Section 4.7 is that the preconditioners discussed here are more easily updatable than traditional ILU-type or DD-type preconditioners.

Acknowledgements. The authors are grateful for resources from the University of Minnesota Supercomputing Institute and for assistance with the computations. The authors would like to thank the PETSc team for their help with the implementation.

REFERENCES

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

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

[3] S. Balay, S. Abhyankar, M. F. Adams, J. Brown, P. Brune, K. Buschelman, V. Eijkhout, W. D. Gropp, D. Kaushik, M. G. Knepley, L. C. McInnes, K. Rupp, B. F. Smith, and H. Zhang, *PETSc users manual*, Tech. Report ANL-95/11 - Revision 3.5, Argonne National Laboratory, 2014.

[4] ———, *PETSc Web page*. `http://www.mcs.anl.gov/petsc`, 2014.

[5] S. Balay, W. D. Gropp, L. C. McInnes, and B. F. Smith, *Efficient management of parallelism in object oriented numerical software libraries*, in Modern Software Tools in Scientific Computing, E. Arge, A. M. Bruaset, and H. P. Langtangen, eds., Birkhäuser Press, 1997, pp. 163–202.

[6] R. Barrett, M. Berry, T. F. Chan, J. Demmel, J. Donato, J. Dongarra, V. Eijkhout, R. Pozo, C. Romine, and H. Van der Vorst, *Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods, 2nd Edition*, SIAM, Philadelphia, PA, 1994.

[7] Ü. V. Çatalyurek and C. Aykanat, *Hypergraph-partitioning-based decomposition for parallel sparse-matrix vector multiplication*, IEEE Transactions on Parallel and Distributed Systems, 10 (1999), pp. 673–693.

[8] E. Chow and Y. Saad, *Approximate inverse preconditioners via sparse-sparse iterations*, SIAM Journal on Scientific Computing, 19 (1998), pp. 995–1023.

[9] T. A. Davis, *Direct Methods for Sparse Linear Systems (Fundamentals of Algorithms 2)*, Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2006.

[10] T. A. Davis and Y. Hu, *The University of Florida Sparse Matrix Collection*, ACM Trans. Math. Softw., 38 (2011), pp. 1:1–1:25.

[11] J. Demmel, J. Dongarra, A. Ruhe, and H. van der Vorst, *Templates for the solution of algebraic eigenvalue problems: a practical guide*, Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2000.

[12] B. Engquist and L. Ying, *Sweeping preconditioner for the helmholtz equation: Hierarchical matrix representation*, Communications on Pure and Applied Mathematics, 64 (2011), pp. 697–735.
[13] H. Fang and Y. Saad, *A filtered Lanczos procedure for extreme and interior eigenvalue problems*, SIAM Journal on Scientific Computing, 34 (2012), pp. A2220–A2246.
[14] G. H. Golub and C. F. Van Loan, *Matrix computations (3rd ed.)*, Johns Hopkins University Press, Baltimore, MD, USA, 1996.
[15] N. Halko, P. Martinsson, and J. Tropp, *Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions*, SIAM Review, 53 (2011), pp. 217–288.
[16] B. Hendrickson and R. Leland, *The Chaco User's Guide Version 2*, Sandia National Laboratories, Albuquerque NM, 1994.
[17] A. S. Householder, *Theory of Matrices in Numerical Analysis*, Blaisdell Pub. Co., Johnson, CO, 1964.
[18] G. Karypis and V. Kumar, *A fast and high quality multilevel scheme for partitioning irregular graphs*, SIAM Journal on Scientific Computing, 20 (1998), pp. 359–392.
[19] G. Karypis and V. Kumar, *A parallel algorithm for multilevel graph partitioning and sparse matrix ordering*, Journal of Parallel and Distributed Computing, 48 (1998), pp. 71 – 95.
[20] T. G. Kolda, *Partitioning sparse rectangular matrices for parallel processing*, Lecture Notes in Computer Science, 1457 (1998), pp. 68–79.
[21] C. Lanczos, *An iteration method for the solution of the eigenvalue problem of linear differential and integral operators*, Journal of Research of the National Bureau of Standards, 45 (1950), pp. 255–282.
[22] S. Le Borne, *$\mathcal{H}$-matrices for convection-diffusion problems with constant convection*, Computing, 70 (2003), pp. 261–274.
[23] S. Le Borne and L. Grasedyck, *$\mathcal{H}$-matrix preconditioners in convection-dominated problems*, SIAM Journal on Matrix Analysis and Applications, 27 (2006), pp. 1172–1183.
[24] R. Li and Y. Saad, *Divide and conquer low-rank preconditioners for symmetric matrices*, SIAM Journal on Scientific Computing, 35 (2013), pp. A2069–A2095.
[25] ———, *GPU-accelerated preconditioned iterative linear solvers*, The Journal of Supercomputing, 63 (2013), pp. 443–466.
[26] Z. Li, Y. Saad, and M. Sosonkina, *pARMS: a parallel version of the algebraic recursive multilevel solver*, Numerical Linear Algebra with Applications, 10 (2003), pp. 485–509.
[27] B. N. Parlett, *The Symmetric Eigenvalue Problem*, Society for Industrial and Applied Mathematics, Philadephia, PA, 1998.
[28] B. N. Parlett and D. S. Scott, *The Lanczos algorithm with selective orthogonalization*, Mathematics of Computation, 33 (1979), pp. pp. 217–238.
[29] F. Pellegrini, Scotch *and* libScotch *5.1 User's Guide*, INRIA Bordeaux Sud-Ouest, IPB & LaBRI, UMR CNRS 5800, 2010.
[30] A. Pothen, H. D. Simon, and K. P. Liou, *Partitioning sparse matrices with eigenvectors of graphs*, SIAM Journal on Matrix Analysis and Applications, 11 (1990), pp. 430–452.
[31] Y. Saad, *A flexible inner-outer preconditioned gmres algorithm*, SIAM Journal on Scientific Computing, 14 (1993), pp. 461–469.
[32] Y. Saad, *Iterative Methods for Sparse Linear Systems, 2nd edition*, SIAM, Philadelpha, PA, 2003.
[33] Y. Saad and M. Sosonkina, *pARMS: A package for solving general sparse linear systems on parallel computers*, in Parallel Processing and Applied Mathematics, Roman Wyrzykowski, Jack Dongarra, Marcin Paprzycki, and Jerzy Waniewski, eds., vol. 2328 of Lecture Notes in Computer Science, Springer Berlin Heidelberg, 2002, pp. 446–457.
[34] Y. Saad and B. Suchomel, *ARMS: An algebraic recursive multilevel solver for general sparse linear systems*, Numerical Linear Algebra with Applications, 9 (2002).
[35] H. D. Simon, *The Lanczos algorithm with partial reorthogonalization*, Mathematics of Computation, 42 (1984), pp. pp. 115–142.
[36] J. Xia, S. Chandrasekaran, M. Gu, and X. S. Li, *Fast algorithms for hierarchically semiseparable matrices*, Numerical Linear Algebra with Applications, 17 (2010), pp. 953–976.
[37] 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.