A robust algebraic domain decomposition preconditioner for sparse normal equations

Article

Accepted Version

A ROBUST ALGEBRAIC DOMAIN DECOMPOSITION PRECONDITIONER FOR SPARSE NORMAL EQUATIONS*

HUSSAM AL DAAS†, PIERRE JOLIVET‡, AND JENNIFER A. SCOTT†§

Abstract. Solving the normal equations corresponding to large sparse linear least-squares problems is an important and challenging problem. For very large problems, an iterative solver is needed and, in general, a preconditioner is required to achieve good convergence. In recent years, a number of preconditioners have been proposed. These are largely serial and reported results demonstrate that none of the commonly used preconditioners for the normal equations matrix is capable of solving all sparse least-squares problems. Our interest is thus in designing new preconditioners for the normal equations that are efficient, robust, and can be implemented in parallel. Our proposed preconditioners can be constructed efficiently and algebraically without any knowledge of the problem and without any assumption on the least-squares matrix except that it is sparse. We exploit the structure of the symmetric positive definite normal equations matrix and use the concept of algebraic local symmetric positive semi-definite splittings to introduce two-level Schwarz preconditioners for least-squares problems. The condition number of the preconditioned normal equations is shown to be theoretically bounded independently of the number of subdomains in the splitting. This upper bound can be adjusted using a single parameter τ that the user can specify. We discuss how the new preconditioners can be implemented on top of the PETSc library using only 150 lines of Fortran, C, or Python code. Problems arising from practical applications are used to compare the performance of the proposed new preconditioner with that of other preconditioners.

Key words. Algebraic domain decomposition, two-level preconditioner, additive Schwarz, normal equations, sparse linear least-squares.

1. Introduction.

We are interested in solving large-scale sparse linear least-squares (LS) problems

$$\min_x \|Ax - b\|_2, \tag{1.1}$$

where $A \in \mathbb{R}^{m\times n}$ $(m \geq n)$ and $b \in \mathbb{R}^m$ are given. Solving (1.1) is mathematically equivalent to solving the $n \times n$ normal equations

$$Cx = A^\top b, \qquad C = A^\top A, \tag{1.2}$$

where, provided A has full column rank, the normal equations matrix C is symmetric and positive definite (SPD). Two main classes of methods may be used to solve the normal equations: direct methods and iterative methods. A direct method proceeds by computing an explicit factorization, either using a sparse Cholesky factorization of C or a "thin" QR factorization of A. While well-engineered direct solvers [2, 12, 33] are highly robust, iterative methods may be preferred because they generally require significantly less storage (allowing them to tackle very large problems for which the memory requirements of a direct solver are prohibitive) and, in some applications, it may not be necessary to solve the system with the high accuracy offered by a direct solver. However, the successful application of an iterative method usually requires a suitable preconditioner to achieve acceptable (and ideally, fast) convergence

*Submitted to the editors January 26, 2022.

†STFC Rutherford Appleton Laboratory, Harwell Campus, Didcot, Oxfordshire, OX11 0QX, UK (hussam.al-daas@stfc.ac.uk, jennifer.scott@stfc.ac.uk).

‡CNRS, ENSEEIHT, 2 rue Charles Camichel, 31071 Toulouse Cedex 7, France (pierre.jolivet@enseeiht.fr).

§School of Mathematical, Physical and Computational Sciences, University of Reading, Reading RG6 6AQ, UK.

rates. Currently, there is much less knowledge of preconditioners for LS problems than there is for sparse symmetric linear systems and, as observed in Bru et al. [8], "the problem of robust and efficient iterative solution of LS problems is much harder than the iterative solution of systems of linear equations." This is, at least in part, because A does not have the properties of differential problems that can make standard preconditioners effective for solving many classes of linear systems.

Compared with other classes of linear systems, the development of preconditioners for sparse LS problems may be regarded as still being in its infancy. Approaches include

- variants of block Jacobi (also known as block Cimmino) and SOR [19];
- incomplete factorizations such as incomplete Cholesky, QR, and LU factorizations, for example, [8, 30, 38, 39];
- sparse approximate inverses [11].

A review and performance comparison is given in [22]. This found that, whilst none of the approaches successfully solved all LS problems, limited memory incomplete Cholesky factorization preconditioners appear to be the most reliable. The incomplete factorization-based preconditioners are designed for moderate size problems because current approaches, in general, are not suitable for parallel computers. The block Cimmino method can be parallelized easily, however, it lacks robustness as the iteration count to reach convergence cannot be controlled and typically increases significantly when the number of blocks increases for a fixed problem [17]. Several techniques have been proposed to improve the convergence of block Cimmino but they still lack robustness [18]. Thus, we are motivated to design a new class of LS preconditioners that are not only reliable but can also be implemented in parallel.

We restrict our study in this paper to the case where C is sparse. We observe that in some practical applications the matrix A contains a small number of rows that have many more nonzero entries than the other rows, resulting in a dense matrix C. Several techniques, including matrix stretching and using the augmented system, have been proposed to handle this type of problem. These result in solving a transformed system of sparse normal equations, see for example [40] and the references therein.

In [3], Al Daas and Grigori presented a class of robust fully algebraic two-level additive Schwarz preconditioners for solving SPD linear systems of equations. They introduced the notion of an algebraic local symmetric positive semi-definite (SPSD) splitting of an SPD matrix with respect to local subdomains. They used this splitting to construct a class of second-level spaces that bound the spectral condition number of the preconditioned system by a user-defined value. Unfortunately, Al Daas and Grigori reported that for general sparse SPD matrices, constructing the splitting is prohibitively expensive. Our interest is in examining whether the particular structure of the normal equations matrix allows the approach to be successfully used for preconditioning LS problems. In this paper, we show how to compute the splitting efficiently. Based on this splitting, we apply the theory presented in [3] to construct a two-level Schwarz preconditioner for the normal equations.

Note that for most existing preconditioners of the normal equations, there is no need to form and store the normal equations matrix C explicitly. For example, the lower triangular part of its columns can be computed one at a time, used to perform the corresponding step of an incomplete Cholesky algorithm, and then discarded. However, forming the normal equations matrix, even piecemeal, can entail a significant overhead and can potentially lead to a severe loss of information in highly ill-conditioned cases. Although building our proposed preconditioner does not need the explicit computation of C, our parallel implementation computes it efficiently

and uses it to setup the preconditioner. This is mainly motivated by technical reasons. As an example, state-of-the-art distributed-memory graph partitioners such as ParMETIS [28] or PT-SCOTCH [36] cannot directly partition the columns of the *rectangular* matrix A. Our numerical experiments on highly ill-conditioned LS problems showed that forming C and using a positive diagonal shift to construct the preconditioner had no major effect on the robustness of the resulting preconditioner.

This paper is organized as follows. The notation used in the manuscript is given at the end of the introduction. In section 2, we present an overview of domain decomposition (DD) methods for a sparse SPD matrix. We present a framework for the DD approach when applied to the sparse LS problem in section 3. Afterwards, we show how to compute the local SPSD splitting matrices efficiently and use them in line with the theory presented in [3] to construct a robust two-level Schwarz preconditioner for the normal equations matrix. We then discuss some technical details that clarify how to construct the preconditioner efficiently. In section 4, we briefly discuss how the new preconditioner can be implemented on top of the PETSc library [7] and we illustrate its effectiveness using large-scale LS problems coming from practical applications. Finally, concluding comments are made in section 5.

Notation. We end our introduction by defining notation that will be used in this paper. Let $1 \leq n \leq m$ and let $A \in \mathbb{R}^{m\times n}$. Let $S_1 \subset [\![1, m]\!]$ and $S_2 \subset [\![1, n]\!]$ be two sets of integers. $A(S_1, :)$ is the submatrix of A formed by the rows whose indices belong to S_1 and $A(:, S_2)$ is the submatrix of A formed by the columns whose indices belong to S_2. The matrix $A(S_1, S_2)$ is formed by taking the rows whose indices belong to S_1 and only retaining the columns whose indices belong to S_2. The concatenation of any two sets of integers S_1 and S_2 is represented by $[S_1, S_2]$. Note that the order of the concatenation is important. The set of the first p positive integers is denoted by $[\![1, p]\!]$. The identity matrix of size n is denoted by I_n. We denote by $ker(A)$ and $range(A)$ the null space and the range of A, respectively.

2. Introduction to domain decomposition. Throughout this section, we assume that C is a general $n \times n$ sparse SPD matrix. Let the nodes V in the corresponding adjacency graph $\mathcal{G}(C)$ be numbered from 1 to n. A graph partitioning algorithm can be used to split V into $N \ll n$ disjoint subsets Ω_{Ii} $(1 \leq i \leq N)$ of size n_{Ii}. These sets are called nonoverlapping subdomains. Defining an overlapping additive Schwarz preconditioner requires overlapping subdomains. Let $\Omega_{\Gamma i}$ be the subset of size $n_{\Gamma i}$ of nodes that are distance one in $\mathcal{G}(C)$ from the nodes in Ω_{Ii} $(1 \leq i \leq N)$. The overlapping subdomain Ω_i is defined to be $\Omega_i = [\Omega_{Ii}, \Omega_{\Gamma i}]$, with size $n_i = n_{\Gamma i} + n_{Ii}$.

Associated with Ω_i is a restriction (or projection) matrix $R_i \in \mathbb{R}^{n_i\times n}$ given by $R_i = I_n(\Omega_i, :)$. R_i maps from the global domain to subdomain Ω_i. Its transpose $R_i^\top$ is a prolongation matrix that maps from subdomain Ω_i to the global domain. The *one-level additive Schwarz preconditioner* [16] is defined to be

$$M_{\text{ASM}}^{-1} = \sum_{i=1}^{N} R_i^\top C_{ii}^{-1} R_i, \qquad C_{ii} = R_i C R_i^\top. \tag{2.1}$$

That is,

$$M_{\text{ASM}}^{-1} = \mathcal{R}_1 \begin{pmatrix} C_{11}^{-1} & & \\ & \ddots & \\ & & C_{NN}^{-1} \end{pmatrix} \mathcal{R}_1^\top,$$

where $\mathcal{R}_1$ is the one-level interpolation operator defined by

$$\begin{aligned} \mathcal{R}_1 \ : \ & \prod_{i=1}^{N} \mathbb{R}^{n_i} \to \mathbb{R}^n \\ & (u_i)_{1\leq i\leq N} \mapsto \sum_{i=1}^{N} R_i^\top u_i. \end{aligned}$$

Applying this preconditioner to a vector involves solving concurrent local problems in the overlapping subdomains. Increasing N reduces the sizes n_i of the overlapping subdomains, leading to smaller local problems and faster computations. However, in practice, the preconditioned system using M_{ASM}^{-1} may not be well-conditioned, inhibiting convergence of the iterative solver. In fact, the local nature of this preconditioner can lead to a deterioration in its effectiveness as the number of subdomains increases because of the lack of global information from the matrix C [16, 21]. To maintain robustness with respect to N, an artificial subdomain is added to the preconditioner (also known as second-level correction or coarse correction) that includes global information.

Let $0 < n_0 \ll n$. If $R_0 \in \mathbb{R}^{n_0 \times n}$ is of full row rank, the *two-level additive Schwarz preconditioner* [16] is defined to be

$$M_{\text{additive}}^{-1} = \sum_{i=0}^{N} R_i^\top C_{ii}^{-1} R_i = R_0^\top C_{00}^{-1} R_0 + M_{\text{ASM}}^{-1}, \qquad C_{00} = R_0 C R_0^\top. \tag{2.2}$$

That is,

$$M_{\text{additive}}^{-1} = \mathcal{R}_2 \begin{pmatrix} C_{00}^{-1} & & & \\ & C_{11}^{-1} & & \\ & & \ddots & \\ & & & C_{NN}^{-1} \end{pmatrix} \mathcal{R}_2^\top,$$

where $\mathcal{R}_2$ is the two-level interpolation operator

$$\begin{aligned} \mathcal{R}_2 \ : \ & \prod_{i=0}^{N} \mathbb{R}^{n_i} \to \mathbb{R}^n \\ & (u_i)_{0\leq i\leq N} \mapsto \sum_{i=0}^{N} R_i^\top u_i. \end{aligned} \tag{2.3}$$

In the rest of this paper, we will make use of the canonical one-to-one correspondence between $\prod_{i=0}^{N} \mathbb{R}^{n_i}$ and $\mathbb{R}^{\sum_{i=0}^{N} n_i}$ so that $\mathcal{R}_2$ can be applied to vectors in $\mathbb{R}^{\sum_{i=0}^{N} n_i}$. Observe that, because C and R_0 are of full rank, C_{00} is also of full rank. For any full rank R_0, it is possible to cheaply obtain upper bounds on the largest eigenvalue of the preconditioned matrix, independently of n and N [3]. However, bounding the smallest eigenvalue is highly dependent on R_0. Thus, the choice of R_0 is key to obtaining a well-conditioned system and building efficient two-level Schwarz preconditioners. Two-level Schwarz preconditioners have been used to solve a large class of systems arising from a range of engineering applications (see, for example, [23, 27, 29, 31, 41, 42, 45] and references therein).

Following [3], we denote by $D_i \in \mathbb{R}^{n_i \times n_i}$ $(1 \le i \le N)$ any non-negative diagonal matrices such that

$$\sum_{i=1}^{N} R_i^\top D_i R_i = I_n. \tag{2.4}$$

We refer to $(D_i)_{1 \le i \le N}$ as an *algebraic partition of unity*. In [3], Al Daas and Grigori show how to select local subspaces $Z_i \in \mathbb{R}^{n_i \times p_i}$ with $p_i \ll n_i$ $(1 \le i \le N)$ such that, if $R_0^\top$ is defined to be $R_0^\top = [R_1^\top D_1 Z_1, \ldots, R_N^\top D_N Z_N]$, the spectral condition number of the preconditioned matrix $M_{\text{additive}}^{-1} C$ is bounded from above independently of N and n.

2.1. Algebraic local SPSD splitting of an SPD matrix. We now recall the definition of an algebraic local SPSD splitting of an SPD matrix given in [3]. This requires some additional notation. Denote the complement of Ω_i in $[\![1, n]\!]$ by Ω_{ci}. Define restriction matrices R_{ci}, R_{Ii}, and $R_{\Gamma i}$ that map from the global domain to Ω_{ci}, Ω_{Ii}, and $\Omega_{\Gamma i}$, respectively. Reordering the matrix C using the permutation matrix $P_i = I_n([\Omega_{Ii}, \Omega_{\Gamma i}, \Omega_{ci}], :)$ gives the block tridiagonal matrix

$$P_i C P_i^\top = \begin{pmatrix} C_{I,i} & C_{I\Gamma,i} & \\ C_{\Gamma I,i} & C_{\Gamma,i} & C_{\Gamma c,i} \\ & C_{c\Gamma,i} & C_{c,i} \end{pmatrix}, \tag{2.5}$$

where $C_{I,i} = R_{Ii} C R_{Ii}^\top$, $C_{\Gamma I,i}^\top = C_{I\Gamma,i} = R_{Ii} C R_{\Gamma i}^\top$, $C_{\Gamma,i} = R_{\Gamma i} C R_{\Gamma i}^\top$, $C_{c\Gamma,i}^\top = C_{\Gamma c,i} = R_{\Gamma i} C R_{ci}^\top$, and $C_{c,i} = R_{ci} C R_{ci}^\top$. The first block on the diagonal corresponds to the nodes in Ω_{Ii}, the second block on the diagonal corresponds to the nodes in Ω_{Γ_i}, and the third block on the diagonal is associated with the remaining nodes.

An *algebraic local SPSD splitting* of the SPD matrix C with respect to the i-th subdomain is defined to be any SPSD matrix $\widetilde{C}_i \in \mathbb{R}^{n \times n}$ of the form

$$P_i \widetilde{C}_i P_i^\top = \begin{pmatrix} C_{I,i} & C_{I\Gamma,i} & 0 \\ C_{\Gamma I,i} & \widetilde{C}_{\Gamma,i} & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

such that the following condition holds:

$$0 \le u^\top \widetilde{C}_i u \le u^\top C u, \quad \text{for all } u \in \mathbb{R}^n.$$

We denote the 2×2 block nonzero matrix of $P_i \widetilde{C}_i P_i^\top$ by $\widetilde{C}_{ii}$ so that

$$\widetilde{C}_i = R_i^\top \widetilde{C}_{ii} R_i.$$

Associated with the local SPSD splitting matrices, we define a multiplicity constant k_m that satisfies the inequality

$$0 \le \sum_{i=1}^{N} u^\top \widetilde{C}_i u \le k_m u^\top C u, \quad \text{for all } u \in \mathbb{R}^n. \tag{2.6}$$

Note that, for any set of SPSD splitting matrices, $k_m \le N$.

The main motivation for defining splitting matrices is to find local seminorms that are bounded from above by the C-norm. These seminorms will be used to determine a subspace that contains the eigenvectors of C associated with its smallest eigenvalues.

2.2. Two-level Schwarz method. We next review the abstract theory of the two-level Schwarz method as presented in [3]. For the sake of completeness, we present some elementary lemmas that are widely used in multilevel methods. These will be used in proving efficiency of the two-level Schwarz preconditioner and will also help in understanding how the preconditioner is constructed.

2.2.1. Useful lemmas. The following lemma [34] provides a unified framework for bounding the spectral condition number of a preconditioned operator. It can be found in different forms for finite and infinite dimensional spaces. Here, we follow the presentation from [16, Lemma 7.4].

LEMMA 2.1 (Fictitious Subspace Lemma). *Let $C \in \mathbb{R}^{n_C \times n_C}$ and $B \in \mathbb{R}^{n_B \times n_B}$ be SPD. Let the operator $\mathcal{R}$ be defined as*

$$\begin{aligned} \mathcal{R} \; : \; \mathbb{R}^{n_B} &\to \mathbb{R}^{n_C} \\ v &\mapsto \mathcal{R}v, \end{aligned}$$

and let $\mathcal{R}^\top$ be its transpose. Assume the following conditions hold:
(i) $\mathcal{R}$ is surjective;
(ii) there exists $c_u > 0$ such that for all $v \in \mathbb{R}^{n_B}$

$$(\mathcal{R}v)^\top C (\mathcal{R}v) \leq c_u v^\top B v;$$

(iii) there exists $c_l > 0$ such that for all $v_C \in \mathbb{R}^{n_C}$ there exists $v_B \in \mathbb{R}^{n_B}$ such that $v_C = \mathcal{R}v_B$ and

$$c_l v_B^\top B v_B \leq (\mathcal{R}v_B)^\top C (\mathcal{R}v_B) = v_C^\top C v_C.$$

Then, the spectrum of the operator $\mathcal{R}B^{-1}\mathcal{R}^\top C$ is contained in the interval $[c_l, c_u]$.

The challenge is to define the second-level projection matrix R_0 such that the two-level additive Schwarz preconditioner M^{-1}_{additive} and the operator $\mathcal{R}_2$ (2.3), corresponding respectively to B and $\mathcal{R}$ in Lemma 2.1, satisfy conditions *(i)* to *(iii)* and, in addition, ensures the ratio between c_l and c_u is small because this determines the quality of the preconditioner.

As shown in [16, Lemmas 7.10 and 7.11], a two-level additive Schwarz preconditioner satisfies *(i)* and *(ii)* for any full rank R_0. Furthermore, the constant c_u is bounded from above independently of the number of subdomains N, as shown in the following result [10, Theorem 12].

LEMMA 2.2. *Let k_c be the minimum number of distinct colours so that the spaces spanned by the columns of the matrices $R_1^\top, \ldots, R_N^\top$ that are of the same colour are mutually C-orthogonal. Then,*

$$(\mathcal{R}_2 u_{\mathcal{B}})^\top C (\mathcal{R}_2 u_{\mathcal{B}}) \leq (k_c + 1) \sum_{i=0}^{N} u_i^\top C_{ii} u_i,$$

for all $u_{\mathcal{B}} = (u_i)_{0 \leq i \leq N} \in \prod_{i=0}^{N} \mathbb{R}^{n_i}$.

Note that k_c is independent of N. Indeed, it depends only on the sparsity structure of C and is less than the maximum number of neighbouring subdomains.

The following result is the first step in a three-step approach to define a two-level additive Schwarz operator $\mathcal{R}_2$ that satisfies condition *(iii)* in Lemma 2.1.

LEMMA 2.3. *[16, Lemma 7.12] Let* $u_{\mathcal{B}} = (u_i)_{0\le i\le N} \in \prod_{i=0}^{N} \mathbb{R}^{n_i}$ *and* $u = \mathcal{R}_2 u_{\mathcal{B}} \in \mathbb{R}^n$. *Then, provided* R_0 *is of full rank,*

$$\sum_{i=0}^{N} u_i^\top C_{ii} u_i \le 2\ u^\top C u + (2k_c+1) \sum_{i=1}^{N} u_i^\top C_{ii} u_i,$$

where k_c *is defined in Lemma 2.2.*

It follows that *(iii)* is satisfied if the squared localized seminorm $u_i^\top C_{ii} u_i$ is bounded from above by the squared C-norm of u.

In the second step, we bound $u_i^\top C_{ii} u_i$ by the squared localized seminorm defined by the SPSD splitting matrix $\widetilde{C}_i$, which can be bounded by the squared C-norm (2.6). The decomposition of $u = \sum_{i=0}^{N} R_i^\top u_i \in \mathbb{R}^n$ is termed *stable* if, for some $\tau > 0$,

$$\tau u_i^\top C_{ii} u_i \le u^\top C u, \qquad 1 \le i \le N.$$

The two-level approach in [3] aims to decompose each $\mathbb{R}^{n_i}$ $(1 \le i \le N)$ into two subspaces, one that makes the decomposition of u stable and the other is part of the artificial subdomain associated with the second level of the preconditioner. Given the partition of unity (2.4), $u = \sum_{i=1}^{N} R_i^\top D_i R_i u$ and, if $\Pi_i = \Pi_i^\top \in \mathbb{R}^{n_i \times n_i}$, we can write

$$\begin{aligned} u &= \sum_{i=1}^{N} R_i^\top D_i (I_{n_i} - \Pi_i) R_i u + \sum_{i=1}^{N} R_i^\top D_i \Pi_i R_i u \\ &= \sum_{i=1}^{N} R_i^\top u_i + \sum_{i=1}^{N} R_i^\top D_i \Pi_i R_i u, \quad \text{with } u_i = D_i \left(I_{n_i} - \Pi_i\right) R_i u. \end{aligned}$$

Therefore, we need to construct Π_i such that

$$\tau u^\top R_i^\top (I_{n_i} - \Pi_i) D_i C_{ii} D_i (I_{n_i} - \Pi_i) R_i u \le u^\top C u.$$

The following lemma shows how this can be done.

LEMMA 2.4. *[3, Lemma 4.2] Let* $\widetilde{C}_i = R_i^\top \widetilde{C}_{ii} R_i$ *be a local SPSD splitting of* C *related to the* i*-th subdomain* $(1 \le i \le N)$. *Let* D_i *be the partition of unity* (2.4). *Let* $P_{0,i}$ *be the projection on* $range(\widetilde{C}_{ii})$ *parallel to* $ker(\widetilde{C}_{ii})$. *Define* $L_i = ker(D_i C_{ii} D_i) \cap ker(\widetilde{C}_{ii})$, *and let* $L_i^\perp$ *denote the orthogonal complementary of* L_i *in* $ker(\widetilde{C}_{ii})$. *Consider the following generalized eigenvalue problem:*

$$\begin{aligned} &\textit{find } (v_{i,k}, \lambda_{i,k}) \ \in \mathbb{R}^{n_i} \times \mathbb{R} \\ &\textit{such that } P_{0,i} D_i C_{ii} D_i P_{0,i} v_{i,k} = \lambda_{i,k} \widetilde{C}_{ii} v_{i,k}. \end{aligned}$$

Given $\tau > 0$, *define*

$$\mathcal{Z}_i = L_i^\perp \oplus span\left\{ v_{i,k} \mid \lambda_{i,k} > \frac{1}{\tau} \right\} \tag{2.7}$$

and let Π_i *be the orthogonal projection on* $\mathcal{Z}_i$. *Then,* $\mathcal{Z}_i$ *is the subspace of smallest dimension such that for all* $u \in \mathbb{R}^n$,

$$\tau u_i^\top C_{ii} u_i \le u^\top \widetilde{C}_i u \le u^\top C u,$$

where $u_i = D_i \left(I_{n_i} - \Pi_i\right) R_i u$.

Lemma 2.5 provides the last step that we need for condition *(iii)* in Lemma 2.1. It defines u_0 and checks whether $(u_i)_{0\leq i\leq N}$ is a stable decomposition.

LEMMA 2.5. *Let $\widetilde{C}_i$, $\mathcal{Z}_i$, and Π_i be as in Lemma 2.4 and let Z_i be a matrix whose columns span $\mathcal{Z}_i$ $(1 \leq i \leq N)$. Let the columns of the matrix $R_0^\top$ span the space*

$$\mathcal{Z} = \bigoplus_{i=1}^{N} R_i^\top D_i Z_i. \tag{2.8}$$

Let $u \in \mathbb{R}^n$ and $u_i = D_i \left(I_{n_i} - \Pi_i\right) R_i u$ $(1 \leq i \leq N)$. Define

$$u_0 = \left(R_0 R_0^\top\right)^{-1} R_0 \left(\sum_{i=1}^{N} R_i^\top D_i \Pi_i R_i u\right).$$

Then,

$$u = \sum_{i=0}^{N} R_i^\top u_i,$$

and

$$\sum_{i=0}^{N} u_i^\top C_{ii} u_i \leq \left(2 + (2k_c+1)\frac{k_m}{\tau}\right) u^\top C u.$$

Finally, using the preceding results, Theorem 2.6 presents a theoretical upper bound on the spectral condition number of the preconditioned system.

THEOREM 2.6. *If the two-level additive Schwarz preconditioner $M^{-1}_{additive}$ (2.2) is constructed using R_0 as defined in Lemma 2.5, then the following inequality is satisfied:*

$$\kappa\left(M^{-1}_{additive} C\right) \leq (k_c+1)\left(2 + (2k_c+1)\frac{k_m}{\tau}\right).$$

2.3. Variants of the Schwarz preconditioner. So far, we have presented M^{-1}_{ASM}, the symmetric additive Schwarz method (ASM) and M^{-1}_{additive}, the additive correction for the second level. It was noted in [9] that using the partition of unity to weight the preconditioner can improve its quality. The resulting preconditioner is referred to as M^{-1}_{RAS}, the *restricted additive Schwarz* (RAS) preconditioner, and is defined to be

$$M^{-1}_{\text{RAS}} = \sum_{i=1}^{N} R_i^\top D_i C_{ii}^{-1} R_i. \tag{2.9}$$

This preconditioner is nonsymmetric and thus can only be used with iterative methods such as GMRES [37] that are for solving nonsymmetric problems. With regards to the second level, different strategies yield either a symmetric or a nonsymmetric preconditioner [44]. Given a first-level preconditioner $M_\star^{-1}$ and setting $Q = R_0^\top C_{00}^{-1} R_0$, the balanced and deflated two-level preconditioners are as follows

$$M^{-1}_{\text{balanced}} = Q + (I - CQ)^\top M_\star^{-1}(I - CQ), \tag{2.10}$$

and

$$M^{-1}_{\text{deflated}} = Q + M_\star^{-1}(I - CQ), \tag{2.11}$$

respectively. It is well-known in the literature that M^{-1}_{balanced} and M^{-1}_{deflated} yield better convergence behavior than M^{-1}_{additive} (see [44] for a thorough comparison). Although the theory we present relies on M^{-1}_{additive}, in practice we will use M^{-1}_{balanced} and M^{-1}_{deflated}. If the one-level preconditioner $M^{-1}_{\star}$ is symmetric, then so is M^{-1}_{balanced}, while M^{-1}_{deflated} is typically nonsymmetric. For this reason, in the rest of the paper, we always couple M^{-1}_{ASM} with M^{-1}_{balanced}, and M^{-1}_{RAS} with M^{-1}_{deflated}. All three variants have the same setup cost, and only differ in how the second level is applied. M^{-1}_{balanced} is slightly more expensive because two second-level corrections (multiplications by Q) are required instead of a single one for M^{-1}_{additive} and M^{-1}_{deflated}.

3. The normal equations. The theory explained thus far is fully algebraic but somewhat disconnected from our initial LS problem (1.1). We now show how it can be readily applied to the normal equations matrix $C = A^\top A$, with $A \in \mathbb{R}^{m\times n}$ sparse, first defining a one-level Schwarz preconditioner, and then a robust algebraic second-level correction. We start by partitioning the n columns of A into disjoint subsets Ω_{Ii}. Let Ξ_i be the set of indices of the nonzero rows in $A(:,\Omega_{Ii})$ and let Ξ_{ci} be the complement of Ξ_i in the set $[\![1,m]\!]$. Now define $\Omega_{\Gamma i}$ to be the complement of Ω_{Ii} in the set of indices of nonzero columns of $A(\Xi_i,:)$. The set $\Omega_i = [\Omega_{Ii},\Omega_{\Gamma i}]$ defines the i-th overlapping subdomain and we have the permuted matrix

$$A([\Xi_i,\Xi_{ci}],[\Omega_{Ii},\Omega_{\Gamma i},\Omega_{ci}]) = \begin{pmatrix} A_{I,i} & A_{I\Gamma,i} & \\ & A_{\Gamma,i} & A_{c,i} \end{pmatrix}. \tag{3.1}$$

To illustrate the concepts and notation, consider the 5×4 matrix

$$A = \begin{pmatrix} 1 & 0 & 6 & 0 \\ 2 & 4 & 0 & 0 \\ 3 & 0 & 0 & 0 \\ 0 & 5 & 0 & 7 \\ 0 & 0 & 0 & 8 \end{pmatrix}$$

and set $N = 2$, $\Omega_{I1} = \{1,3\}$, $\Omega_{I2} = \{2,4\}$. Consider the first subdomain. We have

$$A(:,\Omega_{I1}) = \begin{pmatrix} 1 & 6 \\ 2 & 0 \\ 3 & 0 \\ 0 & 0 \\ 0 & 0 \end{pmatrix}.$$

The set of indices of the nonzero rows is $\Xi_1 = \{1,2,3\}$, and its complement is $\Xi_{c1} = \{4,5\}$. To define $\Omega_{\Gamma,1}$, select the nonzero columns in the submatrix $A(\Xi_1,:)$ and remove those already in Ω_{I1}, that is,

$$A(\Xi_1,:) = \begin{pmatrix} 1 & 0 & 6 & 0 \\ 2 & 4 & 0 & 0 \\ 3 & 0 & 0 & 0 \end{pmatrix}, \tag{3.2}$$

so that $\Omega_{\Gamma 1} = \{2\}$ and $\Omega_{c1} = \{4\}$. Permuting A to the form (3.1) gives

$$A([\Xi_1,\Xi_{c1}],[\Omega_{I1},\Omega_{\Gamma 1},\Omega_{c1}]) = \begin{pmatrix} 1 & 6 & 0 & 0 \\ 2 & 0 & 4 & 0 \\ 3 & 0 & 0 & 0 \\ 0 & 0 & 5 & 7 \\ 0 & 0 & 0 & 8 \end{pmatrix}.$$

In the same way, consider the second subdomain. $\Omega_{I2} = \{2, 4\}$ and

$$A(:, \Omega_{I2}) = \begin{pmatrix} 0 & 0 \\ 4 & 0 \\ 0 & 0 \\ 5 & 7 \\ 0 & 8 \end{pmatrix},$$

so that $\Xi_2 = \{2, 4, 5\}$ and $\Xi_{c2} = \{1, 3\}$. To define $\Omega_{\Gamma 2}$, select the nonzero columns in the submatrix $A(\Xi_2, :)$ and remove those already in Ω_{I2}, that is,

$$A(\Xi_2, :) = \begin{pmatrix} 2 & 4 & 0 & 0 \\ 0 & 5 & 0 & 7 \\ 0 & 0 & 0 & 8 \end{pmatrix}, \tag{3.3}$$

which gives $\Omega_{\Gamma 2} = \{1\}$ and $\Omega_{c2} = \{3\}$. Permuting A to the form (3.1) gives

$$A([\Xi_2, \Xi_{c2}], [\Omega_{I2}, \Omega_{\Gamma 2}, \Omega_{c2}]) = \begin{pmatrix} 4 & 0 & 2 & 0 \\ 5 & 7 & 0 & 0 \\ 0 & 8 & 0 & 0 \\ 0 & 0 & 1 & 6 \\ 0 & 0 & 3 & 0 \end{pmatrix}.$$

Now that we have Ω_{Ii} and $\Omega_{\Gamma i}$, we can define the restriction operators

$$R_1 = I_4(\Omega_1, :) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix}, \quad R_2 = I_4(\Omega_2, :) = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \end{pmatrix}.$$

For our example, $n_{I1} = n_{I2} = 2$ and $n_{\Gamma 1} = n_{\Gamma 2} = 1$. The partition of unity matrices D_i are of dimension $(n_{Ii} + n_{\Gamma i}) \times (n_{Ii} + n_{\Gamma i})$ $(i = 1, 2)$ and have ones on the n_{Ii} leading diagonal entries and zeros elsewhere, so that

$$D_1 = D_2 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{pmatrix}. \tag{3.4}$$

Observe that $D_i(k, k)$ scales the columns $A(:, \Omega_i(k))$.

Note that it is possible to obtain the partitioning sets and the sets of indices using the normal equations matrix C. Most graph partitioners, especially those that are implemented in parallel, require an undirected graph (corresponding to a square matrix with a symmetric sparsity pattern). Therefore, in practice, we use the graph of C to setup the first-level preconditioner for LS problems.

3.1. One-level DD for the normal equations. This section presents the one-level additive Schwarz preconditioner for the normal equations matrix $C = A^\top A$. Following (2.1) and given the sets $\Omega_{Ii}, \Omega_{\Gamma i}$, and Ξ_i, the one-level Schwarz preconditioner of $C = A^\top A$ is

$$\begin{aligned} M_{\text{ASM}}^{-1} &= \sum_{i=1}^{N} R_i^\top \left(R_i A^\top A R_i^\top \right)^{-1} R_i, \\ &= \sum_{i=1}^{N} R_i^\top \left(A(:, \Omega_i)^\top A(:, \Omega_i) \right)^{-1} R_i, \end{aligned}$$

Remark 3.1. Note that the local matrix $C_{ii} = A(:,\Omega_i)^\top A(:,\Omega_i)$ need not be computed explicitly to be factored. Instead, the Cholesky factor of C_{ii} can be computed by using a "thin" QR factorization of $A(:,\Omega_i)$.

3.2. Algebraic local SPSD splitting of the normal equations matrix. In this section, we show how to cheaply construct algebraic local SPSD splittings for sparse matrices of the form $C = A^\top A$. Combining (2.5) and (3.1), we can write

$$P_i A^\top A P_i^\top = \begin{pmatrix} A_{I,i}^\top A_{I,i} & A_{I,i}^\top A_{I\Gamma,i} & \\ A_{I\Gamma,i}^\top A_{I,i} & A_{I\Gamma,i}^\top A_{I\Gamma,i} + A_{\Gamma,i}^\top A_{\Gamma,i} & A_{\Gamma,i}^\top A_{\mathrm{c},i} \\ & A_{\mathrm{c},i}^\top A_{\Gamma,i} & A_{\mathrm{c},i}^\top A_{\mathrm{c},i} \end{pmatrix},$$

where $P_i = I_n([\Omega_{Ii}, \Omega_{\Gamma i}, \Omega_{\mathrm{c}i}], :)$ is a permutation matrix. A straightforward splitting of $P_i A^\top A P_i^\top$ is given by

$$P_i A^\top A P_i^\top = \begin{pmatrix} A_{I,i}^\top A_{I,i} & A_{I,i}^\top A_{I\Gamma,i} & 0 \\ A_{I\Gamma,i}^\top A_{I,i} & A_{I\Gamma,i}^\top A_{I\Gamma,i} & 0 \\ 0 & 0 & 0 \end{pmatrix} + \begin{pmatrix} 0 & 0 & 0 \\ 0 & A_{\Gamma,i}^\top A_{\Gamma,i} & A_{\Gamma,i}^\top A_{\mathrm{c},i} \\ 0 & A_{\mathrm{c},i}^\top A_{\Gamma,i} & A_{\mathrm{c},i}^\top A_{\mathrm{c},i} \end{pmatrix}.$$

It is clear that both summands are SPSD. Indeed, they both have the form $X^\top X$, where X is $\begin{pmatrix} A_{I,i} & A_{I\Gamma,i} & 0 \end{pmatrix}$ and $\begin{pmatrix} 0 & A_{\Gamma,i} & A_{\mathrm{c},i} \end{pmatrix}$, respectively. The local SPSD splitting matrix related to the i-th subdomain is then defined as:

$$\widetilde{C}_{ii} = A(\Xi_i,\Omega_i)^\top A(\Xi_i,\Omega_i) = \begin{pmatrix} A_{I,i} & A_{I\Gamma,i} \end{pmatrix}^\top \begin{pmatrix} A_{I,i} & A_{I\Gamma,i} \end{pmatrix}, \tag{3.5}$$

and

$$\widetilde{C}_i = R_i^\top \widetilde{C}_{ii} R_i = A(\Xi_i,:)^\top A(\Xi_i,:).$$

Hence, the theory presented in [3] and summarised in subsection 2.2 is applicable. In particular, the two-level Schwarz preconditioner M_{additive}^{-1} (2.2) satisfies

$$\kappa(M_{\text{additive}}^{-1} C) \leq (k_c+1)\left(2 + 2(k_c+1)\frac{k_m}{\tau}\right),$$

where k_c is the minimal number of colours required to colour the partitions of C such that each two neighbouring subdomains have different colours, and k_m is the multiplicity constant that satisfies the following inequality

$$\sum_{i=1}^{N} R_i^\top \widetilde{C}_{ii} R_i \leq k_m C.$$

The constant k_c is independent of N and depends only on the graph $\mathcal{G}(C)$, which is determined by the sparsity pattern of A. The multiplicity constant k_m depends on the local SPSD splitting matrices. For the normal equations matrix, the following lemma provides an upper bound on k_m.

LEMMA 3.2. *Let $C = A^\top A$. Let m_j be the number of subdomains such that $A(j,\Omega_{Ii}) \neq 0$ $(1 \leq i \leq N)$, that is,*

$$m_j = \#\{i \mid j \in \Xi_i\}.$$

Then, k_m can be chosen to be $k_m = \max_{1\leq j\leq m} m_j$. Furthermore, if k_{Ω_i} is the number of neighbouring subdomains of the i-th subdomain, that is,

$$k_{\Omega_i} = \#\{j \mid \Omega_i \cap \Omega_j \neq \phi\},$$

then

$$k_m = \max_{1 \leq j \leq m} m_j \leq \max_{1 \leq i \leq N} k_{\Omega_i}.$$

Proof. Since $C = A^\top A$ and $\widetilde{C}_i = A(\Xi_i, :)^\top A(\Xi_i, :)$, we have

$$u^\top C u = \sum_{j=1}^{m} u^\top A(j,:)^\top A(j,:) u,$$

$$u^\top \widetilde{C}_i u = \sum_{j \in \Xi_i} u^\top A(j,:)^\top A(j,:) u,$$

$$\sum_{i=1}^{N} u^\top \widetilde{C}_i u = \sum_{i=1}^{N} \sum_{j \in \Xi_i} u^\top A(j,:)^\top A(j,:) u.$$

From the definition of m_j, the term $u^\top A(j,:)^\top A(j,:)u$ appears m_j times in the last equation. Thus,

$$\begin{aligned} \sum_{i=1}^{N} u^\top \widetilde{C}_i u &= \sum_{j=1}^{m} m_j u^\top A(j,:)^\top A(j,:) u, \\ &\leq \max_{1 \leq j \leq m} m_j \sum_{j=1}^{m} u^\top A(j,:)^\top A(j,:) u, \\ &= \max_{1 \leq j \leq m} m_j (u^\top C u), \end{aligned}$$

from which it follows that we can choose $k_m = \max_{1 \leq j \leq m} m_j$. Now, if $1 \leq l \leq m$, there exist $i_1, \ldots, i_{m_l}$ such that $l \in \Xi_{i_1} \cap \cdots \cap \Xi_{i_{m_l}}$. Furthermore, $m_l \leq \max_{1 \leq p \leq l} k_{\Omega_{i_p}}$. Taking the maximum over l on both sides, we obtain

$$k_m \leq \max_{1 \leq i \leq N} k_{\Omega_i}.$$ ∎

Note that because A is sparse, k_m is independent of the number of subdomains.

3.3. Algorithms and technical details. In this section, we discuss the technical details involved in constructing a two-level preconditioner for the normal equations matrix.

3.3.1. Partition of unity. Because the matrix $A_{I\Gamma,i}$ may be of low rank, the null space of $\widetilde{C}_{ii}$ (3.5) can be large. Recall that the diagonal matrices D_i have dimension $n_i = n_{Ii} + n_{\Gamma i}$. Choosing the entries in positions $n_{Ii}+1, \ldots, n_i$ of the diagonal of D_i to be zero, as in (3.4), results in the subspace of $ker(\widetilde{C}_{ii})$ caused by the rank deficiency of $A_{I\Gamma,i}$ to lie within $ker(D_i C_{ii} D_i)$, reducing the size of the space $\mathcal{Z}$ given by (2.8). In other words, if $A_{I\Gamma,i} u = 0$, we have $\widetilde{C}_{ii} v = 0$, where $v^\top = (0, u^\top)$, i.e., $v \in ker(\widetilde{C}_{ii})$ and because by construction $D_i v = 0$, we have $v \in ker(\widetilde{C}_{ii}) \cap ker(D_i C_{ii} D_i)$, therefore, v need not be included in $\mathcal{Z}_i$.

3.3.2. The eigenvalue problem. The generalized eigenvalue problem presented in Lemma 2.4 is critical in the construction of the two-level preconditioner. Although the definition of $\mathcal{Z}_i$ from (2.7) suggests it is necessary to compute the null

space of $\widetilde{C}_{ii}$ and that of $D_iC_{ii}D_i$ and their intersection, in practice, this can be avoided. Consider the generalized eigenvalue problem

$$D_iC_{ii}D_iv = \lambda\widetilde{C}_{ii}v, \tag{3.6}$$

where, by convention, we set $\lambda = 0$ if $v \in ker(\widetilde{C}_{ii}) \cap ker(D_iC_{ii}D_i)$ and $\lambda = \infty$ if $v \in ker(\widetilde{C}_{ii}) \setminus ker(D_iC_{ii}D_i)$. The subspace $\mathcal{Z}_i$ defined in (2.7) can then be written as

$$\text{span}\left\{ v \mid D_iC_{ii}D_iv = \lambda\widetilde{C}_{ii}v \text{ and } \lambda > \frac{1}{\tau} \right\}.$$

Consider also the shifted generalized eigenvalue problem

$$D_iC_{ii}D_iv = \lambda(\widetilde{C}_{ii} + sI_{n_i})v, \tag{3.7}$$

where $0 < s \ll 1$. Note that if s is such that $\widetilde{C}_{ii} + sI_{n_i}$ is numerically of full rank, (3.7) can be solved using any off-the-shelf generalized eigenproblem solver. Let (v, λ) be an eigenpair of (3.7). Then, we can only have one of the following situations:

- $v \in range(\widetilde{C}_{ii}) \cap ker(D_iC_{ii}D_i)$ or $v \in ker(\widetilde{C}_{ii}) \cap ker(D_iC_{ii}D_i)$. In which case, $(v, 0)$ is an eigenpair of (3.6).
- $v \in range(\widetilde{C}_{ii}) \cap range(D_iC_{ii}D_i)$. Then,

$$\frac{\|D_iC_{ii}D_iv - \lambda\widetilde{C}_{ii}v\|_2}{\lambda\|v\|_2} = s,$$

  and, as s is small, (v, λ) is a good approximation of an eigenpair of (3.6) corresponding to a finite eigenvalue.
- $v \in ker(\widetilde{C}_{ii}) \cap range(D_iC_{ii}D_i)$. Then, $D_iC_{ii}D_iv = \lambda sv$, i.e., λs is a nonzero eigenvalue of $D_iC_{ii}D_i$. Because D_i is defined such that the diagonal values corresponding to the boundary nodes are zero, the nonzero eigenvalues of $D_iC_{ii}D_i$ correspond to the squared singular values of $A(:, \Omega_{Ii})$. Hence, all the eigenpairs of (3.6) corresponding to an infinite eigenvalue are included in the set of eigenpairs (v, λ) of (3.7) such that

$$\sigma^2_{\min}\left(A(:, \Omega_{Ii})\right) \leq \lambda s \leq \sigma^2_{\max}\left(A(:, \Omega_{Ii})\right), \tag{3.8}$$

  where $\sigma_{\min}\left(A(:, \Omega_{Ii})\right)$ and $\sigma_{\max}\left(A(:, \Omega_{Ii})\right)$ are the smallest and largest singular values of $A(:, \Omega_{Ii})$, respectively.

Therefore, choosing

$$s = O(\|\widetilde{C}_{ii}\|_2\varepsilon),$$

where ε is the machine precision, ensures $\widetilde{C}_{ii} + sI_{n_i}$ is numerically invertible and $s \ll 1$. Setting $s = \|\widetilde{C}_{ii}\|_2\varepsilon$ in (3.8), we obtain

$$\sigma^2_{\min}\left(A(:, \Omega_{Ii})\right) \leq \lambda\|\widetilde{C}_{ii}\|_2\varepsilon \leq \sigma^2_{\max}\left(A(:, \Omega_{Ii})\right).$$

By (3.5), we have

$$\|\widetilde{C}_{ii}\|_2 \leq \|C_{ii}\|_2,$$

and because $\Omega_{Ii} \subset \Omega_i$, it follows that

$$\|C_{ii}^{-1}\|_2 = \|\left(A(:, \Omega_i)^\top A(:, \Omega_i)\right)^{-1}\|_2 \leq \sigma^2_{\min}\left(A(:, \Omega_{Ii})\right).$$

Hence, if (v, λ) is an eigenpair of (3.7) with $v \in ker(\widetilde{C}_{ii}) \cap range(D_i C_{ii} D_i)$, then

$$(\kappa(C_{ii})\varepsilon)^{-1} \leq \lambda,$$

where $\kappa(C_{ii})$ is the condition number of C_{ii} and $\mathcal{Z}_i$ can be defined to be

$$\text{(3.9)} \quad \text{span}\left\{ v \mid D_i C_{ii} D_i v = \lambda(\widetilde{C}_{ii} + \varepsilon\|\widetilde{C}_{ii}\|_2 I_{n_i})v \text{ and } \lambda \geq \min\left(\frac{1}{\tau}, (\kappa(C_{ii})\varepsilon)^{-1}\right)\right\}.$$

Z_i is then taken to be the matrix whose columns are the vertical concatenation of corresponding eigenvectors.

Remark 3.3. Note that solving the generalized eigenvalue problem (3.7) by an iterative method such as Krylov–Schur [43] does not require the explicit form of C_{ii} and $\widetilde{C}_{ii}$. Rather, it requires solving linear systems of the form $(\widetilde{C}_{ii} + sI_{n_i})u = v$, together with matrix–vector products of the form $(\widetilde{C}_{ii} + sI_{n_i})v$ and $C_{ii}v$. It is clear that these products do not require the matrices $\widetilde{C}_{ii}$ and C_{ii} to be formed. Regarding the solution of the linear system $(\widetilde{C}_{ii} + sI_{n_i})u = v$, Remark 3.1 also applies to the Cholesky factorization of $\widetilde{C}_{ii} + sI_{n_i} = X^\top X$, where $X^\top = \begin{pmatrix} A(\Xi_i, \Omega_i)^\top & \sqrt{s} I_{n_i} \end{pmatrix}$, that can be computed by using a "thin" QR factorization of X.

From Remarks 3.1 and 3.3, and applying the same technique therein to factor $C_{00} = R_0 C R_0^\top = (AR_0^\top)^\top (AR_0^\top)$, we observe that given the overlapping partitions of A, the proposed two-level preconditioner can be constructed without forming the normal equations matrix. Algorithm 3.1 gives an overview of the steps for constructing our two-level Schwarz preconditioner for the normal equations matrix. The actual implementation of our proposed preconditioner will be discussed in greater detail in subsection 4.1.

Algorithm 3.1 Two-level Schwarz preconditioner for the normal equations matrix.

Input: matrix A, number of subdomains N, threshold τ to bound the condition number.

Output: two-level preconditioner M^{-1} for $C = A^\top A$.

1: $(\Omega_{I1}, \ldots, \Omega_{IN}) = \text{Partition}(A, N)$
2: **for** $i = 1$ to N in parallel **do**
3: $\quad \Xi_i = \text{FindNonzeroRows}(A(:, \Omega_{Ii}))$
4: $\quad \Omega_i = [\Omega_{Ii}, \Omega_{\Gamma i}] = \text{FindNonzeroColumns}(A(\Xi_i, :))$
5: $\quad$ Define D_i as in subsection 3.3.1 and R_i as in section 2
6: $\quad$ Perform Cholesky factorization of $C_{ii} = A(:, \Omega_i)^\top A(:, \Omega_i)$, see Remark 3.1
7: $\quad$ Perform Cholesky factorization of $\widetilde{C}_{ii} = A(\Xi_i, \Omega_i)^\top A(\Xi_i, \Omega_i)$, possibly using a small shift s, see Remark 3.3
8: $\quad$ Compute Z_i as defined in (3.9)
9: **end for**
10: Set $R_0^\top = \begin{bmatrix} R_1^\top D_1 Z_1, \ldots, R_N^\top D_N Z_N \end{bmatrix}$
11: Perform Cholesky factorization of $C_{00} = (AR_0^\top)^\top (AR_0^\top)$
12: Set $M^{-1} = M^{-1}_{\text{additive}} = \sum_{i=0}^{N} R_i^\top C_{ii}^{-1} R_i$ or M^{-1}_{balanced} (2.10) or M^{-1}_{deflated} (2.11)

4. Numerical experiments. In this section, we illustrate the effectiveness of the new two-level LS preconditioners M^{-1}_{balanced} and M^{-1}_{deflated}, their robustness with respect to the number of subdomains, and their efficiency in tackling large-scale sparse

TABLE 1
Test matrices taken from the SuiteSparse Matrix Collection.

Identifier	m	n	nnz(A)	nnz(C)	condest(C)
mesh_deform	234,023	9,393	853,829	117,117	$2.7 \cdot 10^6$
EternityII_E	262,144	11,077	1,503,732	1,109,181	$5.1 \cdot 10^{19}$
lp_stocfor3	23,541	16,675	72,721	223,395	$4.0 \cdot 10^{10}$
deltaX	68,600	21,961	247,424	2,623,073	$3.7 \cdot 10^{20}$
sc205-2r	62,423	35,213	123,239	12,984,043	$1.7 \cdot 10^7$
stormg2-125	172,431	65,935	433,256	1,953,519	∞
Rucci1	1,977,885	109,900	7,791,168	9,747,744	$2.0 \cdot 10^8$
image_interp	232,485	120,000	711,683	1,555,994	$4.7 \cdot 10^7$
mk13-b5	270,270	135,135	810,810	1,756,755	∞
pds-100	514,577	156,016	1,096,002	1,470,688	∞
fome21	267,596	216,350	465,294	640,240	∞
sgpf5y6	312,540	246,077	831,976	2,761,021	$6.0 \cdot 10^6$
Hardesty2	929,901	303,645	4,020,731	3,936,209	$1.2 \cdot 10^{10}$
Delor338K	450,807	343,236	4,211,599	44,723,076	$1.5 \cdot 10^7$
watson_2	677,224	352,013	1,846,391	3,390,279	$1.0 \cdot 10^7$
LargeRegFile	2,111,154	801,374	4,944,201	6,378,592	$3.0 \cdot 10^8$
cont11_l	1,961,394	1,468,599	5,382,999	18,064,261	$2.0 \cdot 10^{10}$

and ill-conditioned LS problems selected from the SuiteSparse Matrix Collection [13]. The test matrices are listed in Table 1. For each matrix, we report its dimensions, the number of entries in A and in the normal equations matrix C, and the condition number of C (estimated using the MATLAB function `condest`).

In subsection 4.1, we discuss our implementation based on the parallel backend [7]. In particular, we show that very little coding effort is needed to construct all the necessary algebraic tools, and that it is possible to take advantage of an existing package, such as HPDDM [27], to setup the new preconditioners efficiently. We then show in subsection 4.2 how M^{-1}_{balanced} and M^{-1}_{deflated} perform compared to other preconditioners when solving challenging LS problems. The preconditioners we consider are:

- limited memory incomplete Cholesky (IC) factorization specialized for the normal equations matrix as implemented in `HSL_MI35` from the HSL library [25] (note that this package is written in Fortran and we run it using the supplied MATLAB interface with default parameter settings);
- one-level overlapping Schwarz methods M^{-1}_{ASM} and M^{-1}_{RAS} as implemented in PETSc;
- algebraic multigrid methods as implemented both in BoomerAMG from the HYPRE library [20] and in GAMG [1] from PETSc.

Finally, in subsection 4.3, we study the strong scalability of M^{-1}_{balanced} and its robustness with respect to the number of subdomains by using a fixed problem and increasing the number of subdomains.

With the exception of the serial IC code `HSL_MI35`, all the numerical experiments are performed on Irène, a system composed of 2,292 nodes with two 64-core AMD Rome processors clocked at 2.6 GHz and, unless stated otherwise, 256 MPI processes are used. For the domain decomposition methods, one subdomain is assigned per process. All computations are performed in double-precision arithmetic.

In all our experiments, the vector b in (1.1) is generated randomly and the initial guess for the iterative solver is zero. When constructing our new two-level preconditioners, with the exception of the results presented in Figure 1, at most 300 eigenpairs are computed on each subdomain and the threshold parameter τ from (3.9) is set to 0.6. These parameters were found to provide good numerical performance after a very quick trial-and-error approach on a single problem. We did not want to adjust them for each problem from Table 1, but it will be shown next that they are fine overall without additional tuning.

4.1. Implementation aspects. The new two-level preconditioners are implemented on top of the well-known distributed memory library PETSc. This section is not aimed at PETSc specialists. Rather, we want to briefly explain what was needed to provide an efficient yet concise implementation. Our new code is open-source, available at https://github.com/prj-/aldaas2021robust. It comprises fewer than 150 lines of code (including the initialization and error analysis). The main source files, written in Fortran, C, and Python, have three major phases, which we now outline.

4.1.1. Loading and partitioning phase. First, PETSc is used to load the matrix A in parallel, following a contiguous one-dimensional row partitioning among MPI processes. We explicitly assemble the normal equations matrix using the routine MatTransposeMatMult [32]. The initial PETSc-enforced parallel decomposition of A among processes may not be appropriate for the normal equations, so ParMETIS is used by PETSc to repartition C. This also induces a permutation of the columns of A.

4.1.2. Setup phase. To ensure that the normal equations matrix C is definite and its Cholesky factorization is breakdown free, C is shifted by $10^{-10}\|C\|_F I_n$ (here and elsewhere, $\|\cdot\|_F$ denotes the Frobenius norm). Note that this is only needed for the construction of the preconditioner; the preconditioner is used to solve the original LS problem. Given the indices of the columns owned by a MPI process, we call the routine MatIncreaseOverlap on the normal equations matrix to build an extended set of column indices of A that will be used to define overlapping subdomains. These are the Ω_i as defined in (3.1). Using the routine MatFindNonzeroRows, this extended set of indices is used to concurrently find on each subdomain the set of nonzero rows. These are the sets Ξ_i as illustrated in (3.2) and (3.3). The subdomain matrices C_{ii} from (2.1) as well as the partition of unity D_i as illustrated in (3.4) are automatically assembled by PETSc when using domain decomposition preconditioners such as PCASM or PCHPDDM. The right-hand side matrices of the generalized eigenvalue problems (3.6) are assembled using MatTransposeMatMult, but note that this product is this time performed concurrently on each subdomain. The small shift s from (3.7) is set to $10^{-8}\|\widetilde{C}_{ii}\|_F$. These matrices and the sets of overlapping column indices are passed to PCHPDDM using routine PCHPDDMSetAuxiliaryMat. The rest of the setup is hidden from the user. It includes solving the generalized eigenvalue problems using SLEPc [24], followed by the assembly and redistribution of the second-level operator using a Galerkin product (2.2) (see [26] for more details on how this is performed efficiently in PCHPDDM).

4.1.3. Solution phase. For the solution phase, users can choose between multiple Krylov methods, including LSQR [35] and GMRES. We use left-preconditioned LSQR (see, for example, [6, Algorithm 2]) and right-preconditioned GMRES. Each iteration of LSQR requires matrix–vector products with A and $A^\top$. For

TABLE 2

Preconditioner comparison when running LSQR. Iteration counts are reported. M_{ASM}^{-1} and $M_{balanced}^{-1}$ are the one- and two-level overlapping Schwarz preconditioners, respectively. † denotes iteration count exceeds 1,000. *‡ denotes either a failure in computing the preconditioner because of memory issues or a breakdown of LSQR.*

Identifier	M_{balanced}^{-1}	M_{ASM}^{-1}	BoomerAMG	GAMG	HSL_MI35
mesh_deform	13	27	‡	35	**5**
EternityII_E	**43**	91	‡	63	199
lp_stocfor3	**34**	136	‡	513	211
deltaX	**23**	98	‡	784	640
sc205-2r	**54**	61	‡	195	97
stormg2-125	**42**	174	‡	†	†
Rucci1	**21**	484	118	364	†
image_interp	**11**	409	40	203	†
mk13-b5	19	21	**11**	‡	**11**
pds-100	18	202	**16**	35	110
fome21	20	104	**16**	20	41
sgpf5y6	224	264	‡	163	**110**
Hardesty2	**30**	913	88	404	†
Delor338K	**10**	11	‡	†	829
watson_2	**15**	109	‡	64	73
LargeRegFile	41	109	19	‡	**12**
cont11_l	**30**	490	53	723	‡

GMRES, instead of using the previously explicitly assembled normal equations matrix, we use an implicit representation of the operator that computes the matrix–vector product with A followed by the product with $A^\top$. The type of overlapping Schwarz method (additive or restricted additive) as well as the type of second-level correction (balanced or deflated) may be selected at runtime by the user. This flexibility is important because LSQR requires a symmetric preconditioner.

4.2. Numerical validation. In this section, we validate the effectiveness of the two-level method when compared to other preconditioners. Table 2 presents a comparison between five preconditioners: two-level additive Schwarz with balanced coarse correction M_{balanced}^{-1}, one-level additive Schwarz M_{ASM}^{-1}, BoomerAMG, GAMG, and HSL_MI35. The first level of the one- and two-level methods both use the additive Schwarz formulation; the second level uses the balanced deflation formulation (2.10). The results are for the iterative solver LSQR. If M denotes the preconditioner, LSQR terminates when the LS residual satisfies

$$\frac{\| \left(AM^{-1}\right)^\top (Ax - b)\|_2}{\|A\|_{M,F}\|Ax - b\|_2} < 10^{-8},$$

where $\|A\|_{M,F} = \sum_{i=1}^{n} \lambda_i(M^{-1}A^\top A)$ is the sum of the positive eigenvalues of $M^{-1}A^\top A$ that is approximated by LSQR itself. Note that if $M^{-1} = W^{-1}W^{-\top}$, then $\|A\|_{M,F} = \|AW^{-1}\|_F$.

It is clear that both the one- and two-level Schwarz methods are more robust than the other preconditioners as they encounter no breakdowns and solve all the LS problems using fewer than 1,000 iterations. Because HSL_MI35 is a sequential code that runs on a single core, there was not enough memory to compute the preconditioner

TABLE 3

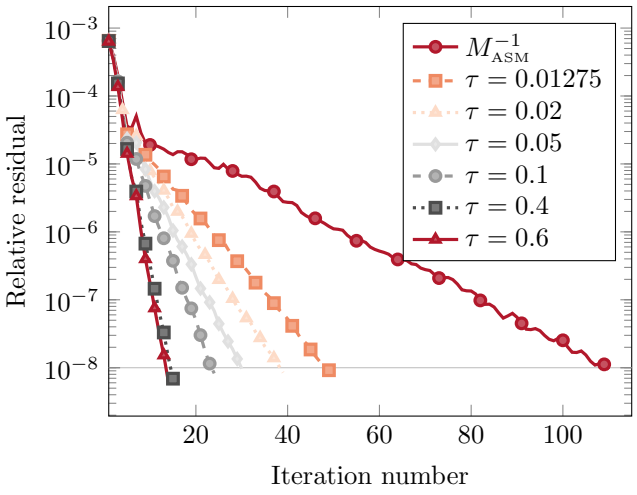
Preconditioner comparison when running GMRES. Iteration counts are reported. M_{RAS}^{-1} and $M_{deflated}^{-1}$ are the one- and two-level overlapping Schwarz preconditioners, respectively. † denotes iteration count exceeds 1,000. *‡ denotes either a failure in computing the preconditioner because of memory issues or a breakdown of GMRES.*

Identifier	M_{deflated}^{-1}	M_{RAS}^{-1}	BoomerAMG	GAMG	HSL_MI35
mesh_deform	6	27	21	50	**5**
EternityII_E	**5**	93	†	97	186
lp_stocfor3	**21**	†	†	†	198
deltaX	**6**	93	†	†	†
sc205-2r	**12**	125	†	490	69
stormg2-125	**23**	‡	‡	‡	†
Rucci1	**10**	958	213	882	†
image_interp	**10**	971	67	476	†
mk13-b5	14	18	21	‡	**12**
pds-100	**10**	84	23	51	115
fome21	**10**	55	22	29	41
sgpf5y6	116	†	†	249	**100**
Hardesty2	**26**	†	155	†	†
Delor338K	**5**	9	†	†	†
watson_2	**7**	134	252	96	73
LargeRegFile	**6**	21	23	‡	11
cont11_l	**45**	†	172	†	‡

for problem cont11_l. For many of the problems, the iteration count for HSL_MI35 can be reduced by increasing the parameters that determine the number of entries in the IC factor (the default values are rather small for the large test examples). LSQR preconditioned with BoomerAMG breaks down for several problems, as reported by PETSc error code KSP_DIVERGED_BREAKDOWN. GAMG is more robust but requires more iterations for problems where both algebraic multigrid solvers are successful. Note that even with more advanced options than the default ones set by PETSc, such as PMIS coarsening [14] with extended classical interpolation [15] for BoomerAMG or Schwarz smoothing for GAMG, these solvers do not perform considerably better numerically. We can also see that the two-level preconditioner outperforms the one-level preconditioner consistently.

Table 3 presents a similar comparison, but using right-preconditioned GMRES applied directly to the normal equations (1.2). A restart parameter of 100 is used. The relative tolerance is again set to 10^{-8}, but this now applies to the unpreconditioned residual. We switch from M_{ASM}^{-1} to M_{RAS}^{-1} (2.9), which is known to perform better numerically. For the two-level method, we switch from M_{balanced}^{-1} to M_{deflated}^{-1} (2.11). Switching from LSQR to GMRES can be beneficial for some preconditioners, e.g., BoomerAMG now converges in 21 iterations instead of breaking down for problem mesh_deform. But this is not always the case, e.g., HSL_MI35 applied to problem deltaX does not converge within the 1,000 iteration limit. The two-level method is the most robust approach, while the restricted additive Schwarz preconditioner struggles to solve some problems, either because of a breakdown (problem stormg2-125) or because of slow convergence (problems lp_stocfor3, sgpf5y6, Hardesty2, and cont11_l).

Recall that for the results in Tables 2 and 3, the two-level preconditioner was



τ	n_0	Iterations
0.01275	2,400	49
0.02	2,683	39
0.05	3,049	30
0.1	3,337	24
0.4	8,979	15
0.6	30,246	14

FIG. 1. *Influence of the threshold parameter τ on the convergence of preconditioned LSQR for problem watson_2 ($m = 677{,}224$ and $n = 352{,}013$).*

constructed using at most 300 eigenpairs and the threshold parameter τ was set to 0.6. Whilst this highlights that tuning τ for individual problems is not necessary to successfully solve a range of problems, it does not validate the ability of our preconditioner to concurrently select the most appropriate local eigenpairs to define an adaptive preconditioner. To that end, for problem watson_2, we consider the effect on the performance of our two-level preconditioner of varying τ. Results for LSQR with M_{ASM}^{-1} and M_{balanced}^{-1} are presented in Figure 1. Here, 512 MPI processes are used and the convergence tolerance is again 10^{-8}. We observe that the two-level method consistently outperforms the one-level method. Furthermore, as we increase τ, the iteration count reduces and the size n_0 of the second level increases. It is also interesting to highlight that the convergence is smooth even with a very small value $\tau = 0.01275$, $n_0 = 2{,}400$ compared to the dimension $3.52 \cdot 10^5$ of the normal equations matrix.

4.3. Performance study. We next investigate the algorithmic cost of the two-level method. To do so, we perform a strong scaling analysis using a large problem not presented in Table 1 but still from the SuiteSparse Matrix Collection, Hardesty3. The matrix is of dimension $8{,}217{,}820 \times 7{,}591{,}564$, and the number of nonzero entries in C is 98,634,426. In Table 4, we report the number of iterations as well as the eigensolve, setup, and solve times as the number N of subdomains ranges from 16 to 4,096. The times are obtained using the PETSc -log_view command line option. For different N, the reported times on each row of the table are the maximum among all processes. The setup time includes the numerical factorization of the first-level subdomain matrices, the assembly of the second-level operator and its factorization. Note that the symbolic factorization of the first-level subdomain is shared between the domain decomposition preconditioner and the eigensolver because we use the Krylov–Schur method as implemented in SLEPc, which requires the factorization of the right-hand side matrices from (3.7). The Cholesky factorizations of the subdomain matrices and of the second-level operator are performed using the sparse direct solver MUMPS [5]. For small numbers of subdomains ($N < 128$), the cost of the eigensolves are clearly prohibitive. By increasing the number of subdomains, thus reducing their

TABLE 4

Strong scaling for problem Hardesty3 (m = 8,217,820 and n = 7,591,564) for N ranging from 16 *to* 4,096 *subdomains. All times are in seconds. Column 2 reports the LSQR iteration count. Column 4 reports the setup time minus the concurrent solution time of the generalized eigenproblems, which is given in column 3.*

N	Iterations	Eigensolve	Setup	Solve	n_0	Total	Speedup
16	113	2,417.4	24.5	301.3	4,800	2,743.2	–
32	117	1,032.7	14.1	154.2	9,600	1,201.0	2.3
64	129	887.2	11.4	112.3	19,200	1,010.9	2.7
128	144	224.1	6.9	55.4	38,400	286.3	9.6
256	97	128.0	6.7	32.2	76,800	166.9	16.4
512	87	45.5	13.0	26.9	153,391	85.3	32.2
1,024	85	23.8	20.2	35.3	303,929	79.3	34.6
2,048	55	14.6	31.4	43.2	497,704	89.1	30.8
4,096	59	11.7	30.8	44.9	695,774	87.3	31.4

size, the time to construct the preconditioner becomes much more tractable and overall, our implementation yields good speedups on a wide range of process counts. Note that the threshold parameter $\tau = 0.6$ is not attained on any of the subdomains for N ranging from 16 up to 256, so that $n_0 = 300 \times N$. For larger N, $\tau = 0.6$ is attained, the preconditioner automatically selects the appropriate eigenmodes, and convergence improves (see column 2 of Table 4). When N is large ($N \geq$ 1,024), the setup and solve times are impacted by the high cost of factorizing and solving the second-level problems, which, as highlighted by the values of n_0, become large. Multilevel variants [4] could be used to overcome this but goes beyond the scope of the current study.

5. Concluding comments. Solving large-scale sparse linear least-squares problems is known to be challenging. Previously proposed preconditioners have generally been serial and have involved incomplete factorizations of A or $C = A^\top A$. In this paper, we have employed ideas that have been developed in the area of domain decomposition, which (as far as we are aware) have not previously been applied to least-squares problems. In particular, we have exploited recent work by Al Daas and Grigori [3] on algebraic domain decomposition preconditioners for SPD systems to propose a new two-level algebraic domain preconditioner for the normal equations matrix C. We have used the concept of an algebraic local SPSD splitting of an SPD matrix and we have shown that the structure of C as the product of $A^\top$ and A can be used to efficiently perform the splitting. Furthermore, we have proved that using the two-level preconditioner, the spectral condition number of the preconditioned normal equations matrix is bounded from above independently of the number of the subdomains and the size of the problem. Moreover, this upper bound depends on a parameter τ that can be chosen by the user to decrease (resp. increase) the upper bound with the costs of setting up the preconditioner being larger (resp. smaller).

The new two-level preconditioner has been implemented in parallel within PETSc. Numerical experiments on a range of problems from real applications have shown that whilst both one-level and two-level domain decomposition preconditioners are effective when used with LSQR to solve the normal equations, the latter consistently results in significantly faster convergence. It also outperforms other possible preconditioners, both in terms of robustness and iteration counts. Furthermore, our numerical experiments on a set of challenging least-squares problems show that the two-level

preconditioner is robust with respect to the parameter τ. Moreover, a strong scalability test of the two-level preconditioner assessed its robustness with respect to the number of subdomains.

Future work includes extending the approach to develop preconditioners for solving large sparse–dense least-squares problems in which A contains a small number of rows that have many more entries than the other rows. These cause the normal equations matrix to be dense and so they need to be handled separately (see, for example, the recent work of Scott and Tůma [40] and references therein). As already observed, we also plan to consider multilevel variants to allow the use of a larger number of subdomains and processes.

Acknowledgments. This work was granted access to the GENCI-sponsored HPC resources of TGCC@CEA under allocation A0090607519. The authors would like to thank L. Dalcin, V. Hapla, and T. Isaac for their recent contributions to PETSc that made the implementation of our preconditioner more flexible. Finally, we are grateful to two anonymous reviewers for their constructive feedback.

Code reproducibility. Interested readers are referred to https://github.com/prj-/aldaas2021robust/blob/main/README.md for setting up the appropriate requirements, compiling, and running our proposed preconditioner. Fortran, C, and Python source codes are provided.

REFERENCES

[1] M. F. Adams, H. H. Bayraktar, T. M. Keaveny, and P. Papadopoulos, *Ultrascalable implicit finite element analyses in solid mechanics with over a half a billion degrees of freedom*, in Proceedings of the 2004 ACM/IEEE Conference on Supercomputing, SC04, IEEE Computer Society, 2004, pp. 34:1–34:15.

[2] E. Agullo, A. Buttari, A. Guermouche, and F. Lopez, *Implementing multifrontal sparse solvers for multicore architectures with sequential task flow runtime systems*, ACM Transactions on Mathematical Software, 43 (2016), http://buttari.perso.enseeiht.fr/qr_mumps.

[3] H. Al Daas and L. Grigori, *A class of efficient locally constructed preconditioners based on coarse spaces*, SIAM Journal on Matrix Analysis and Applications, 40 (2019), pp. 66–91.

[4] H. Al Daas, L. Grigori, P. Jolivet, and P.-H. Tournier, *A multilevel Schwarz preconditioner based on a hierarchy of robust coarse spaces*, SIAM Journal on Scientific Computing, 43 (2021), pp. A1907–A1928.

[5] P. R. Amestoy, I. S. Duff, J.-Y. L'Excellent, and J. Koster, *A fully asynchronous multifrontal solver using distributed dynamic scheduling*, SIAM Journal on Matrix Analysis and Applications, 23 (2001), pp. 15–41, http://mumps.enseeiht.fr.

[6] S. R. Arridge, M. M. Betcke, and L. Harhanen, *Iterated preconditioned LSQR method for inverse problems on unstructured grids*, Inverse Problems, 30 (2014), p. 075009.

[7] S. Balay, S. Abhyankar, M. F. Adams, J. Brown, P. Brune, K. Buschelman, L. Dalcin, A. Dener, V. Eijkhout, W. D. Gropp, D. Karpeyev, D. Kaushik, M. G. Knepley, D. A. May, L. C. McInnes, R. T. Mills, T. Munson, K. Rupp, P. Sanan, B. F. Smith, S. Zampini, H. Zhang, and H. Zhang, *PETSc web page*, 2021, https://petsc.org.

[8] R. Bru, J. Marín, J. Mas, and M. Tůma, *Preconditioned iterative methods for solving linear least-squares problems*, SIAM Journal on Scientific Computing, 36 (2014), pp. A2002–A2022.

[9] X.-C. Cai and M. Sarkis, *A restricted additive Schwarz preconditioner for general sparse linear systems*, SIAM Journal on Scientific Computing, 21 (1999), pp. 792–797.

[10] T. F. Chan and T. P. Mathew, *Domain decomposition algorithms*, Acta Numerica, 3 (1994), pp. 61–143.

[11] X. Cui and K. Hayami, *Generalized approximate inverse preconditioners for least-squares problems*, Japan Journal of Industrial and Applied Mathematics, 26 (2009).

[12] T. A. Davis, *Algorithm 915, SuiteSparseQR: multifrontal multithreaded rank-revealing sparse QR factorization*, ACM Transactions on Mathematical Software, 38 (2011).

[13] T. A. Davis and Y. Hu, *The University of Florida sparse matrix collection*, ACM Transactions on Mathematical Software, 38 (2011), pp. 1–28.
[14] H. De Sterck, R. D. Falgout, J. W. Nolting, and U. M. Yang, *Distance-two interpolation for parallel algebraic multigrid*, Numerical Linear Algebra with Applications, 15 (2008), pp. 115–139.
[15] H. De Sterck, U. M. Yang, and J. J. Heys, *Reducing complexity in parallel algebraic multigrid preconditioners*, SIAM Journal on Matrix Analysis and Applications, 27 (2006), pp. 1019–1039.
[16] V. Dolean, P. Jolivet, and F. Nataf, *An introduction to domain decomposition methods. Algorithms, theory, and parallel implementation*, Society for Industrial and Applied Mathematics, 2015.
[17] I. S. Duff, R. Guivarch, D. Ruiz, and M. Zenadi, *The augmented block Cimmino distributed method*, SIAM Journal on Scientific Computing, 37 (2015), pp. A1248–A1269.
[18] A. Dumitraşc, P. Leleux, C. Popa, D. Ruiz, and S. Torun, *The augmented block Cimmino algorithm revisited*, 2018, https://arxiv.org/abs/1805.11487.
[19] T. Elfving, *Block-iterative methods for consistent and inconsistent linear equations*, Numerische Mathematik, 35 (1980), pp. 1–12.
[20] R. D. Falgout and U. M. Yang, hypre*: a library of high performance preconditioners*, Computational Science—ICCS 2002, (2002), pp. 632–641.
[21] M. J. Gander and A. Loneland, *SHEM: an optimal coarse space for RAS and its multiscale approximation*, in Domain Decomposition Methods in Science and Engineering XXIII, C.-O. Lee, X.-C. Cai, D. E. Keyes, H. H. Kim, A. Klawonn, E.-J. Park, and O. B. Widlund, eds., Cham, 2017, Springer International Publishing, pp. 313–321.
[22] N. I. M. Gould and J. A. Scott, *The state-of-the-art of preconditioners for sparse linear least-squares problems*, ACM Transactions on Mathematical Software, 43 (2017), pp. 36:1–35.
[23] A. Heinlein, C. Hochmuth, and A. Klawonn, *Reduced dimension GDSW coarse spaces for monolithic Schwarz domain decomposition methods for incompressible fluid flow problems*, International Journal for Numerical Methods in Engineering, 121 (2020), pp. 1101–1119.
[24] V. Hernandez, J. E. Roman, and V. Vidal, *SLEPc: a scalable and flexible toolkit for the solution of eigenvalue problems*, ACM Transactions on Mathematical Software, 31 (2005), pp. 351–362, https://slepc.upv.es.
[25] *HSL. A collection of Fortran codes for large-scale scientific computation*, 2018. http://www.hsl.rl.ac.uk.
[26] P. Jolivet, F. Hecht, F. Nataf, and C. Prud'homme, *Scalable domain decomposition preconditioners for heterogeneous elliptic problems*, in Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis, SC '13, New York, NY, USA, 2013, ACM, pp. 80:1–80:11.
[27] P. Jolivet, J. E. Roman, and S. Zampini, *KSPHPDDM and PCHPDDM: extending PETSc with advanced Krylov methods and robust multilevel overlapping Schwarz preconditioners*, Computers & Mathematics with Applications, 84 (2021), pp. 277–295.
[28] G. Karypis and V. Kumar, *Multilevel k-way partitioning scheme for irregular graphs*, Journal of Parallel and Distributed computing, 48 (1998), pp. 96–129.
[29] F. Kong and X.-C. Cai, *A scalable nonlinear fluid–structure interaction solver based on a Schwarz preconditioner with isogeometric unstructured coarse spaces in 3D*, Journal of Computational Physics, 340 (2017), pp. 498–518.
[30] N. Li and Y. Saad, *MIQR: a multilevel incomplete QR preconditioner for large sparse least-squares problems*, SIAM Journal on Matrix Analysis and Applications, 28 (2006), pp. 524–550.
[31] P. Marchand, X. Claeys, P. Jolivet, F. Nataf, and P.-H. Tournier, *Two-level preconditioning for h-version boundary element approximation of hypersingular operator with GenEO*, Numerische Mathematik, 146 (2020), pp. 597–628.
[32] M. McCourt, B. F. Smith, and H. Zhang, *Sparse matrix–matrix products executed through coloring*, SIAM Journal on Matrix Analysis and Applications, 36 (2015), pp. 90–109.
[33] *Intel MKL Sparse QR*, 2018.
[34] S. V. Nepomnyaschikh, *Mesh theorems of traces, normalizations of function traces and their inversions*, Russian Journal of Numerical Analysis and Mathematical Modelling, 6 (1991), pp. 1–25.
[35] C. C. Paige and M. A. Saunders, *LSQR: an algorithm for sparse linear equations and sparse least squares*, ACM Transactions on Mathematical Software, 8 (1982), p. 43–71.
[36] F. Pellegrini and J. Roman, *SCOTCH: a software package for static mapping by dual recursive bipartitioning of process and architecture graphs*, in High-Performance

758 Computing and Networking, Springer, 1996, pp. 493–498.
759 [37] Y. Saad and M. H. Schultz, *GMRES: a generalized minimal residual algorithm for solving*
760 *nonsymmetric linear systems*, SIAM Journal on Scientific and Statistical Computing, 7
761 (1986), pp. 856–869.
762 [38] J. A. Scott and M. Tůma, *Preconditioning of linear least squares by robust incomplete*
763 *factorization for implicitly held normal equations*, SIAM Journal on Scientific Computing,
764 38 (2016), pp. C603–C623.
765 [39] J. A. Scott and M. Tůma, *Solving mixed sparse–dense linear least-squares problems by*
766 *preconditioned iterative methods*, SIAM Journal on Scientific Computing, 39 (2017),
767 pp. A2422–A2437.
768 [40] J. A. Scott and M. Tůma, *Strengths and limitations of stretching for least-squares problems*
769 *with some dense rows*, ACM Transactions on Mathematical Software, 41 (2021), pp. 1:1—
770 1:25.
771 [41] B. F. Smith, P. E. Bjørstad, and W. D. Gropp, *Domain decomposition: parallel multilevel*
772 *methods for elliptic partial differential equations*, Cambridge University Press, 1996.
773 [42] N. Spillane, V. Dolean, P. Hauret, F. Nataf, C. Pechstein, and R. Scheichl, *Abstract*
774 *robust coarse spaces for systems of PDEs via generalized eigenproblems in the overlaps*,
775 Numerische Mathematik, 126 (2014), pp. 741–770.
776 [43] G. W. Stewart, *A Krylov–Schur algorithm for large eigenproblems*, SIAM Journal on Matrix
777 Analysis and Applications, 23 (2002), pp. 601–614.
778 [44] J. M. Tang, R. Nabben, C. Vuik, and Y. A. Erlangga, *Comparison of two-level*
779 *preconditioners derived from deflation, domain decomposition and multigrid methods*,
780 Journal of Scientific Computing, 39 (2009), pp. 340–370.
781 [45] J. Van lent, R. Scheichl, and I. G. Graham, *Energy-minimizing coarse spaces for two-level*
782 *Schwarz methods for multiscale PDEs*, Numerical Linear Algebra with Applications, 16
783 (2009), pp. 775–799.