# parGeMSLR: A Parallel Multilevel Schur Complement Low-Rank Preconditioning and Solution Package for General Sparse Matrices

Tianshi Xu<sup>a,\*</sup>, Vassilis Kalantzis<sup>b</sup>, Ruipeng Li<sup>c</sup>, Yuanzhe Xi<sup>d</sup>, Geoffrey Dillon<sup>e</sup>, Yousef Saad<sup>a</sup>

<sup>a</sup>Department of Computer Science and Engineering, University of Minnesota, Minneapolis, MN 55455
 <sup>b</sup>Thomas J. Watson Research Center, IBM Research, Yorktown Heights, NY 10598
 <sup>c</sup>Center for Applied Scientific Computing, Lawrence Livermore National Laboratory, P.O. Box 808, L-561, Livermore, CA 94551
 <sup>d</sup>Department of Mathematics, Emory University, Atlanta, GA 30322
 <sup>e</sup>Department of Mathematics, University of South Carolina, Columbia, SC 29208

#### **Abstract**

We describe the "parallel Generalized Multilevel Schur complement Low-Rank preconditioner (parGeMSLR)", a C++ software library for the solution of (non-)Hermitian systems of linear algebraic equations via preconditioned Krylov subspace methods in distributed memory computing environments. The preconditioner implemented in parGeMSLR is based on algebraic domain decomposition, where the adjacency graph associated with a symmetrized coefficient matrix is partitioned recursively into several non-overlapping partitions via a *p*-way vertex separator. parGeMSLR offers several approaches to build a Schur complement approximate inverse preconditioner as the sum between the matrix inverse of the interface coupling matrix and a low-rank correction term. To reduce the cost associated with the computation of the approximate inverse matrices, parGeMSLR exploits a multilevel partitioning of the algebraic domain. The parGeMSLR library is implemented on top of the Message Passing Interface and can solve both real and complex linear systems. Furthermore, parGeMSLR can take advantage of hybrid computing environments with in-node access to one or more CPUs/GPUs. The parallel efficiency (weak and strong scaling) of parGeMSLR is demonstrated on a few model problems arising from discretizations of 3D Partial Differential Equations.

*Keywords:* Schur complement, low-rank correction, distributed memory preconditioner, sparse non-Hermitian linear systems, Graphics Processing Units

### 1. Introduction

This paper presents a distributed memory library for the iterative solution of systems of linear algebraic equations of the form

$$Ax = b, (1)$$

where the matrix  $A \in \mathbb{C}^{n \times n}$  is large, sparse, and (non-)Hermitian. Problems of this form typically originate from the discretization of a Partial Differential Equation in 2D or 3D domains. Existing methods to solve problems such as the one in (1) are generally divided into two main classes: direct solvers, and iterative solvers. For large, 3D problems, direct methods can be impractical due to the large computational and memory requirements. On the other hand, iterative approaches solve (1) by a preconditioned Krylov subspace iterative methods [1, 2], e.g., preconditioned Conjugate Gradient [1], if A is Hermitian and positive-definite, or GMRES [3] if A is non-Hermitian. The role of the preconditioner is to cluster the eigenvalues in an effort to accelerate the convergence of Krylov subspace method. For example, an efficient right preconditioner M transforms (1)

into the preconditioned system  $AM^{-1}(Mx) = b$ , where  $M^{-1}$  is an easy-to-apply operator such that the spectrum of  $AM^{-1}$  is included in a small bounded region, e.g., a disk or an ellipse, that does not contain the origin. An additional requirement is that the setup and application of the operator  $M^{-1}$  should be easily parallelizable.

Similarly to Krylov subspace methods, algebraic multigrid (AMG) methods are another widely-used class of iterative solvers [4]. AMG uses the idea of interpolation and restriction to build multilevel frameworks recursively in order to eliminate the smooth components in error efficiently. AMG is provably optimal for Poisson-like problems on regular meshes where the number of iterations to achieve convergence almost stays constant as the problem size increases. This property leads to appealing weak scaling results of AMG in parallel computing [5, 6, 7]. However, AMG can fail when solving indefinite problems or dealing with irregular meshes. It is worth mentioning that AMG can also be used as a preconditioner in the context of Krylov subspace methods.

For general sparse linear systems, a well-known class of general-purpose preconditioners is that of Incomplete LU (ILU) factorization preconditioners [8, 9, 1]. Here, the matrix A is approximately factored as  $A \approx LU$  where L is lower triangular and U is upper triangular, and the preconditioner is defined as M = LU. Applying  $M^{-1}$  then consists of two triangular

Preprint submitted to Elsevier May 4, 2022

<sup>\*</sup>The work of the first and the last author was supported by the National Science Foundation (NSF) grant DMS-1912048. The work of the third author was supported by the NSF grant OAC-2003720. This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344 (LLNL-JRNL-830724)

solves since  $M^{-1} = U^{-1}L^{-1}$ . ILU methods can be applied to a greater selection of problems than AMG, including indefinite problems such as discretized Helmholtz equations [10, 11], and their robustness can be improved by modified/shifted ILU strategies [12, 13, 14]. On the other hand, the scalability of ILU preconditioned Krylov subspace methods is not as good as those of AMG. In particular, even for Poisson-like problems the number of iterations to achieve convergence with ILU preconditioned Krylov subspace methods will still increase as the size of matrices increases. In addition, the sequential nature of ILU-based preconditioners further limits their applications to large-scale problems, especially on distributed memory systems. Thus, most recent developments have been focused on improving their parallel efficiency [15, 16, 17].

The parallel efficiency of ILU-based preconditioners can be enhanced by domain decomposition (DD), where the original problem is decomposed into several subdomains which correspond to different blocks of rows of the coefficient matrix A. The most basic DD-based ILU approach is the block-Jacobi ILU preconditioner, where a local ILU is performed on each local submatrix. Since this method ignores all of the off-diagonal matrices corresponding to inter-domain couplings, its convergence rate tends to become slower as the number of subdomains increases. Several strategies have been proposed to handle the inter-domain couplings in order to improve the convergence rate. Restricted Additive Schwarz (RAS) methods expand the local matrix by a certain level to gain a faster convergence rate at the cost of losing some memory scalability [18]. Global factorization ILU methods factorize local rows corresponding to interior unknowns first, after which a global factorization of the couplings matrix is applied based on some graph algorithms [19, 20]. Another strategy is based on the low-rank approximation techniques [21]. These methods use partial ILU techniques with dropping [22, 23], incomplete triangular solve [24], and low-rank approximation [25] to form the Schur complement system and can be generalized into multilevel ILU approaches [22, 23, 25].

Other available preconditioning strategies include the (factorized) sparse approximate inverse preconditioners [26, 27, 28, 29, 30], polynomial preconditioners [31], and rank-structured preconditioners [32, 33, 34, 35]; see also [36] for a distributed memory hierarchical solver. Some of the these techniques can be further compounded with AMG, as "smoothers", or ILU-based preconditioners. For example, a combination of SLR [37] and polynomial preconditioning is discussed in [31].

This paper describes the implementation of a parallel preconditioner, termed<sup>1</sup> parGeMSLR, for the iterative solution of general systems of linear algebraic equations in distributed memory environments. parGeMSLR<sup>2</sup> is written in C++, and communication among different processor groups is achieved by means of the Message Passing Interface standard (MPI). The numerical approach followed in parGeMSLR is based on the Generalized Multilevel Schur complement Low-Rank (GeM-SLR) algorithm described in [25]. GeMSLR applies a multilevel partitioning of the algebraic domain, and the variables associated with each level are divided into either interior or interface variables. The multilevel structure is built by applying a p-way graph partitioner to partition the induced subgraph associated with the interface variables of the preceding level. Once the multilevel partitioning is completed, GeMSLR creates a separate Schur complement approximate inverse preconditioner at each level. Each preconditioner is the sum of two terms, with the first term being an approximate inverse of the interface coupling matrix, and the second term being a lowrank correction which aims at bridging the gap between the first term and the inverse of the Schur complement matrix associated with that level. Below, we summarize the main features of the parGeMSLR library:

- Scalability. parGeMSLR extends the capabilities of low-rank-based preconditioners, such as GeMSLR, by recursively partitioning the algebraic domain into levels which have the same number of partitions (or an integer multiple) as the number of MPI processes. Thus, no MPI process is left idle as the level counter increases. In turn, this leads to enhanced scalability when running on distributed memory environments.
- 2. Robustness and complex arithmetic. In contrast to ILU-based preconditioners, the numerical method implemented in parGeMSLR is less sensitive to indefiniteness and can be updated on-the-fly without discarding previous computational efforts. Additionally, parGeMSLR supports complex arithmetic and thus can be utilized to solve complex linear systems such as those originating from the discretization of Helmholtz equations.
- 3. Hybrid hardware acceleration. GPU acceleration is now supported in several iterative solver libraries to speed-up the application of preconditioners such as AMG or ILU, e.g., hypre [38], PARALUTION [39], and HIFLOW [40]. Similarly, parGeMSLR can take advantage of GPU accelerator hardware to offload any computation the user provides an interface for. The current release of parGeMSLR provides a CUDA interface to use one or more GPUs during the application of the (dense) low-rank correction part of the GeMSLR preconditioner.

This paper is organized as follows. Section 2 discusses low-rank correction preconditioners and provides an algorithmic description of parGeMSLR. Section 3 provides details on the multilevel reordering used by parGeMSLR. Section 4 presents indepth discussion and details related to the implementation and parallel performance aspects of parGeMSLR. Section 5 demonstrates the performance of parGeMSLR on distributed memory environments. Finally, concluding remarks are made in Section 6.

<sup>&</sup>lt;sup>1</sup>The abbreviation of the library is derived by the complete name "parallel Generalized multilevel Schur complement Low-Rank preconditioner"

<sup>&</sup>lt;sup>2</sup>The source code can be found in https://github.com/Hitenze/pargemslr

# 2. Schur complement approximate inverse preconditioners via low-rank corrections

This section discussed the main idea behind (multilevel) Schur complement preconditioners enhanced by low-rank corrections, e.g., see [41, 37, 25, 42].

#### 2.1. The Schur complement viewpoint

Let the linear system Ax = b be permuted as

$$A_0 x = PAP(P^T x) = Pb, (2)$$

where *P* is an  $n \times n$  permutation matrix such that

$$A_0 = \begin{bmatrix} B & F \\ E & C \end{bmatrix} = \begin{bmatrix} B^{(1)} & & & & F^{(1)} \\ & B^{(2)} & & & F^{(2)} \\ & & \ddots & & \vdots \\ & & & B^{(p)} & F^{(p)} \\ E^{(1)} & E^{(2)} & \cdots & E^{(p)} & C \end{bmatrix},$$

and the matrices  $B^{(i)}$ ,  $F^{(i)}$ , and  $E^{(i)}$  are of size  $d_i \times d_i$ ,  $d_i \times s_i$ , and  $s_i \times d_i$ , respectively. The matrix C is of size  $s \times s$ , and the matrix partitioning satisfies  $d + s = \sum_{j=1}^{j=p} d_j + s_j = n$ . Such matrix permutations can be computed by partitioning the adjacency graph of the matrix  $|A| + |A^T|$  into  $n \in \mathbb{N}$  non-overlapping partitions and

tations can be computed by partitioning the adjacency graph of the matrix  $|A| + |A^T|$  into  $p \in \mathbb{N}$  non-overlapping partitions and reordering the unknowns/equations such that the variables associated with the d interior nodes across all partitions are ordered before the variables associated with the s interface nodes.

Following the above notation, the linear system in (2) can be written in a block form

$$\begin{bmatrix} B & F \\ E & C \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} f \\ g \end{bmatrix}, \tag{3}$$

where  $u, f \in \mathbb{R}^d$  and  $v, g \in \mathbb{R}^s$ . Once the solution in (3) is computed, the solution x of the original, non-permuted system of linear algebraic equations Ax = b can be obtained by the inverse permutation  $x = P \begin{bmatrix} u \\ v \end{bmatrix}$ . Throughout the rest of this section we focus on the solution of the system in (3).

Following a block-LDU factorization of the matrix  $A_0$ , the permuted linear system in (2) can be written as

$$\begin{bmatrix} I \\ EB^{-1} & I \end{bmatrix} \begin{bmatrix} B & \\ & S \end{bmatrix} \begin{bmatrix} I & B^{-1}F \\ & I \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} f \\ g \end{bmatrix},$$

where  $S = C - EB^{-1}F$  denotes the *Schur complement matrix*. The solution of (3) is then equal to

$$\begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} I & -B^{-1}F \\ & I \end{bmatrix} \begin{bmatrix} B^{-1} & \\ & S^{-1} \end{bmatrix} \begin{bmatrix} I \\ -EB^{-1} & I \end{bmatrix} \begin{bmatrix} f \\ g \end{bmatrix},$$

which requires: a) the solution of two linear systems with the block-diagonal matrix B, and b) the solution of one linear system with the Schur complement matrix S. Note that since the matrix B is block-diagonal, the associated linear systems are decoupled into p independent systems of linear algebraic equations. Assuming a distributed memory computing environment

with p separate processor groups, each system of linear algebraic equations can be solved in parallel by means of applying a direct solver locally in each separate process.

In several real-world applications, e.g., those involving the discretization of PDEs on three-dimensional domains, solving the systems of linear algebraic equations with matrices B and S through a direct solver is generally impractical, primarily due to the large computational and memory cost associated with forming and factorizing the Schur complement matrix. An alternative then is to solve the linear systems with matrices B and S inexactly. For example, the solution of linear systems with matrix B can be computed approximately by replacing its exact LU factorization with an incomplete threshold LU (ILUT) [8]. Likewise, the exact Schur complement can be sparsified by discarding entries below a certain threshold value or located outside a pre-determined pattern. Linear systems with the sparsified Schur complement can be then solved approximately by obtaining an ILUT or other incomplete factorization [23, 43]. From a linear algebraic perspective, an ILUT of matrix B replaces the latter by an approximation  $\widehat{B}$  that is equal to the product of the incomplete lower and upper triangular factors. Similarly, the approximate Schur complement can be written as  $S = C - EB^{-1}F$ . The approximate factorizations of the matrices B and S can be combined to form an approximate LDU factorization in (2.1) which can be then used as a preconditioner in a Krylov subspace iterative solver such as GMRES.

### 2.2. Schur complements and low-rank corrections

One of the main drawbacks associated with incomplete factorizations is that they can not be easily updated if one needs a more accurate preconditioner. Moreover, their robustness can be limited when the matrix *A* is indefinite. For such scenarios, it has been advocated to add a *low-rank correction term* to enhance the efficiency of the Schur complement preconditioner, without discarding the previously computed incomplete factorizations. The low-rank enhancement implemented in parGeMSLR follows the GeMSLR multilevel preconditioner [25], a non-Hermitian extension of [37, 42]. Other approaches based on low-rank corrections can be found in [41, 44].

The GeMSLR preconditioner expresses the Schur complement matrix as

$$S = (I - EB^{-1}FC^{-1})C = (I - G)C,$$
 (4)

where  $G = EB^{-1}FC^{-1}$ . Perform a complex Schur decomposition of G

$$G = EB^{-1}FC^{-1} = WRW^{H}, (5)$$

where the  $s \times s$  matrix W is unitary and the  $s \times s$  matrix R is upper-triangular such that its diagonal entries contain the eigenvalues of matrix G. Plugging (5) in (4) gives

$$S = (I - WRW^{H})C = W(I - R)W^{H}C,$$

from which we can write the inverse of the Schur complement matrix as (Sherman-Morrison-Woodbury formula):

$$S^{-1} = C^{-1} + C^{-1}W[(I - R)^{-1} - I]W^{H}.$$
 (6)

Following (6), a system of linear equations with the Schur complement matrix requires the solution of a system of linear equations with matrix C, as well as matrix-vector multiplications and triangular matrix inversions with matrices  $W/W^H$  and  $(I - R)^{-1}$ , respectively. The product of matrices  $W[(I - R)^{-1} - I]W^H$  is a Schur decomposition by itself, with corresponding eigenvalues  $\gamma_i/(1 - \gamma_i)$ ,  $i = 1, \ldots, s$ , where  $\gamma_i$  denotes the i-th eigenvalue of the matrix G. Therefore, as long as the eigenvalues of the latter matrix are not located close to one, the matrix  $C(S^{-1} - C^{-1}) = W[(I - R)^{-1} - I]W^H$  can be approximated by a low-rank matrix, i.e.,  $S^{-1}$  is approximately equal to  $C^{-1}$  plus some low-rank correction.

The expression in (6) can be transformed into a practical preconditioner if the matrix  $W[(I-R)^{-1}-I]W^H$  is replaced by a rank-k approximation, where  $k \in \mathbb{N}$  is generally a user-given parameter. More specifically, let  $W_k$  denote the  $s \times k$  matrix which holds the leading k Schur vectors of matrix G, and let  $R_k$  denote the  $k \times k$  leading principal submatrix of matrix R. Then, a GeMSLR approximate inverse preconditioner is equal to<sup>3</sup>

$$M^{-1} = C^{-1} + C^{-1} W_k [(I - R_k)^{-1} - I] W_k^H \approx S^{-1}.$$
 (7

An approximate rank-k Schur decomposition of matrix G can be computed by the Krylov-Schur iterative algorithm [45]. The main idea is to iteratively expand and truncate an m-step Arnoldi relation with the matrix G,

$$GV_m = V_m H_m + \beta_m v_{m+1} e_m^H,$$

where  $\beta_m \in \mathbb{R}$ ,  $e_m$  denotes the first column of the  $m \times m$  identity matrix,  $[V_m, v_{m+1}]^H[V_m, v_{m+1}] = I$ , and  $H_m$  is either upper-Hessenberg (Arnoldi) or quasi-triangular (Schur-Krylov). Ideally, we would like to compute the eigenvectors/Schur vectors associated with eigenvalues located the closest to one, which, in the case of positive definite problems, is straightforward. Nonetheless, for indefinite problems, the eigenvalues located the closest to one will be interior eigenvalues. Our default strategy is to always compute eigenvectors/Schur vectors associated with eigenvalues located on the periphery of the spectrum of matrix G.

#### 2.3. Computations with an incomplete factorization of B

For large-scale problems, computing an exact factorization of the block-diagonal matrix B can be quite expensive. Instead, what is typically available is an ILUT factorization  $LU \approx B$ . Therefore, instead of computing an approximate rank-k Schur decomposition of matrix G, in practice we approximate a truncated Schur decomposition of the matrix  $\widehat{G} = E(U^{-1}L^{-1})FC^{-1}$ . Let then

$$\widehat{G}\widehat{V}_m = \widehat{V}_m \widehat{H}_m + \beta_m \widehat{v}_{m+1} e_m^H,$$

where  $[\widehat{V}_m, \widehat{v}_{m+1}]^H [\widehat{V}_m, \widehat{v}_{m+1}] = I$ , and  $\widehat{H}_m$  is upper-Hessenberg, denote an m-length Arnoldi relation obtained with matrix  $\widehat{G}$ . Moreover, let  $\widehat{H}_m = QTQ^H$  denote the complex Schur decomposition of matrix  $\widehat{H}_m$ . The low-rank correction term used in

GeMSLR is of the form  $\widehat{W}_k[(I-\widehat{R}_k)^{-1}-I]\widehat{W}_k^H$ , where  $T_k \in \mathbb{R}^{k \times k}$  denotes the  $k \times k$  leading principal submatrix of matrix T, and  $\widehat{W}_k = \widehat{V}_m Q_k$ , where  $Q_k \in \mathbb{R}^{s \times k}$  denotes the matrix holding the k leading Schur vectors of matrix  $\widehat{H}_m$ .

#### 2.4. Multilevel extensions

For large-scale, high-dimensional problems, the application of the matrix  $C^{-1}$  by means of an LU factorization of matrix C can still be expensive; especially when the value of p is too large, leading to large vertex separators. The idea suggested in [42, 21], and employed by GeMSLR, is to take advantage of the purely algebraic formulation developed in the previous section and apply  $C^{-1}$  inexactly by using the Schur complement low-rank preconditioner described in the previous section. In fact, this approach can be repeated more than once, leading to a multilevel preconditioner.

More specifically, let  $l_{ev} \in \mathbb{N}$  denote the number of levels, and define the sequence of matrices

$$A_{l} = P_{l-1}C_{l-1}P_{l-1} = \begin{bmatrix} B_{l} & F_{l} \\ E_{l} & C_{l} \end{bmatrix}, \quad C_{-1} = A, \quad l = 0, 1, \dots, l_{ev} - 1,$$
(8)

where the matrix  $B_l$  is block-diagonal with p on-diagonal matrix blocks. The  $2 \times 2$  block matrix partition of each matrix  $A_l$  is obtained by partitioning the adjacency graph of the matrix  $|C_{l-1}| + |C_{l-1}^T|$  into p non-overlapping partitions and reordering the unknowns/equations such that the variables associated with the interior nodes across all partitions are ordered before the variables associated with the interface nodes of the adjacency graph. The matrix  $C_{l-1}$  is then permuted in-place through the  $s_{l-1} \times s_{l-1}$  permutation matrix  $P_{l-1}$ , where  $s_{l-1}$  denotes the size of the matrix  $C_{l-1}$ .

The solution of a system of linear algebraic equations with matrix  $A_l$  as the coefficient matrix and  $\begin{bmatrix} f_l^T & g_l^T \end{bmatrix}^T$  as the right-hand side, can be computed as

$$\begin{bmatrix} u_l \\ v_l \end{bmatrix} = \begin{bmatrix} I & -B_l^{-1} F_l \\ I \end{bmatrix} \begin{bmatrix} B_l^{-1} & \\ & S_l^{-1} \end{bmatrix} \begin{bmatrix} I \\ -E_l B_l^{-1} & I \end{bmatrix} \begin{bmatrix} f_l \\ g_l \end{bmatrix},$$

where  $S_l = C_l - E_l B_l^{-1} F_l$  denotes the  $s_l \times s_l$  Schur complement matrix associated with the l-th level, where  $s_l \in \mathbb{N}$  denotes the size of the matrix  $C_l$ . Instead of computing the exact LU factorizations of matrices  $B_l$  and  $S_l$ , the preconditioner implemented in the parGeMSLR library substitutes  $B_l^{-1} \approx (L_l U_l)^{-1}$ , where  $L_l U_l$  denotes an ILUT factorization of matrix  $B_l$ , and

$$S_l^{-1} \approx C_l^{-1} + C_l^{-1} W_{l,k} [(I - R_{l,k})^{-1} - I] W_{l,k}^H,$$
 (9)

where  $\widehat{W}_{l,k}$  denotes the matrix which holds the approximate leading k Schur vectors of the matrix  $\widehat{G}_l = E_l^T U_l^{-1} L_l^{-1} F_l C_l^{-1}$ , and  $\widehat{R}_{l,k}$  denotes the approximation of the  $k \times k$  leading principal submatrix of the matrix  $\widehat{R}_l$  that satisfis the Schur decomposition  $\widehat{G}_l = \widehat{W}_l \widehat{R}_l \widehat{W}_l^H$ . Algorithm 2 summarizes the above discussion ("setup phase") in the form of an algorithm. Notice that the recursion stops at level  $l_{ev} - 1$ , and an ILUT of the matrix  $C_{l_{ev}-1}$  is computed explicitly.

<sup>&</sup>lt;sup>3</sup>Note that the  $k \times k$  leading principal submatrix of  $(I - R)^{-1}$  is equal to  $(I - R_k)^{-1}$ .

Algorithm 1 outlines the procedure associated with the application of the GeMSLR preconditioner ("solve phase"). At each level, the preconditioning step consists of a forward and backward substitution with the ILUT triangular factors of  $B_l$ , followed by the application of the rank-k correction term. When  $l = l_{ev} - 1$ , there is no low-rank correction term applied, since this is the last level. Moreover, when l = 0 (root level), it is possible to enhance the GeMSLR preconditioner by applying a few steps of right preconditioned GMRES. Note though that these iterations are performed with the inexact Schur complement  $\widehat{S}_l = C_l - E_l(U^{-1}L^{-1})F_l$ .

# Algorithm 1 Standard Parallel GeMSLR Solve

```
1: procedure PGEMSLRSolve(b, l)
                                                                                  \triangleright Solve for q with
      right-hand-side b at level l
             Apply reordering \begin{bmatrix} b_1 \\ b_2 \end{bmatrix} = P_{l-1}b.

Solve z_1 = U_l^{-1}L_l^{-1}b_1.

Compute z_2 = b_2 - E_l z_1.
 2:
 3:
 4:
             if l = 0 then
 5:
                    Solve \widehat{S}_1 y_2 = z_2 by right preconditioned GMRES.
 6:
 7:
                    Compute u_2 = \widehat{W}_{l,k}[(I - \widehat{R}_{l,k})^{-1} - I]\widehat{W}_{l,k}^H z_2
Call y_2 = \text{pGeMSLRSolve}(u_2 + z_2, l + 1).
 8:
 9:
10:
             Compute y_1 = z_1 - U_l^{-1} L_l^{-1} F_l y_2.
11:
             Apply reordering q = P_{l-1} \begin{bmatrix} y_1 \\ y_2 \end{bmatrix}.
12:
             return x
13:
      end procedure
```

# Algorithm 2 Parallel GeMSLR Setup

```
1: procedure PGEMSLRSETUP(A, l_{ev}) > Setup the parallel GeMSLR preconditioner
2: Generate l_{ev}-level structure by Algorithm 3.
3: for l from 0 to l_{ev} - 1 do
4: Compute ILU factorization L_lU_l \approx B_l.
5: Compute matrices \widehat{W}_{l,k} and \widehat{R}_{l,k}.
6: If l = l_{ev} - 1, compute an ILUT factorization L_{l_{ev}-1}U_{l_{ev}-1} \approx C_{l_{ev}-1}; exit.
7: end for
8: end procedure
```

#### 3. Multilevel reordering

This section outlines the multilevel reordering approach implemented in the parGeMSLR library. For simplicity, we focus on symmetric reorderings obtained by applying a p-way vertex separator to the adjacency graph associated with the matrices  $|C_{l-1}| + |C_{l-1}^T|$ ,  $l = 0, \ldots, l_{ev} - 1$ ,  $C_{-1} = A$ , [46, 47, 48, 49]. In particular, given a graph G = (V, E), a p-way vertex separator computes a separator  $S \subset V$  and p non-overlapping (disjoint) sets  $V_1, \ldots, V_p \subset V$  such that  $V_1 \cup \ldots \cup V_p \cup S = V$  and there are no edges connecting the sets  $V_i$  and  $V_i$  when  $i \neq j$ .

# Algorithm 3 Parallel GeMSLR Reordering

# 3.1. Hierarchical Interface Decomposition

The GeMSLR preconditioner relies on a Hierarchical Interface Decomposition (HID) [50] to reduce the setup cost of the ILU and low-rank correction parts associated with the setup phase of the preconditioner. The main idea behind HID is to partition the adjacency graph of  $|A| + |A^T|$  into  $2^{l_{ev}}$  partitions via nested dissection with a recursion depth of  $l_{ev}$ . The vertex separators at level l are disjoint with each other since they are divided by vertex separators from higher levels. When ordered by levels, the global permutation of matrix A will have a block-diagonal structure with  $2^{l_{ev}-l}$  blocks at level  $0 \le l \le l_{ev} - 1$ , i.e., the number of diagonal blocks at each level reduce by a factor of two.

#### 3.2. Multilevel partitioning through p-way vertex separators

In contrast to previous low-rank correction preconditioners such as MSLR and GeMSLR [42, 25], the main goal of parGeMSLR is to sustain good parallel efficiency, and thus HID is not appropriate.<sup>4</sup> Instead, the default approach in parGeMSLR is to partition the adjacency graph by a multilevel partitioner where each level consists of *p* partitions and a vertex separator. In contrast to HID, where there is only a vertex separator at each level, the new idea is to apply the same *p*-way partitioner recursively on the vertex separator associated with each new level. The main advantage of this multilevel reordering scheme is that it leads to a *fixed number of p partitions at each level*, and thus load balancing is generally much better than that obtained using HID.

A high-level description can be found in Algorithm 3. At the root level (l=0), the graph associated with the matrix  $|A|+|A^T|$ , is partitioned into p subdomains with a p-way vertex separator, resulting to p non-overlapping connected components and their associated vertex separator. The multilevel partitioner then proceeds to the next level, l=1, and applies the p-way vertex partitioner to the induced subgraph associated with the vertex separator at level l=0. This leads to a second set of p non-overlapping connected components and a new, albeit smaller vertex separator. The p-way vertex partitioner is then applied again to the induced subgraph associated with the vertex separator obtained at level l=1, etc. The procedure continues until either level  $l_{ev}-1$  is reached, or the vertex separator

<sup>&</sup>lt;sup>4</sup>Nonetheless, HID is offered in parGeMSLR.

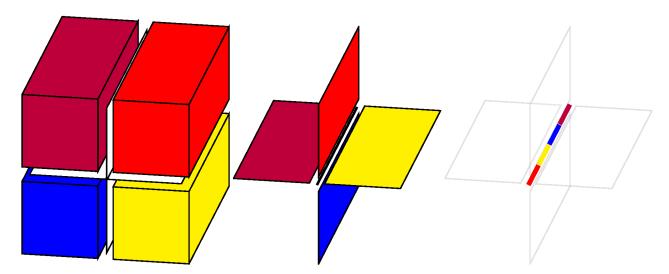


Figure 1: Left: a three-dimensional domain partitioned into p = 4 subdomains. The vertex separator consists of four faces, with each face located between neighboring subdomains. Center: partitioning of the root-level separator into p = 4 subdomains. Right: partitioning of the vertex separator at the second level.

at the current level l has so few vertices that it can not be further partitioned into p non-overlapping partitions.

An illustration of a three-level, four-way partitioner applied to a three-dimensional algebraic domain (a unit cube) is shown in Figure 1. The leftmost subfigure shows the p=4 separate partitions obtained by the application of the four-way vertex partitioner as well as the vertex separator itself (shown in white color) at level l=0. This vertex separator, which consists of four two-dimensional faces, forms the algebraic object to be partitioned at level l=1, and the partitioning is shown in the middle subfigure, where this time the vertex separator is a one-dimensional object. Finally, at level l=2, the most recent vertex separator is further partitioned into four independent partitions, leading to a new vertex separator which consists of only three vertices; see the rightmost subfigure.

In addition to the above illustration, Figure 2 plots the sparsity pattern of a Finite Difference discretization of the Laplace operator on a three-dimensional domain, after reordering its rows and columns according to a p-way, multilevel reordering with  $l_{ev}=4$  and p=4 (left). A zoom-in of the submatrix associated with the permutation of the vertex separators is also shown (right). Note that in this particular example, the last level has already too few variables to be partitioned any further. In addition to the global, multilevel permutation, each matrix  $B_l$  can be further permuted locally by a reordering scheme such as reverse Cuthill-Mckee (RCM) algorithm or approximate minimal degree algorithm (AMD) [51, 52] to reduce the fill-ins.

# 4. Implementation details of parGeMSLR

The parGeMSLR library consists of three main modules: a) a distributed memory reordering scheme, b) a Krylov subspace iterative accelerator, and c) the setup and application of the GeM-SLR preconditioner. The first module was described in greater detail in Section 3, and is implemented through a distributed memory partitioner such as ParMETIS. Additional point-to-point communication between neighboring partitions, as well as a

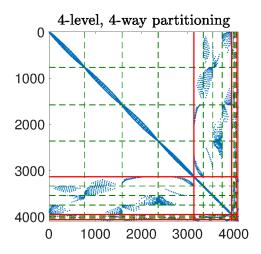
single All-to-All message are required (to find the new neighbors of each partition post-partitioning). Next, we focus on the implementation of the other two modules in a distributed memory environment where different processor groups communicate via MPI.

# 4.1. Distributed memory operations in Krylov accelerators

Standard, non-preconditioned Krylov iterative methods are built on top of simple linear algebraic operations such as matrix-vector multiplication, vector scaling and additions, and DOT products. Iterative solvers such as GMRES or FGMRES also require the solution of small-scale ordinary linear-least squares problems which are typically solved redundantly in each MPI process.

Assuming that the data associated with the system of linear algebraic equations we wish to solve is already distributed across the different MPI process, AXPY operations can be executed locally and involve no communication overhead. On the other hand, sparse matrix-vector multiplications and DOT products involve either point-to-point or collective communication. In particular, assume  $n_p \in \mathbb{N}$  MPI processes. A DOT product then requires a collective operation, i.e., MPI\_Allreduce, to sum the  $n_p$  local DOT products. The cost of this operation is roughly  $O(log(n_p)\alpha)$ , where  $\alpha \in \mathbb{R}$  denotes the latency between any two separate MPI process.<sup>5</sup> On the other hand, a matrixvector multiplication with the coefficient matrix of the linear system requires point-to-point communication, where the local matrix-vector product in each MPI process consists of operations using local data, as well as data associated with MPI processes which are assigned to neighboring subdomains, e.g., see [53] for additional details and recent advances.

<sup>&</sup>lt;sup>5</sup>In practice the latency between different pairs of MPI processes might not be identical depending on the underlying topology, but this difference is generally exceptionally small.



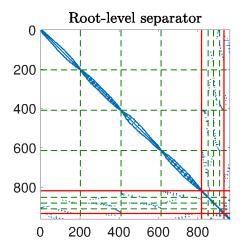


Figure 2: Left: global permutation of matrix A following a multilevel partitioning with  $l_{ev} = 4$  and p = 4. Right: zoom-in at the submatrix associated with the permutation of the vertex separators (right-bottom submatrix of the left subfigure).

# 4.2. Setting up and applying the preconditioner

The main module of parGeMSLR is the setup of the GeM-SLR preconditioner, followed by the application of the latter at each iteration of the Krylov subspace iterative solver of choice. Following a multilevel partition into  $l_{ev}$  levels (see Section 3), the setup phase of the GeMSLR preconditioner associated with each level  $l=0,1,\ldots,l_{ev}-1$ , is further divided into two separate submodules: a) computation of an ILUT factorization  $B_l \approx L_l U_l$ , and b) computation of an approximate rank-k Schur decomposition of the matrix  $\widehat{G}_l = E_l^T U_l^{-1} L_l^{-1} F_l C_l^{-1}$ .

Let us consider each one of the above two tasks separately. Recall that the data matrix at each level  $0 \le l \le l_{ev} - 1$  has the following pattern

$$A_{l} = P_{l}C_{l-1}P_{l} = \begin{bmatrix} B_{l} & F_{l} \\ E_{l} & C_{l} \end{bmatrix} = \begin{bmatrix} B_{l}^{(1)} & & & & F_{l}^{(1)} \\ & B_{l}^{(2)} & & & F_{l}^{(2)} \\ & & \ddots & & \vdots \\ E_{l}^{(1)} & E_{l}^{(2)} & \cdots & E_{l}^{(p)} & C_{l} \end{bmatrix}.$$

Now, without loss of generality, assume that each partition is assigned to a separate MPI process. Figure 3 plots a graphical illustration of the data layout of matrix  $A_l$  obtained by a permutation using p=4, across four different MPI processes. Data associated with separate MPI processes are presented with a different color. Notice that the right-bottom submatrix denotes the matrix  $C_l$  representing the coupling between variables of the vertex separator at level l. Computing an ILUT factorization of the matrix  $B_l$  decouples into p independent ILUT subproblems  $B_l^{(j)} \approx L_l^{(j)} U_l^{(j)}, \ j=1,\ldots,p$ , and thus no communication overhead is enabled. On the other hand, the computation of the low-rank correction term requires the application of several steps of the Arnoldi iteration, and requires communication overhead.

More specifically, the Arnoldi iteration requires communication among the various MPI processes to compute matrix-vector multiplications with the iteration matrix  $\widehat{G}_l$ , as well as to maintain orthogonality of the Krylov basis. When the latter is

# Distribution of matrix A across four MPI processes

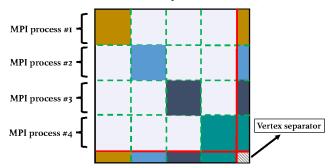


Figure 3: Layout of the matrix correction term across four MPI processes (same for any level  $0 \le l \le l_{ev} - 1$ ).

achieved by means of standard Gram-Schmidt, Arnoldi requires one MPI\_Allreduce operation at each iteration. Similarly, the matrix-vector multiplication between  $\widehat{G}_l$  and a vector z is equal to

$$\begin{bmatrix} E_l^{(1)} & \cdots & E_l^{(p)} \end{bmatrix} \begin{bmatrix} L_l^{(1)} U_l^{(1)} & & & \\ & \ddots & & \\ & & L_l^{(p)} U_l^{(p)} \end{bmatrix}^{-1} \begin{bmatrix} F_l^{(1)} \\ \vdots \\ F_l^{(p)} \end{bmatrix} C_l^{-1} z.$$

The computation of the product  $C_l^{-1}z$  requires access to the incomplete ILUT factorizations and rank-k correction terms associated with all levels  $l < \hat{l} \le l_{ev} - 1$ . Therefore, the rank-k correction terms are built in a bottom-up fashion, from  $l = l_{ev} - 1$  to l = 0, so that level l has immediate access to the data associated with all levels  $\hat{l} > l$ . Once the matrix-vector multiplication  $C_l^{-1}z$  is computed, the matrix-vector multiplication with matrix  $F_l$  is computed with trivial parallelism among the MPI processes, and the same holds for the linear system solutions with matrices  $L_l^{(j)}$ ,  $U_l^{(j)}$ ,  $j = 1, \ldots, p$ . Finally, the matrix-vector multiplication with matrix  $E_l$  requires an MPI\_Allreduce operation. Note though that if we were to replace vertex separators with edge separators (this option is included in parGeMSLR) then the

latter multiplication would also be communication-free.

#### Distribution of low-rank correction terms across four MPI processes

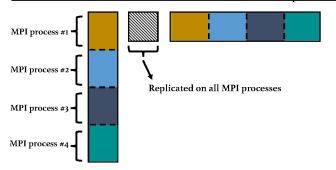


Figure 4: Layout of a rank-k correction term across four MPI processes (same for any level  $0 \le l < l_{ev} - 1$ ).

Finally, applying the preconditioner requires embarrassingly parallel triangular substitutions with the ILUT factorizations of the block-diagonal matrices  $B_l$  as well as dense matrix-vector multiplications with matrices  $\widehat{W}_{l,k}$ ,  $\widehat{W}_{l,k}^H$ , and  $(I-\widehat{R}_{l,k})^{-1}$ . A matrix-vector multiplication with the matrix  $\widehat{W}_{l,k}$  requires no communication among the MPI processes, while a matrix-vector multiplication with the matrix  $\widehat{W}_{l,k}^H$  requires an MPI\_Allreduce operation at level l. Finally, the matrix-vector multiplication with the  $k \times k$  matrix  $(I-\widehat{R}_{l,k})^{-1}$  is performed redundantly in each MPI process since k is typically pretty small.

#### 4.2.1. Communication overhead analysis

In this section we focus on the communication overhead associated with setting up and applying the preconditioner implemented in parGeMSLR. For simplicity, we assume that the number of MPI processes  $n_p$  is equal to the number of partitions p at each level. The main parameters of the preconditioner are the number of levels  $l_{ev}$  and the value of rank k.

Let us first consider the application of m Arnoldi iterations to compute the matrices  $\widehat{W}_{l,k}$  and  $(I - \widehat{R}_{l,k})^{-1}$  for some  $0 \le l \le$  $l_{ev}$  – 1. As was discussed in the previous section, computing matrix-vector products with the matrix  $G_l$  requires communication only during the application of the matrices  $E_l$  and  $C_l^{-1}$ . In turn, the latter requires computations with the distributed matrices  $C_{l+1}^{-1}$ ,  $W_{l+1,k}^{H}$ ,  $C_{l+2}^{-1}$ ,  $W_{l+2,k}^{H}$ , and so on, until we reach level  $l_{ev}-1$  where an ILUT of the matrix  $C_{l_{ev}-1}$  is computed explicitly. Thus, a matrix-vector multiplication with the matrix  $\widehat{G}_l$ requires  $l_{ev}$  – (l+1) (low-rank correction term) and  $l_{ev}$  – l ( $C_l^{-1}$  recursion) MPI\_Allreduce operations. In summary, an *m*-length Arnoldi cycle with standard Gram-Schmidt orthonormalization requires  $(2l_{ev} - 2l + 1)m$  MPI\_Allreduce operations, where we also accounted for the two MPI\_Allreduce operations stemming by Gram-Schmidt and vector normalization at each iteration. This communication overhead is inversely proportional to the level index l. Accounting for all  $l_{ev} - 1$  levels, the total communication overhead associated with the setup phase of the preconditioner amounts is bounded by  $\delta(k) \sum_{l=0}^{l=l_{ev}-1} (2l_{ev} - 1)^{l}$ 2l + 1)m MPI\_Allreduce operations, where  $\delta(k) \in \mathbb{N}$  denotes the maximum number of cycles performed by Arnoldi at any

level. In parGeMSLR, the default cycle length is m=2k iterations. Finally, after the set up phase, one full application of the preconditioner implemented in the parGeMSLR library requires  $2(l_{ev}-l)+1$  MPI\_Allreduce operations.

The analysis presented in this section demonstrates that the communication overhead associated with the construction of the GeMSLR preconditioner is directly proportional to an increase in the value of  $l_{ev}$ . On the other hand, increasing the value of  $l_{ev}$  can reduce the computational complexity associated with setting up the GeMSLR preconditioner in lower levels. Nonetheless, the value of  $l_{ev}$  can not be too large, especially when the value of p is small, since the size of the vertex separator reduces dramatically between successive levels (as is demonstrated in Figure 1).

# 4.3. Applying $C_{l_{cv}-1}^{-1}$

Due to partitioning with a multilevel vertex separator, the matrix  $C_{l_{ev}-1}$  forms a separate partition which is replicated among all MPI processes. Therefore, the simplest approach to apply  $C_{l_{ev}-1}^{-1}$  is to do so approximately, through computing an ILUT redundantly in each MPI process. However, for large problems, this approach can quickly become impractical, even if a shared-memory variant of ILUT is considered [15]. On the other hand, applying a distributed memory approach that requires communication among the MPI processes can lead to considerable high communication overhead since the application of  $C_{l_{ev}-1}^{-1}$  is the most common operation during the setup phase of the preconditioner.

parGeMSLR includes several<sup>6</sup> options to apply an approximation of  $C_{l_{ev}-1}^{-1}$ . The default option considered throughout our experiments is to apply  $C_{l_{ev}-1}^{-1}$  approximately through a block-Jacobi approach where  $C_{l_{ev}-1}$  is first permuted by reverse RCM and then replaced by its on-diagonal block submatrices while the rest of the entries are discarded. Generally speaking, dropping these entries of  $C_{l_{ev}-1}^{-1}$  has minor effects since  $C_{l_{ev}-1}$  is already close to being block-diagonal for modest values of  $l_{ev}$  (e.g., three or four) as was already demonstrated in Figure 1. By default, the number of retained on-diagonal blocks of matrix  $C_{l_{ev}-1}$  is set equal to p. The approximate application of  $C_{l_{ev}-1}^{-1}$  is then trivially parallel among the MPI processes, and each one of the retained on-diagonal blocks is applied through an ILUT.

#### 5. Numerical Experiments

In this section we demonstrate the parallel performance of parGeMSLR. We run our experiments on the Quartz cluster of Lawrence Livermore National Laboratory. Each node of Quartz has 128 GB memory and consists of 2 Intel Xeon E5-2695 CPUs with 36 cores in total. We use the default MPI, MVAPICH2 2.2.3, to compile parGeMSLR. By default, all of

 $<sup>^6</sup> See\ section\ 2.1\ in\ https://github.com/Hitenze/pargemslr/blob/main/ParGeMSLR/DOCS/Documentation.pdf$ 

the experiments presented below are executed in double<sup>7</sup> precision (64-bit precision following the IEE-754 standard). On top of distributed memory parallelism, parGeMSLR can take advantage of shared memory parallelism using either OpenMP or CUDA. The current version of parGeMSLR uses LAPACK for sequential matrix decompositions and ParMETIS for distributed graph partitioning [46]. A detailed documentation of parGeMSLR can be found in the "DOCS" directory of https://github.com/Hitenze/pargemslr. This documentation provides detailed information on how to compile and run parGeMSLR, and includes a detailed description of all command-line parameters as well as visualization of the source code hierarchy. Several test drivers, and a sample input file, are also included.

Throughout the rest of this section, we choose Flexible GM-RES (FGMRES) with a fixed restart size of fifty as the outer iterative solver. The motivation for using FGMRES instead of GMRES is that the application of the preconditioner is subject to variations due to the application of the inner solver in step 9 of Algorithm 1. The stopping tolerance for the relative residual norm in FGMRES is set equal to 1.0e - 6. Unless mentioned otherwise, the solution of the linear system Ax = b will be equal to the vector of all ones with an initial approximation equal to zero. The low-rank correction term at each level consists of approximate Schur vectors such that the corresponding approximate eigenvalues are accurate to two digits of accuracy, and the restart cycle of thick-restart Arnoldi is equal to 2k.

Our distributed memory experiments focus on the parallel efficiency of parGeMSLR both when the problem size remains fixed and  $n_p$  increases (strong scaling) and the problem size increases at the same rate with  $n_p$ . In the case of weak scaling, the parallel efficiency is equal to  $\frac{T_1}{T_{n_p}}$ , where  $T_1$  and  $T_{n_p}$  denote the wall-clock time achieved by the sequential and distributed memory version (using  $n_p$  MPI processes) of parGeMSLR, respectively. Likewise, in the case of strong scaling, the parallel efficiency is equal to  $\frac{T_1}{n_p T_{n_p}}$ . In addition, we also compare parGeMSLR against: a) the BoomerAMG parallel implementation of the algebraic multigrid method in hypre, and b) the two-level SchurILU approach in [23]. The latter preconditioner uses partial ILU to form an approximation of the Schur complement matrix. The preconditioning step is then performed by applying GMRES with block-Jacobi preconditioning to solve the linear system associated with the sparsified Schur complement. The block-Jacobi preconditioner is applied through one step of ILUT, and our implementation of SchurILU is based on the parallel ILU(T) in hypre.

Throughout the rest of this section, we adopt the following notation:

- $\mathbf{n_p} \in \mathbb{N}$ : total number of MPI processes.
- **fill**  $\in \mathbb{R}$ : ratio between the number of non-zero entries of the preconditioner and that of matrix A.
- **p-t**  $\in \mathbb{R}$ : preconditioner setup time. This includes the

- **i-t**  $\in \mathbb{R}$ : iteration time of FGMRES.
- its  $\in \mathbb{N}$ : total number of FGMRES iterations.
- $\mathbf{k} \in \mathbb{N}$ : number of low-rank correction terms at each level.
- **F**: flag signaling that FGMRES failed to converge within 1000 iterations.

#### 5.1. A Model Problem

This section considers a Finite Difference discretization of the model problem

$$-\Delta u - \mathbf{b} \cdot \nabla u - cu = f \text{ in } \Omega,$$
  
$$u = 0 \text{ on } \partial \Omega,$$
 (10)

where  $\mathbf{b} \in \mathbb{R}^3$ . Unless mentioned otherwise, we use a 7-pt stencil, and set  $\mathbf{b} = \mathbf{0}$  and  $\Omega = (0, 1)^3$ .

#### 5.1.1. Weak scaling

Our first set of experiments studies the weak scaling efficiency of parGeMSLR. Since varying the values of  $l_{ev}$  and k lead to different convergence rates, we first consider the case where the number of FGMRES iterations is set equal to thirty, regardless of whether convergence was achieved or not. The problem size on each MPI process is fixed to  $50^3$ , while the number of subdomains at each level is set equal to  $8 \times n_p$ . Moreover, the number of levels is varied as  $l_{ev} \in \{2,3\}^8$  while the rank of the low-rank correction terms is varied as  $k \in \{0,100,200\}$ .

Figure 5 plots the weak scaling efficiency of parGeMSLR on up to  $n_p = 1,024$  MPI processes. The achieved efficiency is similar for both options of  $l_{ev}$  with a slightly higher efficiency observed for the case  $l_{ev} = 3$ . As expected, the highest efficiency achieved during the preconditioner setup phase was for the case k = 0, since there is no communication overhead stemming from the low-rank correction terms. Nonetheless, even in this case there is some loss in efficiency due to load imbalancing introduced by the ILUT factorizations at different levels. Regardless of the value of k, the efficiency of parGeMSLR drops the most when the number of MPI processes is small, regardless of the value of  $l_{ev}$ . This reduction is owed to the relatively large increase on the size of the local Schur complement versus when a larger number of MPI processes is utilized. Note though, although not reported in our experiments, that the weak scaling efficiency is typically much higher when each MPI process handles exactly one subdomain. Finally, the efficiency of the reordering phase is rather limited, since the wall-clock time requires to partition the graph associated with the matrix  $|A| + |A^T|$ and permute the distributed matrix A increases as the problem size grows.

time required to compute the ILUT factorizations and low-rank correction terms in parGeMSLR.

<sup>&</sup>lt;sup>7</sup>We note though that parGeMSLR supports both real and complex arithmetic, as well as both single and double precision.

<sup>&</sup>lt;sup>8</sup>Partitioning the domain into four levels led to a tiny separator in the fourth level.

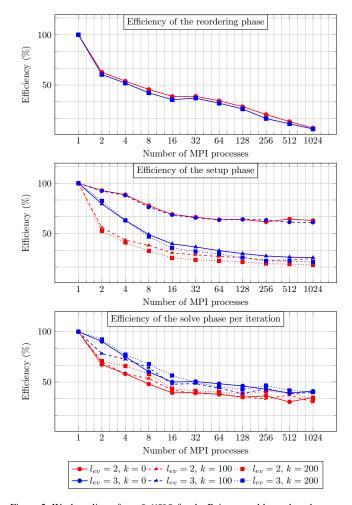


Figure 5: Weak scaling of parGeMSLR for the Poisson problem when the number of iterations performed by FGMRES is fixed to thirty, and the number of levels is set to  $l_{ev}=2$  and  $l_{ev}=3$ . The number of unknowns on each MPI process is 125,000, for a maximum problem size  $n=800\times400\times400$ .

Figure 6 plots the weak scalability of parGeMSLR and two-level SchurILU, where this time we allow enough iterations in FGMRES until convergence. As previously, we use eight subdomains per MPI process, but this time we fix  $l_{ev}=3$  and k=10. In summary, parGeMSLR is both faster and more scalable than SchurILU during the solve phase. Moreover, parGeMSLR also converges much faster than SchurILU, and the number of total FGMRES iterations increases only marginally with the problem size. On the other hand, the weak scaling of the preconditioner setup phase of parGeMSLR is impacted negatively as the problem size increases due to the need to perform more Arnoldi iterations to compute the low-rank correction terms.

#### 5.1.2. Strong scaling

We now present strong scaling results obtained by solving (10) with parGeMSLR on a regular mesh of fixed size as the numbers of MPI processes varies. More specifically, the size of the problem is fixed to  $n = 320^3$  while the number of MPI processes varies up to  $n_p = 1,024$ . The values of  $l_{ev}$  and k are varied as previously.

Figure 7 plots the strong scaling of parGeMSLR. In con-

trast to the weak scaling case, setting  $l_{ev}=2$  leads to higher efficiency during both the setup and application phases of the preconditioner. The reason for this behavior is twofold. First, increasing the value of  $l_{ev}$  generally deteriorates the effectiveness of the preconditioner unless k is large and the threshold used in the local ILUT factorizations is small. Second, decreasing the value of  $l_{ev}$  enhances strong scalability since it leads to smaller communication overheads (i.e., recall the discussion in Section 4). As a general remark, we note that the setup phase of parGeMSLR generally becomes more expensive in terms of floating-point arithmetic operations as  $l_{ev}$  decreases, thus although scalability deteriorates as  $l_{ev}$  increases, the actual wall-clock time might actually decrease if the number of MPI processes used is small.

#### 5.2. General Problems

This section discusses the performance of parGeMSLR on a variety of problems in engineering.

# 5.2.1. Unstructured Poisson problem on a crooked pipe

We consider the numerical solution of (10) on a 3D crooked pipe mesh. The problem is discretized by second-order Finite Elements using the MFEM library [54, 55] with local uniform and parallel mesh refinement. We visualize the (inhomogeneous) mesh using the package GLVis [56] in Figure 8. Our experiments consider different refinement levels to generate problems of different sizes. Moreover, the maximum number of inner iterations in step 9 of Algorithm 1 is varied between three and five. We compare parGeMSLR against BoomerAMG with Hybrid Modified Independent Set (HMIS) coarsening, where we consider both Gauss-Seidel and  $l_1$  Jacobi smoother [57], and report the corresponding results in Table 1. parGeMSLR is able to outperform Schur ILU, especially for larger problems. Moreover, the iteration time of parGeMSLR is similar to that of BoomerAMG with Gauss-Sediel smoother, but much lower than that of BoomerAMG with  $l_1$  Jacobi smoother.

Table 1: Solving (10) on a crooked pipe mesh.

prec	size	$n_p$	k	fill	p-t	i-t	its
Boomer AMG GS	126,805	16	-	1.71	0.17	0.69	106
	966,609	32	-	1.79	0.79	5.7	198
	7,544,257	64	-	1.81	3.36	45.12	250
Boomer	126,805	16	-	1.71	0.18	1.29	226
AMG	966,609	32	-	1.79	0.8	10.95	431
Jacobi	7,544,257	64	-	1.81	3.39	72.1	568
Schur ILU	126,805	16	-	1.53	0.22	0.51	65
	966,609	32	-	1.86	1.2	12.46	383
	7,544,257	64	-	1.94	5.51	-	F
par GeMSLR	126,805	16	10	1.05	0.54	0.46	25
	966,609	32	10	1.18	3.59	4.70	53
	7,544,257	64	10	1.32	11.76	48.35	128

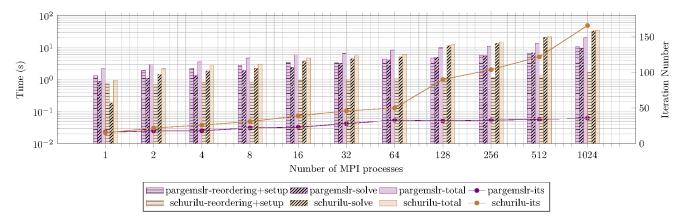


Figure 6: Weak scaling of parGeMSLR and SchurILU on Poisson problems. The number of unknowns on each MPI process is 125,000, for a maximum problem size  $n = 800 \times 400 \times 400$ .

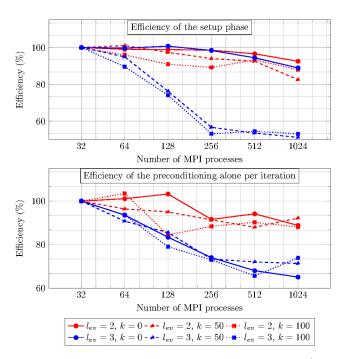


Figure 7: Strong scaling results for Poisson problems of size  $n = 320^3$ . The number of subdomains is set equal to 2048 in all levels.

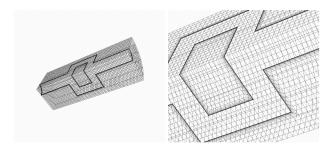


Figure 8: Left: Poisson problem on a crooked pipe mesh. Right: zoom-in of the center part of the mesh.

# 5.2.2. Linear elasticity equation

In the section we consider the solution of the following linear elasticity equation:

$$\mu \Delta u + (\lambda + \mu) \nabla (\nabla \cdot u) = f \text{ in } \Omega, \tag{11}$$

where  $\Omega$  is a 3D cantilever beam as shown in Figure 9. The left

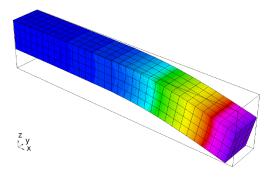


Figure 9: Linear elasticity problem on a 3D beam.

end of the beam is fixed, while a constant force (represented by f) pulls down the beam from the right end. Herein, u is the displacement, while  $\lambda$  and  $\mu$  are the material's Lame constants.

Tables 2 and 3 show a comparison between parGeMSLR and SchurILU for different uniform mesh refinements obtained using first-order Finite Element. For each mesh, the problem becomes more ill-conditioned as the ratio  $\frac{\lambda}{\mu}$  grows larger. For this reason, we fix  $\mu=1$  and vary  $\lambda=10$  and  $\lambda=80$ . Note that standard AMG would typically fail for this problem since the problem is almost singular. Concisely, parGeMSLR leads to considerable wall-clock time savings compared to SchurILU, even when the latter is allowed a higher level of fill-in.

# 5.2.3. Helmholtz equation

In this section we consider the complex version of parGeMSLR and apply it to solve the Helmholtz problem

$$-(\Delta + \omega^2)u = f \text{ in } \Omega = [0, 1]^3, \tag{12}$$

where we use the Perfectly Matched Layer (PML) boundary condition [58] and set the number of points per wavelength equal to eight.

Table 2: Comparison between two-level ILU and the GeMSLR for 3D Linear elasticity problem.  $\mu=1$  and  $\lambda=10$ , Poisson ratio is  $\frac{5}{11}\approx 0.455$ .

prec	size	$n_p$	k	fill	p-t	i-t	its
Schur- ILU	2,475	4	-	2.62	0.03	0.06	49
	15,795	8	-	3.78	0.32	0.60	238
	111,843	16	-	7.81	4.80	19.05	751
	839,619	64	-	11.82	19.67	-	F
par GeMSLR	2,475	4	20	1.94	0.12	0.01	18
	15,795	8	40	3.58	0.92	0.04	23
	111,843	16	40	7.86	10.06	0.64	41
	839,619	64	80	10.05	63.25	3.13	65

Table 3: Comparison between two-level ILU and the GeMSLR for 3D Linear elasticity problem.  $\mu=1$  and  $\lambda=80$ , Poisson ratio is  $\frac{40}{81}\approx0.494$ .

prec	pr	$n_p$	k	fill	p-t	i-t	its
Schur- ILU	2,475	4	-	2.21	0.03	0.26	336
	15,795	8	-	4.03	0.35	1.48	549
	111,843	16	-	8.94	6.45	-	F
	839,619	64	-	14.75	32.17	-	F
par GeMSLR	2,475	4	20	1.91	0.15	0.01	41
	15,795	8	40	3.58	1.09	0.15	75
	111,843	16	80	6.48	16.16	1.49	93
	839,619	64	120	10.31	133.2	6.15	128

Our first set of experiments focuses on the performance of parGeMSLR where the number of low-rank terms is varied as  $k = \{10, 20, ..., 100\}$ , and the number of levels is set equal to  $l_{ev} = 3$ . The size of the Helmholtz problem is set equal to  $n = 50^3$ . The maximum fill-in attributed to the low-rank correction term was roughly equal to three. Figure 10 plots the parallel wall-clock time as a function of the number of low-rank terms k while the number of MPI processes is fixed equal to sixteen. Overall, larger values of k lead to lower total and iteration times up to the point where the time increase associated with constructing the parGeMSLR preconditioner outweighs the gains from improving the convergence rate during the iterative solution by FGMRES. Next, we consider the same problem but

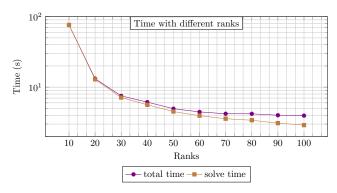


Figure 10: Total and iteration wall-clock times of the 3-level parallel GeMSLR to solve the Helmholtz equation of size  $n = 50^3$  using 16 MPI processes.

this time we add a complex shift equal to  $0.05i * \sum_i |A_{ii}|/n_A$  during the the ILU factorization of the on-diagonal blocks. The same idea was already considered in [59, 25, 14] but this time

Table 4: parGeMSLR with complex shifts. The largest problem size is equal to  $n = 160^3$ .

						i-t	
5π	1	0	2.77	0.04	0.02	0.07	11
$7.5\pi$	1	0	3.45	0.13	0.10	0.52	20
$10\pi$	2	5	3.83	0.17	0.35	1.14	31
$12.5\pi$	4	5	4.01	0.22	0.49	1.77	38
$15\pi$	8	10	3.98	0.24	0.63	2.12	44
$20\pi$	16	20	4.40	0.35	1.10	4.55	57
$40\pi$	64	40	5.18	1.74	4.19	17.26	95

we apply it in the context of distributed memory computing and make it available in parGeMSLR. Similarly to the previous references, adding a shift helps creating a more stable ILU for indefinite problems, i.e., see Table 4.

#### 5.3. GPU acceleration

The parGeMSLR library can take advantage of the availability of GPU hardware to offload the dense linear algebra operations associated with the rectangular matrix-vector multiplications with the low-rank correction term in the GeMSLR preconditioner. To demonstrate the benefits of hybrid CPU+GPU parallelism, we consider a  $n = 128^3$  discretization of the model problem (10) and focus on the speedup achieved per each iteration of FGMRES when the low-rank correction term is applied through GPU hardware, as opposed to the standard option of applying the low-rank correction term through thread-level parallelism using OpenMP. We set the number of levels equal to  $l_{ev} = 2$  and  $l_{ev} = 3$ , and vary the number of low-rank correction terms as  $k \in \{0, 100, 200, 300, 400, 500\}$ . At each level, we apply a 4-way partition and assign each partition to a separate MPI process binded to a V100 NVIDIA GPU. Figure 11 plots the speedups achieved by the hybrid CPU+GPU version of parGeMSLR during its solve phase. As expected, the peak speedup is obtained for the case k = 500, since the cost to apply the low-rank correction term increases linearly with the value of k.

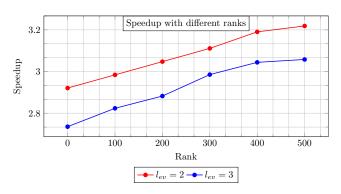


Figure 11: Per-iteration speedup of parGeMSLR with GPU acceleration when  $l_{ev} = \{2,3\}$ , and  $k \in \{0,100,200,300,400,500\}$ . The problem size is equal to  $n=128^3$  problem.

#### 6. Conclusion

In this paper, we presented the parGeMSLR library, a general-purpose C++/MPI software library which extends the GeM-SLR preconditioner into distributed memory computing environments [25]. parGeMSLR is quite flexible by nature, and can be applied to solved both real and complex systems of linear algebraic equations, using either double or single precision. The distributed memory performance of parGeMSLR was demonstrated on a series of problems, verifying the efficiency of the library as a general-purpose solver.

One limitation of parGeMSLR is the amount of time required to complete the setup phase of the preconditioner when the rank of the low-rank correction term is large. As part of our future work we plan to replace standard Arnoldi by its block variant in order to reduce latency during this phase. Another option is to compute the low-rank correction term by applying a few steps of randomized subspace iteration. On a separate note, the cost of the setup phase of the GeMSLR preconditioner can be amortized over the solution of linear systems with the same iteration matrix but varying right-hand sides, e.g. see [60, 61, 62, 63], and we plan to apply parGeMSLR to this type of problems. Finally, as part of future work, we plan to study the application of parGeMSLR to solve linear systems appearing in eigenvalue solvers based on rational filtering [64, 65], and domain decomposition [66, 67].

#### References

- Y. Saad, Iterative Methods for Sparse Linear Systems, Other Titles in Applied Mathematics, Society for Industrial and Applied Mathematics, 2003. doi:10.1137/1.9780898718003.
- [2] H. A. Van der Vorst, Iterative Krylov methods for large linear systems, no. 13, Cambridge University Press, 2003.
- [3] Y. Saad, M. H. Schultz, GMRES: A Generalized Minimal Residual Algorithm for Solving Nonsymmetric Linear Systems, SIAM Journal on Scientific and Statistical Computing 7 (3) (1986) 856–869, publisher: Society for Industrial and Applied Mathematics. doi:10.1137/0907058.
- [4] J. W. Ruge, K. St{\"u}ben, Algebraic Multigrid, in: Multigrid Methods, Frontiers in Applied Mathematics, Society for Industrial and Applied Mathematics, 1987, pp. 73–130. doi:10.1137/1.9781611971057. ch4.
- [5] V. E. Henson, U. M. Yang, BoomerAMG: A parallel algebraic multigrid solver and preconditioner, Applied Numerical Mathematics 41 (1) (2002) 155–177. doi:10.1016/S0168-9274(01)00115-5.
- [6] A. J. Cleary, R. D. Falgout, V. E. Henson, J. E. Jones, T. A. Manteuffel, S. F. McCormick, G. N. Miranda, J. W. Ruge, Robustness and scalability of algebraic multigrid, SIAM Journal on Scientific Computing 21 (5) (2000) 1886–1908.
- [7] N. Bell, S. Dalton, L. N. Olson, Exposing fine-grained parallelism in algebraic multigrid methods, SIAM Journal on Scientific Computing 34 (4) (2012) C123–C152.
- [8] Y. Saad, ILUT: A dual threshold incomplete LU factorization, Numerical Linear Algebra with Applications 1 (4) (1994) 387–402. doi:10.1002/ nla.1680010405.
- [9] E. Chow, Y. Saad, Experimental study of ilu preconditioners for indefinite matrices, Journal of computational and applied mathematics 86 (2) (1997) 387–414.
- [10] O. G. Ernst, M. J. Gander, Why it is Difficult to Solve Helmholtz Problems with Classical Iterative Methods, in: I. G. Graham, T. Y. Hou, O. Lakkis, R. Scheichl (Eds.), Numerical Analysis of Multiscale Problems, Vol. 83, Springer Berlin Heidelberg, Berlin, Heidelberg, 2012, pp. 325–363. doi:10.1007/978-3-642-22061-6\_10.

- [11] X. Liu, Y. Xi, Y. Saad, M. V. de Hoop, Solving the three-dimensional high-frequency helmholtz equation using contour integration and polynomial preconditioning, SIAM Journal on Matrix Analysis and Applications 41 (1) (2020) 58–82.
- [12] M. Magolu monga Made, R. Beauwens, G. Warzée, Preconditioning of discrete Helmholtz operators perturbed by a diagonal complex matrix, Communications in Numerical Methods in Engineering 16 (11) (2000) 801–817. doi:https://doi.org/10.1002/1099-0887 (200011) 16:11<801::AID-CNM377>3.0.C0;2-M.
- [13] Y. A. Erlangga, C. Vuik, C. W. Oosterlee, Comparison of multigrid and incomplete LU shifted-Laplace preconditioners for the inhomogeneous Helmholtz equation, Applied Numerical Mathematics 56 (5) (2006) 648– 666. doi:10.1016/j.apnum.2005.04.039.
- [14] D. Osei-Kuffuor, Y. Saad, Preconditioning Helmholtz linear systems, Applied Numerical Mathematics 60 (4) (2010) 420–431. doi:10.1016/j.apnum.2009.09.003.
- [15] H. Anzt, E. Chow, J. Dongarra, Parilut—a new parallel threshold ilu factorization, SIAM Journal on Scientific Computing 40 (4) (2018) C503– C519.
- [16] H. Anzt, T. Ribizel, G. Flegar, E. Chow, J. Dongarra, Parilut-a parallel threshold ilu for gpus, in: 2019 IEEE International Parallel and Distributed Processing Symposium (IPDPS), IEEE, 2019, pp. 231–241.
- [17] E. Chow, A. Patel, Fine-Grained Parallel Incomplete LU Factorization, SIAM Journal on Scientific Computing 37 (2) (2015) C169–C193. doi: 10.1137/140968896.
- [18] X.-C. Cai, M. Sarkis, A Restricted Additive Schwarz Preconditioner for General Sparse Linear Systems, SIAM Journal on Scientific Computing 21 (2) (1999) 792–797. doi:10.1137/S106482759732678X.
- [19] D. Hysom, A. Pothen, Efficient parallel computation of ILU(k) preconditioners, in: Proceedings of the 1999 ACM/IEEE conference on Supercomputing, SC '99, Association for Computing Machinery, New York, NY, USA, 1999, pp. 29–es. doi:10.1145/331532.331561.
- [20] G. Karypis, V. Kumar, Parallel Threshold-based ILU Factorization, in: Supercomputing, ACM/IEEE 1997 Conference, 1997, pp. 28–28. doi: 10.1145/509593.509621.
- [21] R. Li, Y. Saad, Low-Rank Correction Methods for Algebraic Domain Decomposition Preconditioners, SIAM Journal on Matrix Analysis and Applications 38 (3) (2017) 807–828. doi:10.1137/16M110486X.
- [22] Y. Saad, J. Zhang, BILUTM: A Domain-Based Multilevel Block ILUT Preconditioner for General Sparse Matrices, SIAM Journal on Matrix Analysis and Applications 21 (1) (1999) 279–299, publisher: Society for Industrial and Applied Mathematics. doi:10.1137/ S0895479898341268.
- [23] Z. Li, Y. Saad, M. Sosonkina, pARMS: a parallel version of the algebraic recursive multilevel solver, Numerical Linear Algebra with Applications 10 (5-6) (2003) 485–509. doi:10.1002/nla.325.
- [24] I. C. L. NIEVINSKI, M. SOUZA, P. GOLDFELD, D. A. AUGUSTO, J. R. P. RODRIGUES, L. M. CARVALHO, Parallel Implementation of a Two-level Algebraic ILU(k)-based Domain Decomposition Preconditioner, TEMA (São Carlos) 19 (2018) 59–77, publisher: scielo. doi: 10.5540/tema.2018.019.01.0059.
- [25] G. Dillon, V. Kalantzis, Y. Xi, Y. Saad, A Hierarchical Low Rank Schur Complement Preconditioner for Indefinite Linear Systems, SIAM Journal on Scientific Computing 40 (4) (2018) A2234–A2252. doi:10.1137/ 17M1143320.
- [26] M. Benzi, M. Tuma, A Sparse Approximate Inverse Preconditioner for Nonsymmetric Linear Systems, SIAM Journal on Scientific Computing 19 (3) (1998) 968–994. doi:10.1137/S1064827595294691.
- [27] E. Chow, Y. Saad, Approximate Inverse Preconditioners via Sparse-Sparse Iterations, SIAM Journal on Scientific Computing 19 (3) (1998) 995–1023. doi:10.1137/S1064827594270415.
- [28] C. Janna, M. Ferronato, G. Gambolati, A Block FSAI-ILU Parallel Preconditioner for Symmetric Positive Definite Linear Systems, SIAM Journal on Scientific Computing 32 (5) (2010) 2468–2484, publisher: Society for Industrial and Applied Mathematics. doi:10.1137/090779760.
- [29] H. Anzt, T. K. Huckle, J. Bräckle, J. Dongarra, Incomplete sparse approximate inverses for parallel preconditioning, Parallel Computing 71 (2018) 1–22.
- [30] M. J. Grote, T. Huckle, Parallel preconditioning with sparse approximate inverses, SIAM Journal on Scientific Computing 18 (3) (1997) 838–853.
- [31] X. Ye, Y. Xi, Y. Saad, Preconditioning via gmres in polynomial space

- (2019)
- [32] D. Cai, E. Chow, L. Erlandson, Y. Saad, Y. Xi, SMASH: Structured matrix approximation by separation and hierarchy, Numerical Linear Algebra with Applications 25 (6) (2018) e2204. doi:https://doi.org/10. 1002/nla.2204.
- [33] W. Hackbusch, A Sparse Matrix Arithmetic Based on \$\Cal H\$\-Matrices. Part I: Introduction to \$\Cal H}\-Matrices, Computing 62 (2) (1999) 89–108. doi:10.1007/s006070050015.
- [34] W. Hackbusch, B. N. Khoromskij, A Sparse \$\Cal H\$-Matrix Arithmetic. Part II: Application to Multi-Dimensional Problems, Computing 64 (1) (2000) 21–47. doi:10.1007/PL00021408.
- [35] Y. Xi, J. Xia, S. Cauley, V. Balakrishnan, Superfast and Stable Structured Solvers for Toeplitz Least Squares via Randomized Sampling, SIAM Journal on Matrix Analysis and Applications 35 (1) (2014) 44–72, publisher: Society for Industrial and Applied Mathematics. doi: 10.1137/120895755.
- [36] C. Chen, H. Pouransari, S. Rajamanickam, E. G. Boman, E. Darve, A distributed-memory hierarchical solver for general sparse linear systems, Parallel Computing 74 (2018) 49–64.
- [37] R. Li, Y. Xi, Y. Saad, Schur complement-based domain decomposition preconditioners with low-rank corrections, Numerical Linear Algebra with Applications 23 (4) (2016) 706–729. doi:10.1002/nla.2051.
- [38] R. D. Falgout, U. M. Yang, hypre: A Library of High Performance Preconditioners, in: P. M. A. Sloot, A. G. Hoekstra, C. J. K. Tan, J. J. Dongarra (Eds.), Computational Science ICCS 2002, Lecture Notes in Computer Science, Springer, Berlin, Heidelberg, 2002, pp. 632–641. doi:10.1007/3-540-47789-6 66.
- [39] P. Labs, Paralution v1.1.0, http://www.paralution.com/(2016).
- [40] S. Gawlok, P. Gerstner, S. Haupt, V. Heuveline, J. Kratzke, P. Lösel, K. Mang, M. Schmidtobreick, N. Schoch, N. Schween, J. Schwegler, C. Song, M. Wlotzka, Hiflow3 technical report on release 2.0, Preprint Series of the Engineering Mathematics and Computing Lab (EMCL) 0 (06) (2017). doi:10.11588/emclpp.2017.06.42879.
- [41] L. Grigori, F. Nataf, S. Yousef, Robust algebraic Schur complement preconditioners based on low rank corrections, Research Report RR-8557, INRIA (Jul. 2014).
- [42] Y. Xi, R. Li, Y. Saad, An Algebraic Multilevel Preconditioner with Low-Rank Corrections for Sparse Symmetric Matrices, SIAM Journal on Matrix Analysis and Applications 37 (1) (2016) 235–259. doi:10.1137/ 15M1021830.
- [43] S. Rajamanickam, E. G. Boman, M. A. Heroux, Shylu: A hybrid-hybrid solver for multicore platforms, in: 2012 IEEE 26th International Parallel and Distributed Processing Symposium, IEEE, 2012, pp. 631–643.
- [44] H. A. Daas, T. Rees, J. Scott, Two-level nystr\" om-schur preconditioner for sparse symmetric positive definite matrices, arXiv preprint arXiv:2101.12164 (2021).
- [45] G. W. Stewart, A krylov–schur algorithm for large eigenproblems, SIAM Journal on Matrix Analysis and Applications 23 (3) (2002) 601–614.
- [46] G. Karypis, V. Kumar, A Fast and High Quality Multilevel Scheme for Partitioning Irregular Graphs, SIAM Journal on Scientific Computing 20 (1) (1998) 359–392, publisher: Society for Industrial and Applied Mathematics. doi:10.1137/S1064827595287997.
- [47] U. V. Catalyurek, C. Aykanat, Hypergraph-partitioning-based decomposition for parallel sparse-matrix vector multiplication, IEEE Transactions on Parallel and Distributed Systems 10 (7) (1999) 673–693. doi: 10.1109/71.780863.
- [48] B. Hendrickson, R. Leland, The Chaco User's Guide Version 2, Sandia National Laboratories, Albuquerque NM (1994).
- [49] F. Pellegrini, Scotch and libScotch 5.1 User's Guide, INRIA Bordeaux Sud-Ouest, IPB & LaBRI, UMR CNRS 5800 (2010).
- [50] P. Hénon, Y. Saad, A parallel multistage ilu factorization based on a hierarchical graph decomposition, SIAM Journal on Scientific Computing 28 (6) (2006) 2266–2293.
- [51] P. R. Amestoy, T. A. Davis, I. S. Duff, An Approximate Minimum Degree Ordering Algorithm, SIAM Journal on Matrix Analysis and Applications 17 (4) (1996) 886–905, publisher: Society for Industrial and Applied Mathematics. doi:10.1137/S0895479894278952.
- [52] A. George, J. W. Liu, Computer Solution of Large Sparse Positive Definite, Prentice Hall Professional Technical Reference, 1981.
- [53] A. Bienz, W. D. Gropp, L. N. Olson, Node aware sparse matrix–vector multiplication, Journal of Parallel and Distributed Computing 130 (2019)

- 166-178.
- [54] R. Anderson, J. Andrej, A. Barker, J. Bramwell, J.-S. Camier, J. C. V. Dobrev, Y. Dudouit, A. Fisher, T. Kolev, W. Pazner, M. Stowell, V. Tomov, I. Akkerman, J. Dahm, D. Medina, S. Zampini, MFEM: A modular finite element library, Computers & Mathematics with Applications (2020). doi:10.1016/j.camwa.2020.06.009.
- [55] MFEM: Modular finite element methods [Software], mfem.org. doi: 10.11578/dc.20171025.1248.
- [56] GLVis: Opengl finite element visualization tool, glvis.org. doi:10. 11578/dc.20171025.1249.
- [57] A. H. Baker, R. D. Falgout, T. V. Kolev, U. M. Yang, Multigrid Smoothers for Ultraparallel Computing, SIAM Journal on Scientific Computing 33 (5) (2011) 2864–2887, publisher: Society for Industrial and Applied Mathematics. doi:10.1137/100798806.
- [58] X. Liu, Y. Xi, Y. Saad, M. V. de Hoop, Solving the 3d high-frequency helmholtz equation using contour integration and polynomial preconditioning, arXiv preprint arXiv:1811.12378 (2018).
- [59] Y. A. Erlangga, C. Vuik, C. W. Oosterlee, On a class of preconditioners for solving the helmholtz equation, Applied Numerical Mathematics 50 (3-4) (2004) 409–425.
- [60] V. Simoncini, E. Gallopoulos, An iterative method for nonsymmetric systems with multiple right-hand sides, SIAM Journal on Scientific Computing 16 (4) (1995) 917–933.
- [61] A. Hussam, L. GRIGORI, P. Hénon, P. RICOUX, Enlarged gmres for solving linear systems with one or multiple right-hand sides.
- [62] V. Kalantzis, C. Bekas, A. Curioni, E. Gallopoulos, Accelerating data uncertainty quantification by solving linear systems with multiple righthand sides, Numerical Algorithms 62 (4) (2013) 637–653.
- [63] V. Kalantzis, A. C. I. Malossi, C. Bekas, A. Curioni, E. Gallopoulos, Y. Saad, A scalable iterative dense linear system solver for multiple righthand sides in data analytics, Parallel Computing 74 (2018) 136–153.
- [64] V. Kalantzis, Y. Xi, L. Horesh, Fast randomized non-hermitian eigensolvers based on rational filtering and matrix partitioning, SIAM Journal on Scientific Computing 43 (5) (2021) S791–S815. arXiv:https: //doi.org/10.1137/20M1349217, doi:10.1137/20M1349217.
- [65] Y. Xi, Y. Saad, Computing Partial Spectra with Least-Squares Rational Filters, SIAM Journal on Scientific Computing 38 (5) (2016) A3020– A3045. doi:10.1137/16M1061965.
- [66] V. Kalantzis, Y. Xi, Y. Saad, Beyond automated multilevel substructuring: Domain decomposition with rational filtering, SIAM Journal on Scientific Computing 40 (4) (2018) C477–C502. arXiv:https://doi.org/10. 1137/17M1154527, doi:10.1137/17M1154527.
- [67] V. Kalantzis, A domain decomposition rayleigh–ritz algorithm for symmetric generalized eigenvalue problems, SIAM Journal on Scientific Computing 42 (6) (2020) C410–C435.