parGeMSLR: A Distributed Parallel Generalized Multilevel Schur Complement Low-Rank Preconditioning/Solution Package User's Guide

Tianshi Xu, Vassilis Kalantzis,
Ruipeng Li, Yuanzhe Xi, Geoffrey Dillon, and Yousef Saad

May 18, 2021

Contents

1 Overview of pGeMSLR

The parGeMSLR is an MPI-based sparse linear system solution/preconditioning package implementation with C++. Currently, parMETIS [6] is required to build the parGeMSLR. Thread-level parallelism could be enabled via OpenMP, and GPU acceleration is available via existing CUDA libraries, including cuBlas, cuSparse, and THRUST.

The parallel Generalized Multilevel Schur complement Low-Rank (pGeMSLR) is a Domain Decomposition (DD) based multilevel preconditioner for large and sparse distributed (non)symmetric linear systems of equations. A purely algebraic multilevel reordering is first applied to obtain a block-diagonal structure corresponding to interior nodes and an interface coupling matrix corresponding to the exterior nodes at each given level until reaching the last level. The preconditioner approximately solves the Schur complement system with the interface coupling matrix plus a low-rank correction term. Due to the dense arithmetic introduced by the low-rank correction, this preconditioner is suitable for GPU acceleration.

1.1 Distributed Memory System

Typically, the first step of solving sparse linear systems on distributed memory systems is to partition the original problem's physical domain or partition the adjacency graph of the coefficient matrix A [6, 2, 5, 11]. After that, unknowns are assigned to different MPI processes.

The parGeMSLR focus on the sparse linear systems in the following block form with p MPI processes:

$$Ax = \begin{pmatrix} A_{11} & E_{12} & \cdots & E_{1p} \\ E_{21} & A_{22} & \cdots & E_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ E_{p1} & E_{p2} & \cdots & A_{pp} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_p \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_p \end{pmatrix} = b, \tag{1}$$

where each block row is held by one MPI process, and A_{ii} are square matrices.

If we use the view of DD, each block row is a set of equations that corresponds to a subdomain, and the off-diagonal matrices E_{ij} and E_{ji} are inter-domain couplings between subdomain i and j.

1.2 Edge Separator and Vertex Separator

We will first describe the edge separator and the vertex separator [12].

We could partition the adjacency graph of the coefficient matrix A with vertex-based partition and edge-based partition. With vertex-based partition, we partition the vertices into several subdomains such that no vertices belong to more than one subdomain. With edge-based partition, we partition the edges instead. When the pattern of A is non-symmetric, we could partition the adjacency graph of $|A| + |A^H|$.

Assume each vertex is associated with one unknown. The edge separator is obtained with the vertex-based partition. It is the set of inter-domain edges. The vertex separator is obtained with the edge-based partition. It is the set of vertices shared by multiple subdomains.

Within each subdomain, vertices can be classified into interior vertices and exterior vertices. With the edge separator, interior vertices are those vertices that are not the end of any inter-domain edges. With the vertex separator, interior vertices are those vertices that are not in the vertex separator. In some literature, vertices could also be assigned into three groups [9]. Since each vertex corresponds to one unknown, we can classify unknowns into interior unknowns and exterior unknowns.

In general, if we partition the original problem into p subdomains, we can reorder the matrix to the following block form

$$A_0 = \begin{pmatrix} B_0 & F_0 \\ E_0 & C_0 \end{pmatrix} = \begin{pmatrix} \hat{B}_{0,1} & & & & \hat{F}_{0,1} \\ & \hat{B}_{0,2} & & & \hat{F}_{0,2} \\ & & \ddots & & \vdots \\ & & & \hat{B}_{0,p} & \hat{F}_{0,p} \\ \hat{E}_{0,1} & \hat{E}_{0,2} & \cdots & \hat{E}_{0,p} & C_0 \end{pmatrix}, \tag{2}$$

where $\big(B, F\big)$ corresponds to interior unknowns, and $\big(E, C\big)$ corresponds to exterior unknowns.

If edge-separator is used, the E_0 and F_0 would also have the block diagonal structure:

$$\begin{pmatrix} \hat{B}_{0,1} & & & & \hat{F}_{0,1} & & & \\ & \hat{B}_{0,2} & & & & \hat{F}_{0,2} & & \\ & & \ddots & & & & \ddots & \\ & & & \hat{B}_{0,p} & & & & \hat{F}_{0,p} \\ \hat{E}_{0,1} & & & & \hat{C}_{0,1} & \hat{E}_{12} & \cdots & \hat{E}_{1p} \\ & \hat{E}_{0,2} & & & \hat{E}_{21} & \hat{C}_{0,2} & \cdots & \hat{E}_{2p} \\ & & \ddots & & \vdots & \vdots & \ddots & \vdots \\ & & & \hat{E}_{0,p} & \hat{E}_{p1} & \hat{E}_{p2} & \cdots & \hat{C}_{0,p} \end{pmatrix}, \tag{3}$$

and the Schur complement cam be formed locally by computing the local Schur complement $\hat{S}_{0,i} = \hat{C}_{0,i} - \hat{E}_{0,i}\hat{B}_{0,i}^{-1}\hat{F}_{0,i}$. No communication is required.

1.3 Parallel Preconditioning Methods in pGeMSLR

1.3.1 Block Jacobi Preconditioner

One straight-forward way of preconditioning A in parallel is to use the block Jacobi approach. All off-diagonal blocks $\hat{E}_{ij}$ in equation 1 are ignored. The block Jacobi ILU preconditioners have the following structure:

$$M^{-1} = \begin{pmatrix} U_1^{-1}L_1^{-1} & & & \\ & U_2^{-1}L_2^{-1} & & \\ & & \ddots & \\ & & & U_p^{-1}L_p^{-1} \end{pmatrix}. \tag{4}$$

The major advantage of the block Jacobi approach is that no communication is required when building and applying the preconditioner, making the setup phase scalable and the solve phase scalable per iteration.

However, as the number of MPI processes increases, more entries are dropped. The number of iterations to achieve convergence would typically increase a lot, thus influence the parallel performance.

In the pGeMSLR package, users can also choose to solve the local Schur complement system using sequential GeMSLR preconditioner.

1.3.2 Two-level Schur Complement Approach

The Schur complement approach can be deduced from a block LDU factorization of A_0 in Equation 2 as

$$\begin{pmatrix} I & \\ E_0B_0^{-1} & I \end{pmatrix} \begin{pmatrix} B_0 & \\ & S_0 \end{pmatrix} \begin{pmatrix} I & B_0^{-1}F_0 \\ & I \end{pmatrix} \begin{pmatrix} u_0 \\ v_0 \end{pmatrix} = \begin{pmatrix} f_0 \\ g_0 \end{pmatrix}, \tag{5}$$

where $S_0 = C_0 - E_0B_0^{-1}F_0$ is the Schur complement. With this factorization, A_0^{-1} can be written as

$$\begin{pmatrix} I & -B_0^{-1}F_0 \\ & I \end{pmatrix} \begin{pmatrix} B_0^{-1} & \\ & S_0^{-1} \end{pmatrix} \begin{pmatrix} I & \\ -EB_0^{-1} & I \end{pmatrix}. \tag{6}$$

If edge separator is used on the top level, the global system is in the form of Equation 3. Due to the block diagonal structure of E, B, and F, we can see that $EB^{-1}F$ also has block diagonal structure. Thus, the computation of $EB^{-1}F$ could be done with out communication, and the Schur complement on the top level takes the form

$$S_0 = \begin{pmatrix} \hat{S}_{0,1} & \hat{E}_{12} & \cdots & \hat{E}_{1p} \\ \hat{E}_{21} & \hat{S}_{0,2} & \cdots & \hat{E}_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ \hat{E}_{p1} & \hat{E}_{p2} & \cdots & \hat{S}_{0,p} \end{pmatrix}, \tag{7}$$

.

To form the Schur complement system, we only need to form the local Schur complement $\hat{S}_{0,i}$. Thus, we can easily use some local approaches like Partial ILU or incomplete triangular solve to form the global Schur complement system.

After the explicit form of Schur complement system is available, we can then use other strategies to solve it.

In the pGeMSLR package, we provide the option to form the approximate Schur complement using partial ILUT. After that, the Schur complement is solved with preconditioned GMRES. The simplest approach is to use block Jacobi ILU to preconditioning the Schur complement system. Low-rank approximation could be used to improve the robustness of the block Jacobi ILU preconditioner.

Low-rank approximation of the explicit Schur complement In many cases, the matrix $S^{-1} - C^{-1}$ has low-rank property. More detail could be found in [8, 7, 3].

The matrix C is usually much sparser than matrix S. Given the low-rank property of the matrix $S^{-1} - C^{-1}$, we could use the approximation of C^{-1} as the preconditioner, and add low-rank correction to improve the robustness.

Denote by $\tilde{S}_0$ the explicit approximation of the Schur complement. We can write $\tilde{S}_0^{-1} \approx C_0^{-1}(I + G_0(I - G_0)^{-1})$ where $G_0 = E_0 B_0^{-1} F_0 C_0^{-1}$. Since $I + G_0(I - G_0)^{-1} = (I - G_0)^{-1}$, we have $G_0 \approx I - \tilde{S}_0 C_0^{-1}$. Denote by $\tilde{G}_0$ the matrix $I - \tilde{S}_0 C_0^{-1}$, we use the low-rank approximation of $\tilde{G}_0$ as the low-rank correction.

In the pGeMSLR package, user can choose to use the thick-restart Arnoldi method or the subspace iteration method [13] to solve the eigenvalue problems.

1.3.3 Parallel GeMSLR Approach

For some challenging problems, forming an approximation of the Schur complement might not be accurate enough. For those problem, we can use the parallel version of the Generalized Multilevel Schur complement Low-Rank (GeMSLR) preconditioner proposed in [3]. This is a Schur complement-based approach with low-rank correction, where the Schur complement is kept implicit.

We first recursively apply DD to the C matrix to obtain the multilevel structure

$$A_l = \begin{pmatrix} B_l & F_l \\ E_l & C_l \end{pmatrix}, A_{l+1} := C_l, l = 0, 1, \ldots, L-2., \tag{8}$$

where B_l on each level is block diagonal.

Consider the following inverse of the LDU factorization of A_l

$$\begin{pmatrix} I & -B_l^{-1}F_l \\ & I \end{pmatrix} \begin{pmatrix} B_l^{-1} & \\ & S_l^{-1} \end{pmatrix} \begin{pmatrix} I & \\ -EB_l^{-1} & I \end{pmatrix}. \tag{9}$$

where S_l is the local Schur complement $C_l - E_l B_l^{-1} F_l$.

We can write $S_l^{-1} = C_l^{-1}(I + W_l R_l W_l^H)$ for some orthonormal W. A_l^{-1} could be written as

$$\begin{pmatrix} I & -B_l^{-1}F_l \\ & I \end{pmatrix} \begin{pmatrix} B_l^{-1} & \\ & C_l^{-1}(I + W_l R_l W_l^H) \end{pmatrix} \begin{pmatrix} I & \\ -E_l B_l^{-1} & I \end{pmatrix}. \tag{10}$$

The parallel GeMSLR preconditioner approximating B_l^{-1} through ILU factorization. The low-rank approximation of $I + W_l R_l W_l^H$ is used together with the recursive solve with C_l to approximate S_l^{-1}.

For positive definite problems, typically a small rank number could leads to good convergence performance. More details could be found in [8, 3].

This parallel preconditioner is highly parallelizable except on the last level. When the last level is small enough, each MPI process would holds a copy of the last level, and solve it locally. When the last level is relatively large, we provide two different options. The first option is to use the block Jacobi approach with residual correction. The second option is to use a power method similar to the algorithm described in [14].

1.3.4 Multiplicative Modified GeMSLR Approach

We also provide a multiplicative solve option. With this option, the preconditioner is applied multiplicatively instead of additively. This option is especially useful when the ILU factorization is not accurate enough. One of the major difference is that the Schur complement takes the form $C - E(2B^{-1} - B^{-1}BB^{-1})F$ instead of $C - EB^{-1}F$ where B^{-1} is approximated by the ILU factorization.

2 pGeMSLR Components

2.1 Available Parallel Preconditioners

1. **Global Preconditioner Options**
   - **Block Jacobi**. Block Jacobi preconditioner with ILU or GeMSLR as local preconditioner.
   - **Two-level Schur Complement**. Using Partial ILU to form the Schur complement on the top level explicitly. Preconditioned GMRES could be used to solve the top-level Schur complement system.
   - **Parallel GeMSLR**. The parallel GeMSLR algorithm.
2. **pGeMSLR Solve Options**
   - **LU Solve**. The standard parallel GeMSLR solve.
   - **U Solve**. Only use the upper part of the preconditioner. This option might be useful for highly indefinite problems.
   - **MUL Solve**. The multiplicative parallel GeMSLR algorithm. Each iteration is more expensive than LU Solve, the the conference rate is typically better.
   - **MMUL Solve**. The modified multiplicative parallel GeMSLR algorithm. This option might be useful when the ILU factorization is not accurate enough (ILU0).
3. **Local Preconditioner Options**
   - **ILUT** Threshold-based ILU factorization.
   - **ILUK** Level-based ILU factorization.
   - **GeMSLR** Sequential GeMSLR algorithm.

4. **Last Level Preconditioner Options**

   - **ILUT** Threshold-based ILU factorization.
   - **ILUK** Level-based ILU factorization.
   - **BJ ILUT** Block Jacobi ILUT.
   - **BJ ILUK** Block Jacobi ILUK.
   - **POWER ILUT** Block Jacobi ILUT with power iteration.
   - **POWER ILUK** Block Jacobi ILUK with power iteration.

5. **Low-rank Options**

   - **Standard Arnoldi** Standard Arnoldi method.
   - **Thick-restart Arnoldi** Arnoldi method with thick-restart.
   - **Subspace Iteration** Subspace iteration method.

2.2 Available Accelerators

- **Flexible GMRES**. Currently we only support the Flexible GMRES accelerator.

2.3 GPU Acceleration

- **Prepossessing.** We use the parMETIS for the DD. Thus, the prepossessing phase is host only. If the input matrix is on the device memory, it will be moved to the host memory first.

- **Setup.** Currently we only support ILU factorization on the host memory. On each level, we compute the ILU factorization on the host memory first, and move the results to the device memory to compute the low-rank terms.

- **Solve.** The solve phase is completely on the device memory. It is worth mentioning that for the MPI communication, we still need to copy the data from the device memory to the host memory.

3 Test Files in pGeMSLR

3.1 TESTS/sequential directory

- **driver_gen_gemslr_seq.cpp**. Test with real general problems in the Matrix Market format. Read matrix based on the file matfile_real.

- **driver_gen_gemslrz_seq.cpp**. Test with real complex problems in the Matrix Market format. Read matrix based on the file matfile_complex.

- **driver_laplacian_gemslr_seq.cpp**. Test with real Laplacian problems. Generate Laplacian matrix based on the file lapfile_real.

- **driver_laplacian_gemslrz_seq.cpp**. Test with complex Laplacian problems. Generate Laplacian matrix based on the file lapfile_complex.

3.1.1 Input File

Those test files read most of the settings from a single file. By default, the input file is "inputs". We also provide another file named "inputs_ilut". Sample runs are

- **./driver_laplacian_gemslr_seq.ex -help**

  Print help information.

- **./driver_laplacian_gemslrz_seq.ex**

  Use vector with all ones as solution and zero initial guess. Read input from file "inputs".

- **./driver_gen_gemslr_seq.ex -initrand -fromfile inputs_ilut -writesol test1**

  Use random initial guess, read input from file "inputs_ilut", write solution to file named "test1.sol".

We now explain the input file line by line.

1. **B_precond_top**. B preconditioner option for the top several levels (ILUT, ILUK, GEMSLR).
2. **B_precond_top_levels**. Number of levels we apply the top preconditioner.
3. **B_precond_lower**. B preconditioner option for other levels (ILUT, ILUK, GEMSLR).
4. **kdim**. Dimension of Krylov subspace in (outer) FGMRES.
5. **maxits**. Maximum number of iterations in outer FGMRES.
6. **tol**. Tolorance for FGMRES, stop when $||r|| < tol$ or $\frac{||r||}{||b||} < tol$.
7. **absolute_tol** Use absolute tol $||r||$ or relative tol $\frac{||r||}{||b||}$. (1: absolute; 0: relative).
8. **nlev0**. The number of levels of the GeMSLR preconditioner.
9. **nlev1**. If GeMSLR is used to solve B, the number of levels for the local GeMSLR preconditioner for B.
10. **ncomp0**. The number of subdomains of the GeMSLR preconditioner.
11. **ncomp1**. If GeMSLR is used to solve B, the number of subdomains of the GeMSLR preconditioner.
12. **partition_option0**. The partition option of the GeMSLR preconditioner. (ND, RKWAY).

13. **partition_option1**. If GeMSLR is used to solve B, the partition option of the GeMSLR preconditioner. (ND, RKWAY).

14. **ilu_droptol0**. Tolerance of ILUT for the B block.

15. **ilu_droptol1**. Tolerance of ILUT for the last level.

16. **ilu_max_row_nnz0**. Maximum number of nonzeros per row of ILUT for the B block.

17. **ilu_max_row_nnz1**. Maximum number of nonzeros per row of ILUT for the last level.

18. **ilu_lfil0**. Level of fill of ILUK for the B block.

19. **ilu_lfil1**. Level of fill of ILUK for the last level.

20. **ilu_perm_option**. Permutation option for ILU. (NO, RCM).

21. **lr_arnoldi_option0**. The Low-rank Option in GEMSLR for the top level, standard Arnoldi, Thick-restart Arnoldi, or subspace iteration (STD, TR, SUB).

22. **lr_arnoldi_option1**. The Low-rank Option in GEMSLR for other levels, standard Arnoldi, Thick-restart Arnoldi, or subspace iteration (STD, TR, SUB). Other levels are typically much smaller than the top level, thus we can use different options.

23. **lr_rank0**. The target size of the low-rank correction on the top level. The actual number can be smaller if we can't find enough convergenced eigenvalues.

24. **lr_rank1**. The target size of the low-rank correction on other levels. The actual number can be smaller if we can't find enough convergenced eigenvalues.

25. **lr_tol_eig0**. On the top level, eigenvalues with residual norm smaller than this value are considered convergenceded. This option is only used in the thick-restart Arnoldi.

26. **lr_tol_eig1**. On other levels, eigenvalues with residual norm smaller than this value are considered convergenceded. This option is only used in the thick-restart Arnoldi.

27. **lr_rank_factor0**. This value should be no less than 1.0. We compute neig = lr_rank * lr_rank_factor eigenvalues on the top level. When standard Arnoldi is used, see option lr_arnoldi_factor0. When thick-restart Arnoldi is used, we compute at least neig convergenced eigenvalues. When subspace iterations is used, we use neig vectors for the subspace iteration.

28. **lr_rank_factor1**. This value should be no less than 1.0. We compute neig = lr_rank * lr_rank_factor eigenvalues on other levels. When standard Arnoldi is used, see option lr_arnoldi_factor0. When thick-restart Arnoldi is used, we compute at least neig convergenced eigenvalues. When subspace iterations is used, we use neig vectors for the subspace iteration.

29. **lr_arnoldi_factor0**. This value should be no less than 1.0. We apply msteps = neig * lr_arnoldi_factor Arnoldi steps on the top level. When subspace iterations is used, this option is not used.

30. **lr_arnoldi_factor1**. This value should be no less than 1.0. We apply msteps = neig * lr_arnoldi_factor Arnoldi steps on other levels. When subspace iterations is used, this option is not used.

31. **lr_maxits0**. The max number of restarts for the thick-restart Arnoldi or number of iterations for the subspace iteration on the top level.

32. **lr_maxits1**. The max number of restarts for the thick-restart Arnoldi or number of iterations for the subspace iteration on other levels.

33. **inner_iteration**. Do we use GMRES to solve the top-level Schur complement? (1: yes, 0: no).

34. **inner_iter_tol**. The tolorance of the inner iteration.

35. **inner_iter_maxits**. The max number of FGMRES outer iteration of the inner iteration.

36. **diag_shift**. Complex Version Only. Enable complex shift in the complex ILUT? (1: yes, 0: no).

37. **global_solve**. The global solve option for GeMSLR (LU, U). Multiplicative solve is not supported in the local GeMSLR.

38. **print_level**. The print option. 0: basic info only; 1: more info; 2: plot with gnuplot graphics.

3.1.2 Laplacian Files

lapfile_real and lapfile_complex contains parameters for driver_laplacian_gemslr_seq.cpp and driver_laplacian_gemslrz_seq.cpp.

$$-\Delta \mathbf{u} + \mathbf{a} \cdot \nabla \mathbf{u} + s\mathbf{u} = 0$$

The first line is the number of test matrices. Starting from the second line are the parameters for each test.

In lapfile_real each line is

$$n_x \ n_y \ n_z \ s \ a_x \ a_y \ a_z$$

. In lapfile_complex each line is

$$n_x \ n_y \ n_z \ real(s) \ imag(s) \ real(a_x) \ imag(a_x) \ real(a_y) \ imag(a_y) \ real(a_z) \ imag(a_z)$$

.

3.1.3 Matrix Files

matfile_real and matfile_complex contains parameters for driver_gen_gemslr_seq.cpp and driver_gen_gemslrz_seq.cpp.

The first line is the number of test matrices. Starting from the second line are the parameters for each test. Each test problem requires one line for the matrix and one line for the right-hand-side.

The first line is the path of the test problem in Matrix Market format. The second line starts from a number. If this line starts from 0, we use the artificial right-hand-side. Otherwise we read the right-hand-side in Matrix Marker format from the path after the first number.

3.1.4 Command Line Options

There are several command line options.

- **-fromfile** [**str**]. Read input parameters from a new file instead of "inputs".
- **-outfile** [**str**]. Write output to file instead of terminal output.
- **-writesol** [**str**]. Write solution vector to file.
- **-zerobase**. For general matrices. The input Matrix Marker format is 0-based instead of 1-based.
- **-rand_perturb**. Add random real diagonal perturbation $A = A - D$. Each entry in D is Rand()$*|shift|$.
- **-solone**. Using vector with all ones as solution.
- **-solrand**. Using random vector as solution.
- **-rhsone**. Using vector with all ones as right-hand-side.
- **-rhsrand**. Using random vector as right-hand-side.
- **-initzero**. Using zero vector as initial guess.
- **-initone**. Using vector with all ones as initial guess.
- **-initrand**. Using random vector as initial guess.
- **-gpu**. Enable GPU acceleration. Need to compile with CUDA. Note that the ILU factorization, LAPACK, and parMETIS are host only. Thus, some parts of the setup phase remain on the host.
- **-nthreads** [**int**]. Number of OpenMP threads for each MPI process. Need to compile with OpenMP.

3.2 TESTS/parallel directory

- **driver_gen_gemslr_par.cpp**. Test with real general problems in the Matrix Market format. Read matrix based on the file matfile_real. Currently we only support reading from single Matrix Market file.

- **driver_gen_gemslrz_par.cpp**. Test with real complex problems in the Matrix Market format. Read matrix based on the file matfile_complex. Currently we only support reading from single Matrix Market file.

- **driver_laplacian_gemslr_par.cpp**. Test with real Laplacian problems. Generate Laplacian matrix based on the file lapfile_real.

- **driver_laplacian_gemslrz_par.cpp**. Test with complex Laplacian problems. Generate Laplacian matrix based on the file lapfile_complex.

3.2.1 Input File

Those test files read most of the settings from a single file. By default, the input file is "inputs". We also provide another file named "inputs_ilut". Sample runs are

- **mpirun -np 1 ./driver_laplacian_gemslr_par.ex -help**

  Print help information.

- **mpirun -np 2 ./driver_laplacian_gemslrz_par.ex**

  Use vector with all ones as solution and zero initial guess. Read input from file "inputs". Using 2 MPI processes.

- **mpirun -np 16 ./driver_gen_gemslr_par.ex -initrand -fromfile inputs_ilut -writesol test1**

  Use random initial guess, read input from file "inputs_ilut", write solution to files named "test1xxx.sol". Here "xxx" is the MPI process number starting from 0.

We now explain the input file line by line.

1. **global_precond**. The global preconditioner option. Currently we support block Jacobi (BJ); parallel GeMSLR (GEMSLR); and an option which forms the top-level Schur complement system using edge separator and partial ILU (ESCHUR).

2. **use_global_partition**. Should we call parMETIS on the entire matrix A? If set to true, we'll call parMETIS on the entire matrix A which would be more expensive. Otherwise we'll use the vertex-based partition provided by the input distributed system. Note that this option is only used for GEMSLR global preconditioner. Global partition is not used for BJ and ESCHUR. (1: yes; 0: no).

3. **B_precond_top**. B preconditioner option for the top several levels (ILUT, ILUK, GEMSLR).

4. **B_precond_top_levels**. Number of levels we apply the top preconditioner.

5. **B_precond_lower**. B preconditioner option for other levels except the last level (ILUT, ILUK, GEMSLR).

6. **C_precond**. Preconditioner option for the last levels (ILUT, ILUK, BJILUT, BJILUK, BJILUT2, BJILUK2). If the size of the last level is small, ILUT/ILUK are recommended. The BJILUT/BJILUK are the block Jacobi ILU options. Those options are less accurate. The BJILUT2/BJILUK2 are the block Jacobi ILU with power iteration. With those options, the low-rank correction on the last level is modified. Those options are more expensive, but the parallel performance is in general better for larger problems.

7. **C_iter**. If BJILUT2/BJILUK2 is used, the number of power iterations we apply. BJILUT2/BJILUK2 are equivalent to BJILUT/BJILUK when this number is set to 0.

8. **kdim**. Dimension of Krylov subspace in (outer) FGMRES.

9. **maxits**. Maximum number of iterations in outer FGMRES.

10. **tol**. Tolorance for FGMRES, stop when $||r|| < tol$ or $\frac{||r||}{||b||} < tol$.

11. **absolute_tol** Use absolute tol $||r||$ or relative tol $\frac{||r||}{||b||}$. (1: absolute; 0: relative).

12. **nlev**. The number of levels of the global parallel GeMSLR preconditioner and the local sequential GeMSLR preconditioner.

13. **ncomp**. The number of subdomains of the global parallel GeMSLR preconditioner and the local sequential GeMSLR preconditioner.

14. **global_partition_option**. The partition option of the global parallel GeMSLR preconditioner. (ND, RKWAY). Recursive k-way partition (RKWAY) is the recommended option for best parallel performance. Note that we use the ND from the parMETIS. For best performance, the number of MPI processes should be some power of 2.

15. **local_partition_option**. If GeMSLR is used to solve B, the partition option of the local GeMSLR preconditioner. (ND, RKWAY).

16. **ilu_droptol**. Tolerance of ILUT for the B block and the last level.

17. **ilu_max_row_nnz**. Maximum number of nonzeros per row of ILUT for the B block and the last level.

18. **ilu_lfil**. Level of fill of ILUK for the B block and the last level.

19. **ilu_perm_option**. Permutation option for ILU. (NO, RCM).

20. **lr_arnoldi_option1**. The Low-rank option for the top level, standard Arnoldi, Thick-restart Arnoldi, or subspace iteration (STD, TR, SUB).

21. **lr_arnoldi_option2**. The Low-rank option other levels, standard Arnoldi, Thick-restart Arnoldi, or subspace iteration (STD, TR, SUB). Other levels are typically much smaller than the top level, thus we can use different options.

22. **lr_arnoldi_optionA**. The low-rank option outside the preconditioner. That is, when applying the M^{-1} to A, another low-rank correction is added as $(I + LR)M^{-1}A$.

23. **lr_rank**. The target size of the low-rank correction on the top level, other levels, and A. The actual number can be smaller if we can't find enough convergenced eigenvalues. Or larger if the smallest eigenvalue we pick is a pair of eigenvalues.

24. **lr_tol_eig**. On the top level, other levels, and A, eigenvalues with residual norm smaller than this value are considered convergenceded. This option is only used in the thick-restart Arnoldi.

25. **lr_rank_factor**. This value should be no less than 1.0. We compute neig = lr_rank * lr_rank_factor eigenvalues on the top level and other levels. When standard Arnoldi is used, see option lr_arnoldi_factor0. When thick-restart Arnoldi is used, we compute at least neig convergenced eigenvalues. When subspace iterations is used, we use neig vectors for the subspace iteration.

26. **lr_arnoldi_factor**. This value should be no less than 1.0. We apply msteps = neig * lr_arnoldi_factor Arnoldi steps on the top level, other levels, and A. When subspace iterations is used, this option is not used.

27. **lr_maxits**. The max number of restarts for the thick-restart Arnoldi or number of iterations for the subspace iteration on the top level, other levels, and A.

28. **inner_iteration**. Do we use GMRES to solve the top-level Schur complement? (1: yes, 0: no).

29. **inner_iter_tol**. The tolorance of the inner iteration.

30. **inner_iter_maxits**. The max number of FGMRES outer iteration of the inner iteration.

31. **diag_shift**. Complex Version Only. Enable complex shift in the complex ILUT? (1: yes, 0: no).

32. **global_solve**. The global solve option for GeMSLR (LU, U, MUL, MMUL). See section 2.1 for detail.

33. **residual_correction**. The number of residual correction with B solve. If larger than 0, we apply residual correction with $x = M^{-1}rhs, r = b - B * x, e = M^{-1}r, x = x + e$ several times.

34. **gs_option**. Gram-Schmidt option. 0: CGS-2; 1: MGS. The CGS-2 is the classic Gram-Schmidt with re-orthgonization.

35. **B_smoother**. The smoother used in the MMUL solve. The default option is the MILU. (MILU,ILU). If MILU option is used, we use the approximation of B^{-1} in the interpolation operator as the fine-grid smoother. With this option, the memory cost is typically halved, and the preconditioner is more robust. However, the weak scalability for simple problems (like Poisson equation) is reduced. If ILU option is used, the standard ILU is used as the fine-grid smoother. This option provides better weak scaling results for elliptic PDEs on regular meshes. However, the memory cost is higher.

36. **npx, npy, npz**. The number of MPI processes on each direction. If $npx * npy * npz \neq np$, default value $1, np, 1$ is used.

37. **ndx, ndy, ndz**. For Laplacian tests, we can pre-assign a domain decomposition. This is the number of partition on each direction.

38. **print_level**. The print option. 0: basic info only; 1: more info; 2: plot with gnuplot graphics.

3.2.2 Laplacian Files

lapfile_real and lapfile_complex contains parameters for driver_laplacian_gemslr_seq.cpp and driver_laplacian_gemslrz_seq.cpp.

$$-\Delta \mathbf{u} + \mathbf{a} \cdot \nabla \mathbf{u} + s\mathbf{u} = 0$$

The first line is the number of test matrices. Starting from the second line are the parameters for each test.

In lapfile_real each line is

$$n_x \ n_y \ n_z \ s \ a_x \ a_y \ a_z$$

. In lapfile_complex each line is

$$n_x \ n_y \ n_z \ real(s) \ imag(s) \ real(a_x) \ imag(a_x) \ real(a_y) \ imag(a_y) \ real(a_z) \ imag(a_z)$$

.

3.2.3 Matrix Files

matfile_real and matfile_complex contains parameters for driver_gen_gemslr_seq.cpp and driver_gen_gemslrz_seq.cpp.

The first line is the number of test matrices. Starting from the second line are the parameters for each test. Each test problem requires one line for the matrix and one line for the right-hand-side.

The first line is the path of the test problem in Matrix Market format. The second line starts from a number. If this line starts from 0, we use the artificial right-hand-side. Otherwise we read the right-hand-side in Matrix Marker format from the path after the first number.

3.2.4 Command Line Options

There are several command line options.

- **-fromfile** [**str**]. Read input parameters from a new file instead of "inputs".
- **-outfile** [**str**]. Write output to file instead of terminal output.
- **-writesol** [**str**]. Write solution vector to file.
- **-zerobase**. For general matrices. The input Matrix Marker format is 0-based instead of 1-based.
- **-rand_perturb**. Add random real diagonal perturbation $A = A - D$. Each entry in D is Rand()$*|shift|$.
- **-solone**. Using vector with all ones as solution.
- **-solrand**. Using random vector as solution.
- **-rhsone**. Using vector with all ones as right-hand-side.
- **-rhsrand**. Using random vector as right-hand-side.
- **-initzero**. Using zero vector as initial guess.
- **-initone**. Using vector with all ones as initial guess.
- **-initrand**. Using random vector as initial guess.
- **-gpu**. Enable GPU acceleration. Need to compile with CUDA. Note that the ILU factorization, LAPACK, and parMETIS are host only. Thus, some parts of the setup phase remain on the host.
- **-nthreads** [**int**]. Number of OpenMP threads for each MPI process. Need to compile with OpenMP.

3.3 TESTS/hypre directory

We provide an interface to hypre[4] in the pGeMSLR.

- **ij.c**. Test file with the interface to hypre. The solver option of FGMRES-pGeMSLR is set to 83. Use ./ij -help to see more instructions.

3.4 TESTS/hypre/mfem directory

We've also provide an interfance to MFEM[1, 10] in the pGeMSLR.

- **ex1p.cpp**. Test file for Poisson equation. Use ./ex1p -help to see more instructions.
- **ex2p.cpp**. Test file for a linear Elasticity problem. Use ./ex2p -help to see more instructions.

4 Installation of pGeMSLR

4.1 Required External Packages

Currently, BLAS, LAPACK, and parMETIS are required to build the pGeMSLR package. Some routines from hypre and ITSOL are used in the package for ILU factorization.

The cuSPARSE, cuBLAS, THRUST, and cuRand are required to build the GPU version of the package.

The package hypre and MFEM are required form hypre interface and MFEM interface.

4.2 Installation

1. **Compile Required External Packages**. The first step is to make the required external packages.

   - The parMETIS need to be compiled with 64 bit signed integers and double precision floating point (double).

2. **Update the makefile.in**.

   A sample makefile.in is provided at the root folder.

   Before installation, modify the makefile.in to set the installation options and the PATH to required packages.

3. **Make the Package**. Simply use the make command to compile the package.

4. **Make test files**. Go to folder TESTS/sequential or TESTS/parallel, and use make command to build the test files.

5. **Make the hypre test file**. Go to folder TESTS/hypre, and use make command to build the test file. Note that the path to the src folder of hypre is required. The makefile might need to be updated manually.

6. **Make the MFEM test files**. Make the hypre test file first.

   Go to folder TESTS/hypre/mfem, and use make command to build the test file. Note that the path to the MFEM is required. The Makefile need to be updated manually.

5 Guide on Parallel Preconditioning Options

Finally we provide a guide on how to select the parallel preconditioners.

5.1 Common Preconditioners

- **Block Jacobi Preconditioner BJ**. The standard block Jacobi preconditioner.
  - **pros**. No communication required. Scalable per iteration.
  - **cons**. Not feasible for large problems with large number of MPI processes.

This option is useful for simple problems with small number of MPI processes.

- **Two-level Schur Complement Preconditioner ESCHUR**. To use the standard two-level Schur complement ILU preconditioner, use the ESCHUR global preconditioner, set the rank number on the top level to be 0, and turn on the inner iteration.

  By doing so, the top-level Schur complement is formed with Partial ILU, and solved with preconditioned GMRES.

  This option has similar performance compared to the two-level ILU preconditioners in hypre.

  - **pros**. Faster setup phase compared with low-rank preconditioner options. More robust than the **BJ**.
  - **cons**. Not very efficient for challenging problems. Limited weak scalability.

  This option could be used to solve any problems. For challenging problems, accurate ILU factorization and more inner iterations are required for good convergence results.

- **Two-level Schur Complement Preconditioner with low-rank correction ESCHUR-LR**.

  By adding a low-rank correction on the top level, we can improve the convergence performance of the two-level Schur complement preconditioner.

  - **pros**. Better convergence rate compared with **ESCHUR**. Faster setup phase compared with **GEMSLR** options.
  - **cons**. Slower setup phase compared with **ESCHUR**. Worse convergence rate compared with **GEMSLR** options.

  This option is similar to the **ESCHUR** options.

  For easy problems, the inner iteration could be turned off for faster solve phase.

  For challenging problems, the inner iterations is required for good convergence results. The low-rank correction could improve the robustness of the preconditioner. The **ESCHUR-LR** is typically better than **ESCHUR** in the solve phase.

- **GeMSLR Preconditioners GEMSLR**.

  - **pros**. Better convergence rate compared with **ESCHUR** and **ESCHUR-LR**. Faster solve phase. Better GPU speedup.
  - **cons**. Slower setup phase compared with **ESCHUR** and **ESCHUR-LR**. A pretty accurate ILU factorization is required for challenging problems. Otherwise increase rank number doesn't guarantee an improvement of the convergence rate.

  The standard GeMSLR option is useful when solving same problem with multiple right-hand-sides, or solving challenging problems.

  The setup phase could be very slow, but this is sometime the only feasible option.

- **GeMSLR Preconditioners with U solve GEMSLR-U**.

  This only apply only the upper part of the preconditioner.

  - **pros**. Improved robustness for some highly indefinite problems compared with **GEMSLR**.
  - **cons**. For most problems **GEMSLR** is better than **GEMSLR-U**.

  It is recommended to use the **GEMSLR** and the **GEMSLR-MUL** options first. If the convergence is not satisfied, **GEMSLR-U** could be tested as an alternative.

- **Multiplicative GeMSLR Preconditioners GEMSLR-MUL**.

  The multiplicative parallel GeMSLR option.

  - **pros**. Better convergence rate compared with **GEMSLR**, especially when the ILU factorization is not accurate enough.
  - **cons**. Slower time per iteration.

  When the memory is an issue, **GEMSLR-MUL** could be used together with a less accurate ILU factorization. When the ILU factorization is accurate enough, typically **GEMSLR** is the better option.

- **Modified Multiplicative GeMSLR Preconditioners GEMSLR-MMUL**.

  This is a experimental option. The modified multiplicative parallel GeMSLR option.

  - **pros**. Better weak scalability for some problems.
  - **cons**. Not feasible for some challenging problems.

  For positive definite problems and moderately indefinite problems, the **GEMSLR-MMUL** could be tested first. Typically this option doesn't require very accurate ILU factorization.

  If this option doesn't work, user can then switch to **GEMSLR-MUL** or **GEMSLR**.

5.2 Different Problems

- **Positive Definite problems or moderately indefinite problems**. For positive definite problems, users could try **GEMSLR-MMUL** with ILU(0) or ILU(1) first. If this option doesn't work well, **ESCHUR** or **ESCHUR-LR** are recommended.

  If multiple right-hand-sides are solved with the same system, **GEMSLR** could also be useful.

- **Large Sparse Linear Systems**. For large problem, users could try **GEMSLR-MMUL** first. If the convergence is not satisfied, **GEMSLR** is recommended.

- **Challenging problems**. For challenging problems, typically **GEMSLR** with a high rank number and inner iteration is the only option. User could also try **GEMSLR-U**.

  For complex problems, enable diagonal shift could sometime improve the robustness of the preconditioner.

5.3 OpenMP and GPU

- **GPU acceleration**. The GPU version of the triangular solve is inefficient for small matrices. Thus, to have better GPU performance, each diagonal block in B_l should not be too small. GPU version of the pGeMSLR is only recommended for larger problems.

  For small problems, a better option is to use BJ-ILU preconditioner with GPU acceleration, or CPU version of the GeMSLR preconditioner if the convergence of BJ-ILU is not satisfied.

- **OpenMP**.

  We provide level-scheduling triangular solve in our OpenMP implementation. This is again inefficient for small matrices.

  By default, the level-scheduling triangular solve is **NOT** enabled in the pGeMSLR preconditioner. We assign the diagonal blocks of B_l to OpenMP threads, and each block is solved sequentially.

6 Contact Information

Pleases submit bugs to `saad@umn.edu` or `xuxx1180@umn.edu`. Thank you!

References

[1] Anderson, R., Andrej, J., Barker, A., Bramwell, J., Camier, J.-S., Dobrev, J. C. V., Dudouit, Y., Fisher, A., Kolev, T., Pazner, W., Stowell, M., Tomov, V., Akkerman, I., Dahm, J., Medina, D., and Zampini, S. MFEM: A modular finite element library. *Computers & Mathematics with Applications* (2020).

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

[3] Dillon, G., Kalantzis, V., Xi, Y., and Saad, Y. A Hierarchical Low Rank Schur Complement Preconditioner for Indefinite Linear Systems. *SIAM Journal on Scientific Computing 40*, 4 (Jan. 2018), A2234–A2252.

[4] Falgout, R. D., and Yang, U. M. hypre: A Library of High Performance Preconditioners. In *Computational Science ICCS 2002* (Berlin, Heidelberg, 2002), P. M. A. Sloot, A. G. Hoekstra, C. J. K. Tan, and J. J. Dongarra, Eds., Lecture Notes in Computer Science, Springer, pp. 632–641.

[5] Hendrickson, B., and Leland, R. *The Chaco User's Guide Version 2*. Sandia National Laboratories, Albuquerque NM, 1994.

[6] Karypis, G., and Kumar, V. A Fast and High Quality Multilevel Scheme for Partitioning Irregular Graphs. *SIAM Journal on Scientific Computing 20*, 1 (Jan. 1998), 359–392. Publisher: Society for Industrial and Applied Mathematics.

[7] Li, R., and Saad, Y. Low-Rank Correction Methods for Algebraic Domain Decomposition Preconditioners. *SIAM Journal on Matrix Analysis and Applications 38*, 3 (Jan. 2017), 807–828.

[8] Li, R., Xi, Y., and Saad, Y. Schur complement-based domain decomposition preconditioners with low-rank corrections. *Numerical Linear Algebra with Applications 23*, 4 (Aug. 2016), 706–729.

[9] Li, Z., Saad, Y., and Sosonkina, M. pARMS: a parallel version of the algebraic recursive multilevel solver. *Numerical Linear Algebra with Applications 10*, 5-6 (July 2003), 485–509.

[10] MFEM: Modular finite element methods [Software]. `mfem.org`.

[11] Pellegrini, F. *Scotch and libScotch 5.1 User's Guide*. INRIA Bordeaux Sud-Ouest, IPB & LaBRI, UMR CNRS 5800, 2010.

[12] Saad, Y. *Iterative Methods for Sparse Linear Systems*. Other Titles in Applied Mathematics. Society for Industrial and Applied Mathematics, Jan. 2003.

[13] Saad, Y. *Numerical Methods for Large Eigenvalue Problems*. Classics in Applied Mathematics. Society for Industrial and Applied Mathematics, Jan. 2011.

[14] Zheng, Q., Xi, Y., and Saad, Y. A power schur complement low-rank correction preconditioner for general sparse linear systems, 2020.