A HIERARCHICAL LOW-RANK SCHUR COMPLEMENT PRECONDITIONER FOR INDEFINITE LINEAR SYSTEMS *

GEOFFREY DILLON†, VASSILIS KALANTZIS†, YUANZHE XI†, AND YOUSEF SAAD†

Abstract. Nonsymmetric and highly indefinite linear systems can be quite difficult to solve by iterative methods. This paper combines ideas from the Multilevel Schur Low-Rank preconditioner developed by Y. Xi, R. Li, and Y. Saad [*SIAM J. Matrix Anal.*, 37 (2016), pp. 235–259] with classic block preconditioning strategies in order to handle this case. The method to be described generates a tree structure $\mathcal{T}$ that represents a hierarchical decomposition of the original matrix. This decomposition gives rise to a block structured matrix at each level of $\mathcal{T}$. An approximate inverse of the original matrix based its block LU factorization is computed at each level via a low rank property that characterizes the difference between the inverses of the Schur complement and another block of the reordered matrix. The low rank correction matrix is computed by several steps of the Arnoldi process. Numerical results illustrate the robustness of the proposed preconditioner with respect to indefiniteness for a few discretized Partial Differential Equations (PDEs) and publicly available test problems.

Key words. block preconditioner, Schur complements, multilevel, low rank approximation, Krylov subspace methods, domain decomposition, Nested Dissection ordering.

AMS subject classifications. 65F08, 65F10, 65F50, 65N55, 65Y05

1. Introduction.

This paper focuses on the solution of large nonsymmetric sparse linear systems

$$Ax = b \tag{1.1}$$

via Krylov subspace methods where $A \in \mathbb{C}^{n\times n}$ and $b \in \mathbb{C}^n$. When solving (1.1) it is often necessary to combine one of these Krylov methods with some form of preconditioning. For example, a *right-preconditioning* method would solve the system $AM^{-1}u = b, M^{-1}u = x$, in place of (1.1). Other variants include left and 2-sided preconditioners. Ideally, M is an approximation to A such that it is significantly easier to solve linear systems with it than with the original A.

A commonly used preconditioner is the Incomplete LU (ILU) factorization of A, where $A \approx LU = M$. ILU preconditioners can be very effective for certain types of linear systems. However, if the original matrix A is poorly conditioned or highly indefinite (A has eigenvalues on both sides of the imaginary axis) then ILU methods can fail due to very small pivots or unstable factors [12, 42]. Another disadvantage of ILU methods is their poor performance on high-performance computers, e.g., those with GPUs [33] or Intel Xeon Phi processors. Algebraic multigrid (AMG) is another popular technique for solving problems arising from discretized PDEs. Multigrid methods are provably optimal for a wide range of SPD matrices and also perform well in parallel. However, without specialization, multigrid will fail on even mildly indefinite problems. Sparse approximate inverses emerged in the 1990s as alternatives to ILU factorizations [9, 13, 22]. These methods were mostly abandoned due to their high cost both in terms of arithmetic and memory usage. A subsequent class of preconditioners were based on *rank-structured matrices* [10]. Two such types of matrices are $\mathcal{H}^2$-matrices [23, 24] and hierarchically semiseparable (HSS) matrices [49, 50, 51].

*This work was supported by NSF under grant DMS-1521573 and by the Minnesota Supercomputing Institute.

†Address: Department of Computer Science & Engineering, University of Minnesota, Twin Cities. {gdillon,kalan019,yxi,saad}@umn.edu

Both of these forms are the result of a partition of the original matrix where some of the off-diagonal blocks are approximated by low rank matrices. These ideas have been used to develop both sparse direct solvers and preconditioners [52]. Similarly, it also is possible to exploit preconditioners based on hierarchical LU factorizations [4].

In this paper we focus on approximate inverse preconditioners which are based on low rank corrections. Such approaches include the Multilevel Low-Rank (MLR) [32], the Schur complement low rank (SLR) preconditioner [34], and the Multilevel Schur complement Low-Rank (MSLR) preconditioner [46]. The idea behind the MSLR preconditioner is to combine a multilevel Hierarchical interface decomposition (HID) ordering [25] along with an efficient Schur complement approximation. This approach is shown to be much less sensitive to indefiniteness than the classical ILU and domain decomposition based methods. However, MSLR is designed for symmetric problems. This paper presents a preconditioner that incorporates a modified hierarchical low rank approximation of the inverse Schur complement from the MSLR preconditioner into a block preconditioner based on the block LU factorization of A. The resulting method will be called a Generalized Multilevel Schur complement Low-Rank (GMSLR) preconditioner. Two characteristics of GMSLR are worth highlighting. First *GMSLR is designed to be applicable to a wide range of problems.* The preconditioner is nonsymmetric and it changes at each iteration, since it incorporates inner solves, and flexible GMRES [40] is used as the accelerator. The method also performs well for symmetric matrices. As observed in [7, Section 10.1.2], the loss of symmetry incurred by application of a nonsymmetric preconditioner is not a major concern provided that good approximations to certain blocks of A are available. The numerical experiments will confirm this observation. Second, a property that is inherited from MSLR is that the *GMSLR preconditioner computes a recursive, multilevel approximation to the inverse of the Schur complement.* GMSLR is a block preconditioner with inner sub-solves required at every outer iteration. These inner solves can themselves be preconditioned in order to reduce computational costs. One of these required inner solves is with the Schur complement, i.e., we must solve $Sy = g$. For most problems, this inverse Schur complement approximation turns out to be an effective preconditioner for these inner solves. Since an important goal of this paper is to deal with indefinite problems, we explored another improvement targetted specifically at such problems. This improvement consists of a well-established strategy [20, 35, 39, 45, 48] of adding complex shifts to the diagonal prior to performing any of the ILU factorizations required by GMSLR. In the case of GMSLR, this entails modifying the diagonal of coefficient matrix at each level by adding a complex scalar. As is the case for other (standard) preconditioners [39, 48] this strategy also has the effect of improving robustness while decreasing the fill-in required by GMSLR, especially for highly indefinite problems such that those arising from Helmholtz problems.

We note at this point that our focus is on a purely algebraic viewpoint where A is a general sparse indefinite matrix that does not necessarily originate from the discretization of a Partial Differential Equation. Therefore, we do not consider approaches based on hierarchical matrices.

This paper is organized as follows. In Section 2 we briefly review the HID ordering. Section 3 has a brief overview of block preconditioning that motivates the need for the low rank property of the inverse of the Schur complement. The details of the Schur complement approximation are given in Section 4. In Section 5 we present the preconditioner construction process. A two level analysis of the preconditioned eigenvalues is presented in Section 6. Then, in Section 7, we present some numerical

results from test problems and problems from the SuiteSparse matrix collection [17]. Concluding remarks and some ideas for future work can be found in Section 8.

2. HID ordering. Reordering the original system matrix A is essential for the performance of direct as well as iterative methods [8, 31, 38, 43]. GMSLR uses one such reordering technique known as the *Hierarchical Interface Decomposition (HID)* [25]. This ordering has also been used in the context of hierarchical linear system solvers [5] but is applicable to a wide class of sparse matrices, not just those that originate from PDEs. An HID ordering can be obtained in a number of ways. A particular method for obtaining such an ordering is the well known nested dissection method [21]. Nested dissection recursively partitions the adjacency graph of A into two disjoint subgraphs and a *vertex separator* in such a way that the removal of the vertices of the separator from the original graph results in the two disjoint subgraphs. Each level of bisection produces a new separator and new subgraphs. This level information can be represented by an HID tree $\mathcal{T}$. The matrix itself is reordered according to level, starting with level 0 and ending with level $L-1$.

Since we assume that A is large, sparse, and nonsymmetric, then an HID ordering has the multilevel, recursive structure

$$A_l = \begin{pmatrix} B_l & F_l \\ E_l & C_l \end{pmatrix} \text{ and } C_l = A_{l+1} \text{ for } l = 0 : L-1. \tag{2.1}$$

In this notation, A_0 denotes the original matrix A after HID ordering whereas A_L is the submatrix associated with the L^{th} level connector. The B_l block itself has a block diagonal structure due to the block independent set ordering [43], making solves with B_l ideally suited for parallel computation. Figure 2.1 shows an example of the HID ordering for a 3D convection-diffusion operator discretized with the standard 7-point finite difference stencil. The left subfigure plots the non-zero pattern of the entire matrix. The right subfigure is a close-up view of the non-zero pattern of the coupling of the interface variables at the root level.

3. Block Preconditioning. Domain decomposition reordering gives rise to linear systems of the form

$$A = \begin{pmatrix} B & F \\ E & C \end{pmatrix}, \tag{3.1}$$

see [2, 11]. Similar block structured matrices also arise from the discretization of systems of partial differential equations. In these coupled systems, the individual blocks usually correspond to differential/integral operators, however in this context they represent different sets of unknowns (interior, interface, coupling) that result from domain decomposition. There is a large body of work on preconditioning these systems mostly from the point of view of saddle point systems, see [6, 7, 28, 36, 37]. For examples of preconditioning other coupled systems of PDEs, see [14, 26, 27].

At the initial (root) level GMSLR uses a block triangular preconditioner of the form

$$\mathcal{P} = \begin{pmatrix} \widetilde{B}_0 & F_0 \\ 0 & \widetilde{S}_0 \end{pmatrix} \tag{3.2}$$

where $\widetilde{B}_0$ is an approximation to the $(1,1)$ block of A_0 and $\widetilde{S}_0$ is an approximation to the Schur complement $S_0 = C_0 - E_0 B_0^{-1} F_0$.

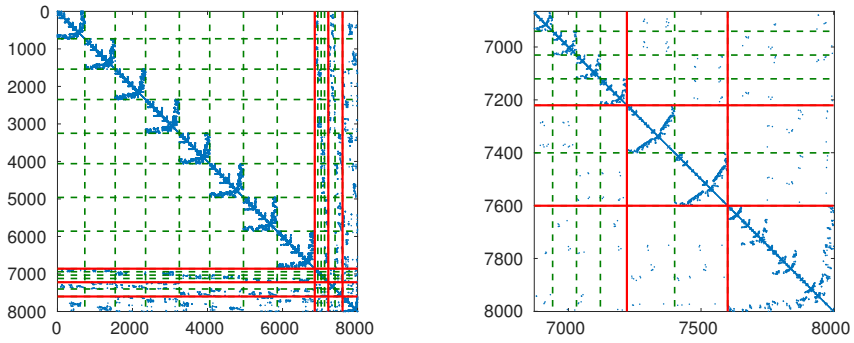

Fig. 2.1: *A 4-level HID ordered 3D convection-diffusion matrix with zero Dirichlet boundary conditions. The red (solid) lines separate the different levels. The green (dashed) lines separate subdomains located at the same level. Left: The original matrix is discretized on a* $32 \times 32 \times 32$ *regular grid with the standard 7-point stencil. Right: close-up view of the non-zero pattern of the coupling of the interface variables at the root level.*

In the ideal case where $\widetilde{B}_0 = B_0$ and $\widetilde{S}_0 = S_0$, it is well known that the matrix $A_0\mathcal{P}_{\text{ideal}}^{-1}$ has a quadratic minimal polynomial, which means that GMRES will converge in two iterations [28, 37]. Therefore the total cost of the procedure based on the ideal form of (3.2) is 2 linear solves with B_0 and two linear solves with S_0 plus additional sparse matrix-vector products. This is made clear by looking at the factored form of $\mathcal{P}_{\text{ideal}}^{-1}$:

$$\mathcal{P}_{\text{ideal}}^{-1} = \begin{pmatrix} B_0 & F_0 \\ & S_0 \end{pmatrix}^{-1} = \begin{pmatrix} B_0^{-1} & \\ & I \end{pmatrix} \begin{pmatrix} I & -F_0 \\ & I \end{pmatrix} \begin{pmatrix} I & \\ & S_0^{-1} \end{pmatrix}. \tag{3.3}$$

This choice corresponds to using only the upper triangular part of the block LU factorization of A_0 as a preconditioner. If both parts of this factorization are used, i.e., if our preconditioner is of the form

$$\mathcal{P}^{-1} = \begin{pmatrix} B_0^{-1} & \\ & I \end{pmatrix} \begin{pmatrix} I & -F_0 \\ & I \end{pmatrix} \begin{pmatrix} I & \\ & S_0^{-1} \end{pmatrix} \begin{pmatrix} I & \\ -E_0B_0^{-1} & I \end{pmatrix}, \tag{3.4}$$

then in the ideal case we have an exact inverse of A_0 and a Krylov method will converge in a single iteration at the total cost of two solves with B_0 and one solve with S_0. Thus, in all, using (3.4) saves one S_0 solve over (3.3).

The scenario just described involves ideal preconditioners (3.3) and (3.4) which are however not practical since they involve the exact computation of S_0^{-1}. In practice, $\widetilde{B}_0$ and $\widetilde{S}_0$ are approximated, at the cost of a few extra outer iterations. With these approximations in place it turns out that there is little difference in practice between these two options and, based on our experience, we prefer to use (3.2). This issue will be revisited at the end of Section 7.1.1.

Similar to [34], we solve linear systems with the B blocks by using incomplete LU (ILU) factorizations. Approximations to the Schur complement are typically tailored specifically to the problem being studied (e.g. the pressure convection diffusion [19] and least-squares commutator [18] preconditioners for Navier-Stokes). However, in

our framework, the block form of A is the result of a reordering of the unknowns and so our Schur complement approximation is inherently algebraic and not based on the physics of the problem. We base our Schur complement approximation on ideas from [34, 46].

4. Schur complement approximation. GMSLR is an extension of the MSLR preconditioner of [46] based on approximating the block $\mathcal{LDU}$ factorization of (2.1):

$$A_l = \begin{pmatrix} I & \\ E_l B_l^{-1} & I \end{pmatrix} \begin{pmatrix} B_l & \\ & S_l \end{pmatrix} \begin{pmatrix} I & B_l^{-1} F_l \\ & I \end{pmatrix} \tag{4.1}$$

at every level $l = 0, \ldots L-1$. We write the Schur complement as

$$S_l = \left(I - E_l B_l^{-1} F_l C_l^{-1}\right) C_l \equiv \left(I - G_l\right) C_l. \tag{4.2}$$

Let the *complex* Schur decomposition of G_l be

$$G_l = E_l B_l^{-1} F_l C_l^{-1} = W_l R_l W_l^H \tag{4.3}$$

where W_l is unitary and R_l is an upper triangular matrix whose diagonal contains the eigenvalues of G_l. Substituting (4.3) into (4.2) we get that

$$S_l = \left(I - W_l R_l W_l^H\right) C_l = W_l \left(I - R_l\right) W_l^H C_l. \tag{4.4}$$

Then, the Sherman-Morrison-Woodbury formula yields the inverse of S_l:

$$S_l^{-1} = C_l^{-1} W_l (I - R_l)^{-1} W_l^H = C_l^{-1} \left[I + W_l ((I - R_l)^{-1} - I) W_l^H\right] \tag{4.5}$$

which reduces to

$$S_l^{-1} = C_l^{-1} + C_l^{-1} W_l \left[(I - R_l)^{-1} - I\right] W_l^H. \tag{4.6}$$

Some observations about the matrix $S_l^{-1} - C_l^{-1}$ will be stated in the next section.

In our algorithm, we do not compute the full Schur decomposition of G_l, just the $k_l \times k_l$ leading submatrix of R_l and the first k_l Schur vectors. These choices give rise to the following inverse Schur complement approximation.

DEFINITION 4.1. *Let $G_l = E_l B_l^{-1} F_l C_l^{-1}, l = 0 \ldots L-1$ and $G_l = W_l R_l W_l^H$ be its Schur decomposition at level l. Let W_{l,k_l} be the matrix of the first k_l Schur vectors ($k_l < s$) of W_l. If we define R_{l,k_l} to be the $k_l \times k_l$ leading principal submatrix of R_l, then the approximate l^{th} level inverse Schur complement $\widetilde{S}_{l,k_l}^{-1}$ is given by*

$$\widetilde{S}_{l,k_l}^{-1} = C_l^{-1}(I + W_{l,k_l} H_{l,k_l} W_{l,k_l}^H). \tag{4.7}$$

where

$$H_{l,k_l} = [(I - R_{l,k_l})^{-1} - I]. \tag{4.8}$$

The inverse Schur complement approximation in (4.7) will be used at every level $l = 0, \ldots, L-1$. Due to the potential large size of the C_l blocks, we can only afford to factor C_{L-1} (i.e., at the top level) since it is the smallest of all the C_l blocks. For $l \neq L-1$ we use a slightly modified version of the recursive scheme of [46] for approximating the action of C_l^{-1} on a vector. The details of this approximation will be shown in Section 5.

4.1. Low rank property of $S_l^{-1} - C_l^{-1}$. Consider the inverse Schur complement formula given by (4.6). In this section we claim that for certain problems, the matrix $S_l^{-1} - C_l^{-1}$ is of low rank. If this is the case, then (4.7) will be a good approximation to (4.6). The only assumption we make on the blocks B_l, C_l is that they have LU factorizations, i.e.,

$$B_l = L_{B_l} U_{B_l},\ C_l = L_{C_l} U_{C_l}. \tag{4.9}$$

In practice we will use incomplete LU factorizations, so instead

$$B_l \approx L_{B_l} U_{B_l},\ C_l \approx L_{C_l} U_{C_l}.$$

Note that for large, 3D problems, the number of interface points (i.e., the size of the C_l block) can be quite large, making this factorization too costly. This is part of the motivation for the multilevel decomposition.

To see that $S_l^{-1} - C_l^{-1}$ is usually of low rank, again define the matrix G_l by

$$G_l = E_l B_l^{-1} F_l C_l^{-1} = (C_l - S_l)\, C_l^{-1}. \tag{4.10}$$

Let $\gamma_i, i = 1, \ldots, s$ be the eigenvalues of G_l (and also R_l) and define $X_l \equiv C_l(S_l^{-1} - C_l^{-1})$. By equation (4.6) the eigenvalues $\theta_1, \theta_2, \cdots, \theta_{s-1}, \theta_s$ of X_l are given explicitly by

$$\theta_i = \frac{\gamma_i}{1 - \gamma_i},\ i = 1, \ldots s \tag{4.11}$$

since $(I - G_l)^{-1} - I = G_l(I - G_l)^{-1}$.

As long as the eigenvalues γ_i of G_l are not clustered at 1, the eigenvalues θ_i of X_l will be well separated. This in turn means that $S_l^{-1} - C_l^{-1}$ can be approximated by a low rank matrix. This was studied in detail in [46, Section 2] for the symmetric case, where a theoretical bound for the numerical rank was established.

4.2. Building the low rank correction. We use Arnoldi's method [1] to build the low rank correction matrices in (4.7). This approximation can be efficient if the desired eigenpairs of G_l are on the periphery of the spectrum. However, as we shall see in the numerical results, this is simply not the case for some of the more indefinite problems. A particular remedy is to take more steps of Arnoldi's method.

Taking m steps of Arnoldi's method on G_l yields the Krylov factorizations:

$$\begin{aligned} G_l U_m &= U_m H_m + h_{m+1,m} u_{m+1} e_m^T \\ U_m^T G_l U_m &= H_m \end{aligned}$$

where U_m is an orthonormal matrix and H_m is a Hessenberg matrix whose eigenvalues (also called *Ritz values*) are good estimates to the extreme eigenvalues of G_l. We then take the complex Schur factorization of H_m

$$Q^H H_m Q = T. \tag{4.12}$$

We can reorder the k_l eigenvalues closest to 1 we wish to deflate so that they appear as the first k_l diagonal entries of T [3, 44]. The low rank matrices in (4.7) are approximated by :

$$R_{l,k_l} \approx T_{1:k_l,1:k_l} \text{ and } W_{l,k_l} \approx U_m Q_{:,1:k_l}. \tag{4.13}$$

5. Preconditioner construction process. In this section we show how the low rank property discussed in the previous section is used to build an efficient preconditioner. The only assumption we make is that each of the B_l, C_l blocks are non-singular. This assumption is typically satisfied unless the original matrix has a block of all zeros (e.g. a saddle point system). At the end of this section we also present an analysis of the computational and memory costs of this preconditioner.

5.1. 3-level scheme. We illustrate the steps taken to solve $Ax = b$ with a 3-level example.

Step 0: Apply a 3-level HID ordering to the original matrix A and right hand side b. Call the resulting reordered matrix and right hand side A_0, b_0 respectively.

Step 1: At this level (only) we use the block triangular matrix

$$\mathcal{U}_0^{-1} = \begin{pmatrix} B_0 & F_0 \\ & S_0 \end{pmatrix}^{-1} = \begin{pmatrix} B_0^{-1} & \\ & I \end{pmatrix} \begin{pmatrix} I & -F_0 \\ & I \end{pmatrix} \begin{pmatrix} I & \\ & S_0^{-1} \end{pmatrix}$$

as a right preconditioner for A_0 i.e., we solve $A_0\mathcal{U}_0^{-1}u = b_0$. Here we approximately factor B_0 by ILU and approximate the Schur complement by

$$S_0^{-1} \approx \widetilde{S}_0^{-1} = C_0^{-1}(I + W_0 H_0 W_0^T)$$

where H_0 and W_0 are taken from (4.8) and (4.13) respectively. To solve with C_0, we refer to (2.1) and move from level 0 to level 1.

Step 2: At level 1, we have

$$C_0^{-1} = A_1^{-1} = \begin{pmatrix} I & -B_1^{-1}F_1 \\ & I \end{pmatrix} \begin{pmatrix} B_1^{-1} & \\ & S_1^{-1} \end{pmatrix} \begin{pmatrix} I & \\ -E_1B_1^{-1} & I \end{pmatrix}$$

where S_1^{-1} is approximated by C_1^{-1} plus a low rank correction:

$$S_1^{-1} \approx \widetilde{S}_1^{-1} = C_1^{-1}(I + W_1 H_1 W_1^T).$$

Next we move up a level again to define an approximate inverse for C_1, referring again to (2.1).

Step 3: At level 2 we have:

$$C_1^{-1} = A_2^{-1} = \begin{pmatrix} I & -B_2^{-1}F_2 \\ & I \end{pmatrix} \begin{pmatrix} B_2^{-1} & \\ & S_2^{-1} \end{pmatrix} \begin{pmatrix} I & \\ -E_2B_2^{-1} & I \end{pmatrix}.$$

Similarly to Step 2, we now approximate S_2^{-1} by C_2^{-1} plus a low rank correction term, i.e.,

$$S_2^{-1} \approx \widetilde{S}_2^{-1} = C_2^{-1}(I + W_2 H_2 W_2^T).$$

At this level, we decide that C_2 is sufficiently small and compute its ILU factorization: $C_2 \approx L_{C_2}U_{C_2}$.

In order to apply the preconditioner $\mathcal{U}_0^{-1}$, the actual algorithm starts at level 2 and proceeds up to level 0. For this particular example, that means we start forward-backward solving with the ILU factorization of C_2 since C_2^{-1} is needed in order to apply S_2^{-1}. Now that the action of S_2^{-1} is available we can then approximate A_2^{-1} and the pattern continues until we hit level 0, i.e.,

$$L_{C_2}U_{C_2} \to C_2^{-1} \to \widetilde{S}_2^{-1} \to A_2^{-1} \to \widetilde{S}_1^{-1} \to A_1^{-1} \to \widetilde{S}_0^{-1} \to \mathcal{U}_0^{-1}.$$

Once C_l^{-1} (or its action on a vector) is available, the low rank correction matrices W_l, H_l can be computed.

5.2. General Case. When computing the partial Schur decomposition of the matrix G_l, we need to be able to compute matrix vector products with the matrix $E_l B_l^{-1} F_l C_l^{-1}$ at each level l. We already have the factors of B_l, so any matrix-vector product with B_l^{-1} can be computed with one forward and one backward substitution. The same does not hold true for C_l, since we only compute its factorization at level $L-1$. However, we already have an approximate factorization of A_{l+1}^{-1} and since $C_l^{-1} = A_{l+1}^{-1}$ we can use this approximation to apply C_l^{-1} to a vector. The construction of the preconditioner is summarized in Algorithm 1. The details of the recursively defined product of C_l^{-1} with a vector b are given in Algorithm 2.

Algorithm 1 Generalized Multilevel Schur Low-Rank (Construction phase)

procedure GMSLR
  Apply an L-level reordering to A (A_0 = reordered matrix).
  for level l from $L-1$ to 0 **do**
    if $l = L-1$ **then**
      Compute ILU factorization of C_{L-1}, $C_{L-1} \approx L_{C_{L-1}} U_{C_{L-1}}$
    end if
    Compute ILU factorization of B_l, $B_l \approx L_{B_l} U_{B_l}$.
    Perform k_l steps of the Arnoldi process ▷ *Call Algorithm 2 to apply* C_l^{-1}

$$[V_l, K_l] = \text{Arnoldi}(E_l U_{B_l}^{-1} L_{B_l}^{-1} F_l C_l^{-1}, k_l)$$

    Compute the complex Schur decomposition $K_l = WTW^T$.
    Compute $W_{l,k_l} = V_l W$ and set $R_{l,k_l} = T_{1:k_l,1:k_l}$.
    Compute $H_l = (I - R_{l,k_l})^{-1} - I = R_{l,k_l}(I - R_{k_l})^{-1}$.
  end for
end procedure

Algorithm 2 Approximation of $y = C_l^{-1} b$ for $l \geq 1$ and $y = \mathcal{U}_0^{-1} b$

procedure `RecursiveSolve`(l, b)
  if $l = L-1$ **then**
    return $y = U_{C_{L-1}}^{-1} L_{C_{L-1}}^{-1} b$
  else
    Split $b = (b_1^T, b_2^T)^T$ conformingly with the blocking of C_l
    Compute $z_1 = U_{B_{l+1}}^{-1} L_{B_{l+1}}^{-1} b_1$
    Compute $z_2 = b_2 - E_{l+1} z_1$
    if $1 \leq l < L-1$ **then**
      Compute $w_2 = W_{l+1,k_{l+1}} H_{l+1} W_{l+1,k_{l+1}}^T z_2$
      Compute $y_2 =$ `RecursiveSolve`$(l+1, z_2 + w_2)$
      Compute $y_1 = z_1 - U_{B_{l+1}}^{-1} L_{B_{l+1}}^{-1} F_{l+1} y_2$
    else
      Solve the system $S_0 y_2 = z_2$ with $\widetilde{S}_0^{-1}$ as a right preconditioner
      Compute $y_1 = U_{B_0}^{-1} L_{B_0}^{-1} (b_1 - F_0 y_2)$
    end if
    return $y = (y_1^T, y_2^T)^T$
  end if
end procedure

Similarly to MSLR the HID ordering gives rise to B_l matrices that are block-diagonal in structure, and so all of these blocks can be factored in parallel. Furthermore, the triangular solves associated with B_l can also be done in parallel for each block. In addition, while Algorithm 2 generally provides an accurate approximation to C_l^{-1}, we must point out that due to the presence of the inner solve at level $l = 0$ (Line 13 of Algorithm 2), GMSLR is (potentially) more expensive per iteration than MSLR. This expense can be lessened somewhat by the fact that the inner solve can only require 1-2 digits of accuracy without radically affecting the convergence rate of the outer solve.

5.3. Computational and memory complexity of the preconditioner. Let $\text{mem}(\text{ILU}(B_l))$ denote the memory cost associated with the storage of the incomplete factorization of B_l. Then the total memory cost μ^L_{GMSLR} of the GMSLR preconditioner using L levels is:

$$\mu^L_{GMSLR} = \left(\sum_{l=0}^{L-1} \left[\text{mem}(\text{ILU}(B_l)) + \max\{2s_l k_l + k_l^2, 2s_l^2 + s_l^2\} \right] \right) + \text{mem}(\text{ILU}(C_{L-1})),$$

where the second term inside the summation accounts for the memory cost associated with the partial Schur decompositions of order $1 \leq k_l \leq s_l$ at levels $0 \leq l \leq L-1$, and s_l denotes the number of interface variables at level l, i.e., the leading dimension of each C_l. For simplicity, we treat the upper triangular matrix H_{l,k_l} as a dense matrix. In the case where the incomplete factorization of matrices $B_l,\ l = 0, \ldots, L-1$, and C_{L-1} are obtained by a thresholded version of ILU, with a maximum number of non-zero entries per row equal to τ, the above memory cost is bounded by

$$\mu^L_{GMSLR} \leq \left(\sum_{l=0}^{L-1} \left[2\tau d_l + \max\{2s_l k_l + k_l^2, 2s_l^2 + s_l^2\} \right] \right) + 2\tau s_{L-1},$$

where d_l denotes the leading dimension of B_l.

To obtain an estimate of the computational cost to apply the GMSLR preconditioner at the root level $l \equiv 0$, we need to consider the computational cost associated with all intermediate levels. Let $\text{trisol}(\text{ILU}(B_l))$ and $\text{trisol}(\text{ILU}(C_{L-1}))$ denote the cost of the triangular solves with B_l and C_{L-1} respectively and let $\gamma^{(L-1)}_{GMSLR}$ denote the cost associated with level $l = L-1$. At level $l = L-2$, the cost to apply the GMSLR preconditioner is equal to the sum of the cost to apply the preconditioner at level $l+1 = L-1$ and the cost $2\text{trisol}(\text{ILU}(B_{L-2})) + O(s_{L-2}k_{L-2})$, where we assumed that $k_{L-2} \ll s_{L-2}$. Continuing in the same spirit, we finally get that the cost to apply the GMSLR preconditioner at level l, $\gamma^{(l)}_{GMSLR}$, is equal to

$$\gamma^{(l)}_{GMSLR} = \gamma^{(l+1)}_{GMSLR} + 2\text{trisol}(\text{ILU}(B_l)) + O(s_l k_l),\ l = 0, \ldots, L-2,$$

where

$$\gamma^{(L-1)}_{GMSLR} = \text{trisol}(\text{ILU}(C_{L-1})) + 2\text{trisol}(\text{ILU}(B_{L-1})) + O(s_{L-1}k_{L-1}).$$

6. Eigenvalue Analysis. This section studies the spectra of linear systems preconditioned by GMSLR. We only consider a 2 level decomposition since the recursive nature of both algorithms makes the analysis difficult. In what follows, let $\widetilde{B}_0$ denote

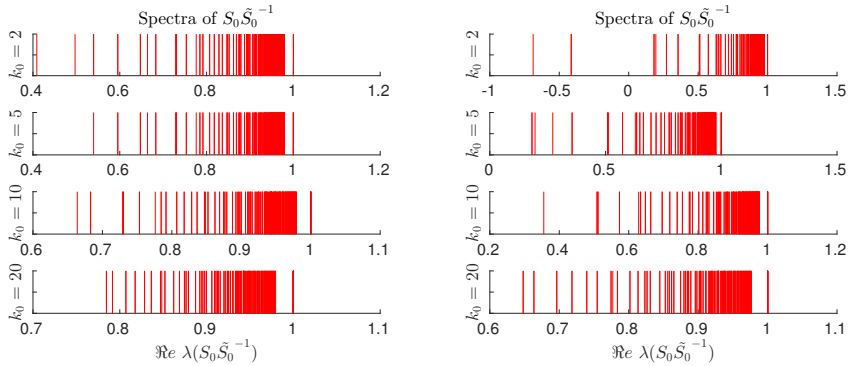

Fig. 6.1: *The spectrum of* $S_0\tilde{S}_0^{-1}$ *for different values of* $k_0 \equiv k_1$ *(*$L = 2$*). Left:* $c = 0.0$*. Right:* $c = 0.5$*.*

an approximation to B_0 and $\widetilde{S}_0$ the GMSLR approximation to the Schur complement $S_0 = C_0 - E_0 B_0^{-1} F_0$ respectively. GMSLR starts with a 2×2 block partition of the original matrix A, i.e.,

$$A_0 = \begin{pmatrix} B_0 & F_0 \\ E_0 & C_0 \end{pmatrix} \tag{6.1}$$

where B_0 is $n_B \times n_B$ and C_0 is $s \times s$.

As was already seen, the GMSLR preconditioner is based on the block-LU factorization of (6.1), so at level 0 we have

$$A_0 = \begin{pmatrix} B_0 & F_0 \\ E_0 & C_0 \end{pmatrix} = \begin{pmatrix} I & 0 \\ E_0 B_0^{-1} & I \end{pmatrix} \begin{pmatrix} B_0 & F_0 \\ 0 & S_0 \end{pmatrix} = \mathcal{L}_0 \mathcal{U}_0,$$

and the preconditioner $\widetilde{\mathcal{U}}_0^{-1}$ is

$$\widetilde{\mathcal{U}}_0^{-1} = \begin{pmatrix} \widetilde{B}_0^{-1} & 0 \\ 0 & I \end{pmatrix} \begin{pmatrix} I & -F_0 \\ 0 & I \end{pmatrix} \begin{pmatrix} I & 0 \\ 0 & \widetilde{S}_0^{-1} \end{pmatrix}.$$

A simple calculation shows that

$$A_0 \widetilde{\mathcal{U}}_0^{-1} = \begin{pmatrix} B_0 \widetilde{B}_0^{-1} & (I - B_0 \widetilde{B}_0^{-1}) F_0 \widetilde{S}_0^{-1} \\ E_0 \widetilde{B}_0^{-1} & S_0 \widetilde{S}_0^{-1} \end{pmatrix}. \tag{6.2}$$

If we assume that $\widetilde{B}_0 = B_0$, then (6.2) simplifies to

$$A_0 \widetilde{\mathcal{U}}_0^{-1} = \begin{pmatrix} I & 0 \\ E_0 B_0^{-1} & S_0 \widetilde{S}_0^{-1} \end{pmatrix}, \tag{6.3}$$

which has eigenvalues $\lambda(A_0 \widetilde{\mathcal{U}}_0^{-1}) = \{1, \lambda(S_0 \widetilde{S}_0^{-1})\}$.

Convergence will be rapid if the eigenvalues of $S_0 \widetilde{S}_0^{-1}$ are also close to 1. To illustrate the influence the rank has on convergence, we show in Figure 6.1 the spectra of $S_0 \tilde{S}_0^{-1}$ for a small test problem. Here A is the discretized shifted Laplacian operator $-\Delta u - cu = f$ with $c = 0.0$ (left figure) and $c = 0.5$ (right figure) and homogeneous

Dirichlet boundary conditions. For reference, when $c = 0.5$, this 8000×8000 matrix has 35 negative eigenvalues. This is a matrix selected for illustrative purposes, so we use two levels with equal ranks k_0, k_1 and compute the exact LU factorization of B_0. As the ranks k_0, k_1 increase, the real part of the eigenvalues of $S_0\tilde{S}_0^{-1}$ clusters more tightly around 1.

7. Numerical experiments. All experiments were run on a single node of the `Mesabi` Linux cluster at the Minnesota Supercomputing Institute. This node has a memory of 64 GBs and consists of two sockets each having a twelve core 2.5 GHz Intel Haswell processor. The GMSLR preconditioner was written in C++ and compiled by Intel's C++ compiler using `-O3` optimization. Simple thread-level parallelism was achieved with OpenMP with a maximum of 24 threads. The B_l blocks are factored by the ILUT routine from ITSOL. The Intel Math Kernel Library (MKL) was used for the BLAS and LAPACK routines. We use flexible GMRES [40] with a fixed restart size of 40 as the outer solver, denoted by GMRES(40). The inner solve in step 14 of Algorithm 2 is also done with FGMRES. Unless otherwise noted, we follow the methodology of [32, 41, 46] where the right hand side vector b is given by $Ae = b$ where e is the vector of all ones.

The HID ordering was obtained by the function `PartGraphRecursive` from the METIS [30] package. The diagonal blocks of each $B_l, C_l,\ l = 0, \ldots, L-1$, were reordered using the approximate minimum degree (AMD) ordering [15, 16] in order to reduce the fill-in generated by their ILU factorizations. In our experiments the reported preconditioner construction time comes from the factorization of the B_l blocks and the computation of the low rank correction matrices. The reordering time is regarded as preprocessing and is therefore not reported. Similarly, the iteration time is the combined time spent on the inner and outer solves.

The parameters we are most interested in varying are: the number L of levels in the HID and the maximum rank used in the low rank correction, i.e., the number of steps of Arnoldi's method (see Section 4.2), denoted by rk. In particular we set k_l in (4.13) to be equal to rk for any $l = 0, \ldots, L-1$, and thus all Arnoldi vectors are included in the low rank correction terms.

We use the following notation in the results that follow:

- fill $= \frac{nnz(prec)}{nnz(A)}$, where nnz denotes the number of non-zero entries of the input matrix;
- p-t: wall clock time to build the preconditioner (in seconds);
- its: number of *outer* iterations of preconditioned GMRES(40) required for $\|r_k\|_2 < 10^{-6}$. We use "`F`" to indicate that GMRES(40) did not converge after 500 iterations;
- i-t: wall clock time for the iteration phase of the solver.
- rk: max rank used in building the low rank corrections.

The value of $nnz(prec)$ is the sum of the non-zero entries associated with the incomplete factorizations (ILU) and Low-Rank Correction (LRC) terms. These quantities are computed as $\sum_{l=0}^{L-1}\left[(nnz(U_{B_l}) + nnz(L_{B_l})\right] + \left(nnz(U_{C_{L-1}}) + nnz(L_{C_{L-1}})\right)$ for ILU, and $\sum_{l=0}^{L-1}\left[2d_l\mathrm{rk} + \mathrm{rk}^2\right]$ for LRC, respectively.

7.1. Problem 1. We begin our tests with the symmetric indefinite problem:

$$\begin{aligned} -\Delta u - cu &= f \quad \text{in } \Omega, \\ u &= 0 \quad \text{on } \partial\Omega, \end{aligned} \tag{7.1}$$

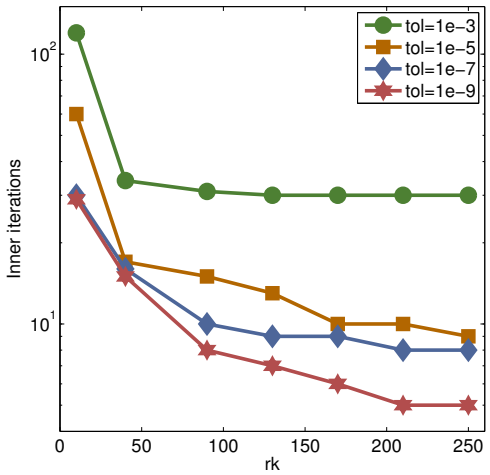

Fig. 7.1: *Number of inner iterations in GMSLR as a function of* rk *for different values of the drop tolerance* tol *in the incomplete factorizations. We set* $L = 2$.

where $\Omega = (0,1)^3$. The discretization is via finite differences with the standard 7-point stencil in 3D. This test problem is useful for testing robustness with respect to definiteness. For reference, GMRES preconditioned by standard AMG fails to converge when applied to (7.1) with even a small positive shift on a 32×32 regular mesh.

7.1.1. Varying the number of levels. First, we study the effect of adding more levels to the preconditioner. We solve (7.1) with $c > 0$ in order to make the problem indefinite. In the cases where $c > 0$, we shift the diescretized Laplacian operator by sI, where $s = h^2 c$ for mesh size h. For this first example, we set $s = 0.5$. The associated coefficient matrix has 163 negative eigenvalues. The maximum rank was fixed at 50. By Table 7.1 we can see that, as L grows larger, the ILU fill-factor decreases monotonically while the low rank correction fill-factor increases monotonically. The optimal number of levels occurs when these two quantities are roughly equal. For this particular example, we pick $L_{opt} = 6$ as it strikes the right balance of fill, iteration count, and total computational time. Note that as L increases, so does the number of interface variables at the root level, s_0. This can be verified immediately by looking at Table 7.2 where we list s_0 for all values of L from $L = 2$ to $L = 6$. Figure 7.1 plots the number of inner iterations performed by GMSLR as a function of the rank rk for different values of the drop tolerance (denoted by tol) in the ILU factorizations.

Finally, recall that we could have used the inexact version of (3.4) instead of (3.2). For SPD problems there is not a significant difference in the results obtained by either preconditioner. However, as shown in Table 7.2, for an indefinite problem such as (7.1) with $s = 0.5$, (3.2) performs better. The likely explanation for this behavior is that (3.2) involves fewer solves with the B_l matrices which are highly indefinite and therefore admit poor ILU factorizations.

7.1.2. Varying the maximum rank in the low rank corrections. Next, we keep the number of levels fixed, but increase the maximum rank. We again solve (7.1)

Table 7.1: *The fill factor and iteration counts for solving (7.1) with $s = 0.5$ on a 32^3 grid with the FGMRES-GMSLR method. Here, the maximum rank for the LRC matrices was fixed at 50.*

L	ILU fill	LRC fill	fill	p-t	i-t	its
2	34.61	.23	34.84	5.16	1.25	16
3	21.03	.68	21.71	.986	2.69	16
4	15.64	1.35	16.99	.382	1.03	12
5	8.69	2.46	11.15	.169	.97	19
6	5.56	3.96	9.52	.172	.95	17

Table 7.2: *Comparison between GMSLR with only using $\mathcal{U}_0^{-1}$ and GMSLR with $\mathcal{L}_0^{-1}$ and $\mathcal{U}_0^{-1}$ on (7.1) with $s = 0.5$ on a 32^3 grid. The maximum rank was fixed at 50.*

		GMSLR - $\mathcal{U}_0^{-1}$ only			GMSLR - $\mathcal{U}_0^{-1}\mathcal{L}_0^{-1}$		
L	s_0	p-t	i-t	its	p-t	i-t	its
2	1,024	5.16	1.25	16	5.15	3.59	47
3	2,016	.986	2.69	16	1.01	5.24	37
4	2,977	.382	1.03	12	.391	2.88	34
5	4,955	.169	.97	19	.181	1.49	27
6	6,699	.172	.95	17	.176	1.43	24

with $s = 0.5$ discretized on a 32^3 regular grid. The ILU fill factor is constant because we are keeping the number of levels fixed at 6. The fill factor from the low rank corrections increases at an almost constant rate. Increasing the maximum rank has the unfortunate effect of increasing the fill-factor and the preconditioner construction time. As we see in Table 7.3, the effect of increasing the rank (at least for this model problem) is difficult to predict. As a general rule, it seems as though a large maximum rank is unavoidable for highly indefinite problems.

Table 7.3: *Iteration counts for solving (7.1) with $s = 0.5$ on a 32^3 grid with the FGMRES-GMSLR method. The number of levels was fixed at 6.*

rk	ILU fill	LRC fill	fill	p-t	i-t	its
20	5.56	1.58	7.14	.091	1.34	24
30	5.56	2.37	7.93	.118	1.14	19
40	5.56	3.17	8.73	.139	1.04	18
50	5.56	3.96	9.52	.174	.972	17
60	5.56	4.75	10.31	.208	1.29	22
70	5.56	5.24	10.8	.221	1.35	24
80	5.56	5.99	11.55	.291	.968	15

7.1.3. Increasingly indefinite problems. The model problem (7.1) becomes significantly more difficult to solve as s increases. Here, we increase s from 0 to 1 while tuning the maximum rank and number of levels to compensate for solving this increasingly difficult problem. We report the results that give the best balance

between iteration count and fill in Table 7.4. The fill factor increases dramatically for two reasons: first, we must increase the rank of the low rank correction and second, we must keep the number of levels low, which, as was observed in Section 7.1.1, leads to increased fill-in for the same drop tolerance. If the rank is too low or the number of levels is too high, GMRES(40) simply will not converge. Recall that the construction of the low rank correction is based on finding approximate eigenvalues of the matrix $E_l U_{B_l}^{-1} L_{B_l}^{-1} F_l C_l^{-1}$ using Arnoldi's method. When B_0 is indefinite, as is the case here, the eigenvalues we seek get pushed deeper inside the spectrum, i.e. they become interior eigenvalues. Since the Arnoldi process does a poorer job for these interior eigenvalues than it does for extreme ones, we are forced to take more Arnoldi steps in order to approximate them.

Table 7.4: *Results of solving symmetric linear systems with increasing shift values s on a* 32^3 *regular mesh with GMSLR.*

s	L	max rank	fill	p-t	i-t	its
0	8	20	5.89	.109	.068	3
.25	6	30	7.59	.117	.449	8
.5	6	50	9.52	.174	.973	17
.75	5	80	12.77	.291	.826	13
1.0	5	120	13.73	.406	1.87	29

7.2. Problem 2. The second problem of interest is nonsymmetric:

$$\begin{aligned} -\Delta u - \alpha \cdot \nabla u - cu &= f \quad \text{in } \Omega, \\ u &= 0 \quad \text{on } \partial\Omega, \end{aligned} \tag{7.2}$$

where $\Omega = (0,1)^3, \alpha \in \mathbb{R}^3$. This problem is simply a shifted convection-diffusion equation, again discretized by the 7-point finite difference stencil. As before we shift the discretized convection-diffusion operator by sI where $s = h^2 c$.

7.2.1. Varying the number of levels. In this next set of experiments we fix $\alpha = [.1, .1, .1]$ and solve (7.2) in 3D with no shift and then with a shift of $s = .25$. As before, we start by increasing the number of levels. The results of the first problem with a maximum rank of 20 are in Table 7.5. These results are comparable to those obtained from the SPD problem (7.1) with $s = 0$, i.e., for this problem, the convergence rate is not adversely affected by the loss of symmetry.

Next, we solve (7.2) with $s = .25$. The shift significantly increases the number of eigenvalues with negative real parts, so we increase the maximum rank to 50. The results can be found in Table 7.6. It is interesting to note that the fill from the low rank correction is almost exactly the same as in Table 7.1. This is due to the fact that both problems used a maximum rank of 50 to build the low rank corrections.

7.3. Problem 3. The third model problem is a Helmholtz equation of the form

$$\left(-\Delta - \frac{\omega^2}{v(x)^2}\right) u(x,\omega) = s(x,\omega). \tag{7.3}$$

In this formulation, Δ is the Laplacian operator, ω the angular frequency, $v(x)$ the velocity field, and $s(x,\omega)$ is the external forcing function with corresponding time-harmonic wave field solution $u(x,\omega)$. The computational domain is the unit cube

Table 7.5: *The fill factor and iteration counts for solving (7.2) with no shift and* $\alpha = [.1, .1, .1]$ *on a* 32^3 *grid with the FGMRES-GMSLR method. Here, the maximum rank for the LRC matrices was fixed at 20.*

L	ILU fill	LRC fill	fill	p-t	i-t	its
2	11.69	.092	11.78	.505	.159	7
3	10.13	.272	10.4	.234	.079	6
4	8.8	.539	9.34	.126	.044	5
5	6.47	.983	7.46	.09	.041	5
6	4.89	1.58	6.47	.086	.074	4
7	3.8	2.34	6.14	.092	.066	4
8	2.53	3.35	5.88	.116	.066	3

Table 7.6: *The fill factor and iteration counts for solving (7.2) with* $s = .25$ *and* $\alpha = [.1, .1, .1]$ *on a* 32^3 *grid with the FGMRES-GMSLR method. Here, the maximum rank for the LRC matrices was fixed at 50.*

L	ILU fill	LRC fill	fill	p-t	i-t	its
2	24.11	.23	24.34	2.03	.88	16
3	15.44	.681	16.12	.58	.61	13
4	11.64	1.35	12.99	.237	.381	12
5	7.25	2.46	9.71	.149	.91	19
6	5.16	3.96	9.12	.167	.741	13
7	3.91	5.86	9.77	.214	1.00	14
8	2.56	8.39	10.95	.288	4.54	53

$\Omega = (0, 1)^3$ where we again use the seven-point finite difference discretization on a regular mesh. The Perfectly Matched Layer (PML) boundary condition is used on all faces of Ω. The resulting linear systems are complex non-Hermitian. If we assume that the mean of $v(x)$ is 1 in (7.3), then the wave number is $\omega/(2\pi)$ and $\lambda = 2\pi/\omega$ is the wavelength. The number of grid points in each dimension is $N = q\omega/(2\pi)$ where q is the number of points per wavelength. As a result, the discretized system is of size $n = N^3 \times N^3$.

Table 7.7: *Results from solving (7.3) on a sequence of 3D meshes with GMSLR. All problems have* $q = 8$ *points per wavelength. The second set of results is with a small complex shift added to the* B_ℓ *matrices.*

				GMSLR - no shift				GMSLR w/ complex shift			
$\omega/(2\pi)$	$n = N^3$	L	rk	fill	p-t	i-t	its	fill	p-t	i-t	its
2.5	20^3	5	16	3.79	.063	.318	14	3.56	.062	.205	17
3	30^3	6	16	5.18	.156	.308	13	4.72	.135	.547	16
5	40^3	7	16	6.19	.282	1.94	57	5.43	.251	.556	17
6	50^3	7	16	8.16	.768	3.52	54	6.64	0.54	1.3	21
8	60^3	8	16	7.73	.867	29.53	F	6.52	0.73	2.05	21
10	80^3	9	16	7.85	1.57	65.57	F	6.64	1.4	6.31	28

We test the performance of the GMSLR preconditioner on 6 cubes, setting $q = 8$, and report the results in Table 7.7. Since q is fixed, an increase in the wave number means an increase in N, so the higher frequency problems lead to much larger linear systems. In these experiments, we set the inner solve tolerance to 10^{-2} or a maximum of 10 iterations. Results reported under the legend "GMSLR - no shift" stand for the regular GMSLR preconditioner. Results reported under the legend "GMSLR w/ complex shift" stand for runs where the GMSLR preconditioner was built by first shifting B_l by $\sigma = \left(\sum_{j=1}^{d_l}(B_l)_{jj}/d_l\right) * .05 * i$. Without a complex shift, these problems can be much more difficult, especially as the matrix size grows. Indeed, for the last two examples no convergence was achieved after 300 outer iterations. On the other hand, the shift benefits all test problems as it allows for an increased number of levels (and thus less fill-in introduced by ILU) while also keeping the number of outer iterations relatively small (the number of outer iterations only increased from 17 to 28 as the matrix size grew from 20^3 to 80^3).

7.4. General sparse matrices. To further illustrate the robustness of the GMSLR preconditioner, we tested it on several large, nonsymmetric matrices from the SuiteSparse Matrix Collection [17]. These matrices come from a wide range of application areas, not just PDEs. As a benchmark, we also tested ILUT for these nonsymmetric matrices. Information about the matrices is shown in Table 7.8. Table 7.9 shows the results of these experiments. The ILUT parameters were chosen such that the fill of both methods was comparable.

Table 7.8: *Set of nonsymmetric test matrices from the SuiteSparse Matrix Collection where nnz is the number of nonzero entries in the matrix.*

Matrix	Order	nnz	SPD	Origin
`cbuckle`	13,681	676,515	yes	structural problem
`epb2`	25,228	175,027	no	thermal problem
`wang4`	26,068	177,196	no	semiconductor device problem
`barrier2-1`	113,076	3,805,068	no	semiconductor device problem
`Cage12`	130,228	2,032,536	no	directed weighted graph
`offshore`	259,789	4,242,673	yes	electromagnetics problem
`CoupCons`	416,800	22,322,336	no	structural problem
`AtmosModd`	1,270,432	8,814,880	no	atmospheric model
`AtmosModL`	1,489,752	10,319,760	no	atmospheric model
`Cage14`	1,505,785	27,130,349	no	directed weighted graph
`Transport`	1,602,111	23,500,731	no	CFD problem

Results are shown in Table 7.9, where `F` indicates a failure to converge in 500 iterations. As can be seen, for these problems, GMSLR is superior to ILUT. It is worth adding that ILUT is a highly sequential preconditioner both in its construction and its application. In contrast, GMSLR is by design a domain decomposition-type preconditioner that offers potential for excellent parallelism.

Figure 7.2 plots the value of i-t and p-t as both L and the drop tolerance "tol" of the incomplete factorizations are varied for matrices "barrier2-1" and "offshore". In agreement with the results reported so far, an increase in the value of L reduces the time to construct and apply the preconditioner. On the other hand, an increase in the value of L might also lead to a larger number of inner iterations necessary to

Table 7.9: *Comparison between GMSLR and ILUT preconditioners for solving the above problems. ILUT parameters were chosen so that the fill factor was close to that of GMSLR. Both sets of tests use the same reordered matrix.*

Matrix	GMSLR						ILUT			
	fill	L	rk	p-t	i-t	its	fill	p-t	i-t	its
`cbuckle`	2.13	5	5	.22	.10	9	2.27	.38	.32	22
`epb2`	3.63	5	15	.23	.13	4	3.43	.98	.76	19
`wang4`	4.83	2	35	.14	.08	13	4.92	.45	.41	18
`barrier2-1`	3.72	5	10	.60	1.91	6	3.69	44.19	14.32	F
`Cage12`	0.95	5	25	.23	.28	4	1.00	.24	.31	5
`offshore`	0.99	12	35	1.27	2.35	5	1.09	1.02	1.60	10
`CoupCons`	1.82	10	16	1.68	.64	5	1.64	17.49	2.03	23
`AtmosModd`	5.86	10	4	1.23	3.05	11	5.68	8.10	8.60	47
`AtmosModL`	5.81	11	4	1.67	2.12	7	6.03	11.35	6.37	30
`Cage14`	1.54	6	4	3.10	.89	4	1.57	5.09	0.70	4
`Transport`	2.52	11	4	1.85	7.45	23	2.59	27.91	59.7	116

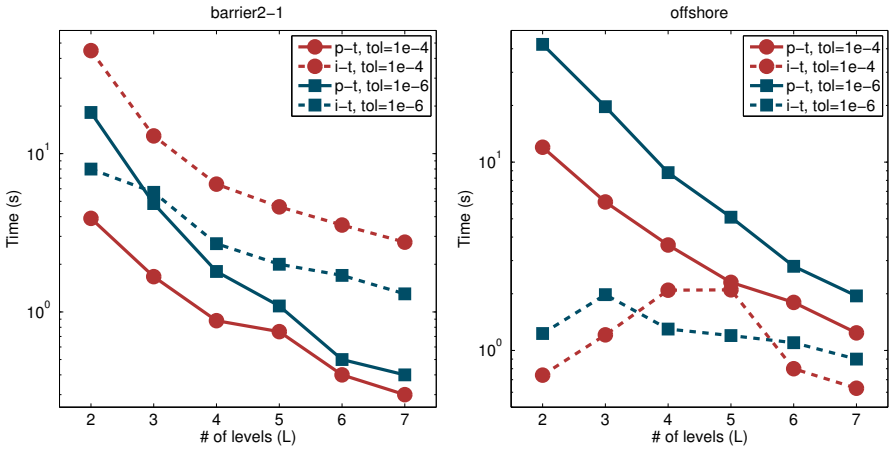

Fig. 7.2: *Amount of time spent on building the GMSLR preconditioner and solving the linear system as the number of levels L and the drop tolerance* tol *in the incomplete factorizations vary. The rank rk was fixed to 10.*

achieve convergence, and thus might lead to higher iteration times.

8. Conclusion. The GMSLR preconditioner combines several ideas. First is the HID ordering method, which has a recursive multilevel structure. The $(1,1)$ block of each level of this structure is block diagonal, which means that solves with this block are easily parallelizable. Motivated by the block LU factorization of the reordered matrix, we use a block triangular preconditioner at the bottom level of the HID tree. For the other levels, we use approximate inverse factorizations exploiting a recursive relationship between the different levels. Finally, we approximate the inverse Schur complement at each level of the HID tree via a low rank correction technique.

Because it is essentially an approximate inverse preconditioner, GMSLR is capable of solving a wide range of highly indefinite problems that would be difficult to solve by standard methods such as ILU. The numerical experiments we showed confirm this. Additional benefits of GMSLR include its inherent parallelism and its fast construction.

GMSLR is also promising for use in eigenvalue computations, especially in the context of rational filtering eigenvalue solvers where complex, indefinite linear systems need be solved [29, 47]. The factorization of these systems can be slow and costly for large 3D problems. We plan on investigating the use of Krylov subspace methods preconditioned by GMSLR to solve such systems. Among other objectives, we also plan to implement and publicly release a fully parallel, domain-decomposition based version of GMSLR.

Acknowledgements. The authors would like to thank the Minnesota Supercomputing Institute for the use of their extensive computing resources and the anonymous referees for their careful reading of this paper and helpful suggestions.

REFERENCES

[1] W. E. Arnoldi, *The principle of minimized iterations in the solution of the matrix eigenvalue problem*, Quarterly of Applied Mathematics, 9 (1951), pp. 17–29.
[2] O. Axelsson and B. Polman, *Block preconditioning and domain decomposition methods II*, Journal of Computational and Applied Mathematics, 24 (1988), pp. 55–72.
[3] Z. Bai and J.W. Demmel, *On swapping diagonal blocks in real Schur form*, Linear Algebra and its Applications, 186 (1993), pp. 75–95.
[4] M. Bebendorf, *Why finite element discretizations can be factored by triangular hierarchical matrices*, SIAM Journal on Numerical Analysis, 45 (2007), pp. 1472–1494.
[5] M. Bebendorf and T. Fischer, *On the purely algebraic data-sparse approximation of the inverse and the triangular factors of sparse matrices*, Numerical Linear Algebra with Applications, 18 (2011), pp. 105–122.
[6] M. Benzi, *Preconditioning techniques for large linear systems: A survey*, J. of Computational Physics, 182 (2002), pp. 418–477.
[7] M. Benzi, G. H. Golub, and J. Liesen, *Numerical solution of saddle point problems*, Acta Numerica, 14 (2005), pp. 1–137.
[8] M. Benzi, D. Szyld, and A. Van Duin, *Orderings for incomplete factorization preconditioning of nonsymmetric problems*, SIAM J. Sci. Comput., 20 (1999), pp. 1652–1670.
[9] M. Benzi and M. Tůma, *A sparse approximate inverse preconditioner for nonsymmetric linear systems*, SIAM J. Sci. Comput., 19 (1998), pp. 968–994.
[10] D. Cai, E. Chow, Y. Saad, and Y. Xi, *SMASH: Structured matrix approximation by separation and hierarchy.*, Preprint ys-2016-10, Dept. Computer Science and Engineering, University of Minnesota, Minneapolis, MN, (2016).
[11] E. Chow and Y. Saad, *Approximate inverse techniques for block-partitioned matrices*, SIAM J. Sci. Comput., 18 (1997), pp. 1657–1675.
[12] ———, *Experimental study of ILU preconditioners for indefinite matrices*, J. Comput. Appl. Math., 86 (1997), pp. 387–414.
[13] ———, *Approximate inverse preconditioners via sparse-sparse iterations*, SIAM J. Sci. Comput., 19 (1998), pp. 995–1023.
[14] E. C. Cyr, J. N. Shadid, R. S. Tuminaro, R.P. Pawlowski, and L. Chacón, *A new approximate block factorization preconditioner for two-dimensional incompressible (reduced) resistive MHD*, SIAM J. Sci. Comput., 35 (2013), pp. B701–B730.
[15] P.R. Amestoy T.A. Davis and I.S. Duff, *An approximate minimum degree ordering algorithm*, SIAM J. Matrix Anal. Appl., 17 (1996), pp. 886–905.
[16] ———, *Algorithm 837: An approximate minimum degree ordering algorithm*, ACM Trans. Math. Software, 30 (2004), pp. 381–388.
[17] T.A. Davis and Y. Hu, *The University of Florida Sparse Matrix Collection*, ACM Trans. Math. Software, 38 (2011), p. 1.
[18] H. Elman, V. Howle, J. Shadid, R. Shuttleworth, and R. Tuminaro, *Block preconditioners based on approximate commutators*, SIAM J. Sci. Comput., 27 (2006), pp. 1651–1668.

[19] H. C. Elman, D. J. Silvester, and A. J. Wathen, *Finite Elements and Fast Iterative Solvers*, Oxford University Press, Oxford, 2005.
[20] Y. A. Erlangga, C. W. Oosterlee, and C. Vuik, *A novel multigrid based preconditioner for heterogeneous Helmholtz problems*, SIAM J. Sci. Comput., 27 (2006), pp. 1471–1492.
[21] A. George, *Nested dissection of a regular finite element mesh*, SIAM J. Numer. Anal., 10 (1973), pp. 345–363.
[22] M. Grote and T. Huckle, *Parallel preconditioning with sparse approximate inverses*, SIAM J. Sci. Comput., 18 (1997), pp. 838–853.
[23] W. Hackbusch, *A sparse matrix arithmetic based on $\mathcal{H}$-matrices. Part I: Introduction to $\mathcal{H}$-matrices*, Computing, 62 (1999), pp. 89–108.
[24] W. Hackbusch and B. N. Khoromskij, *A sparse $\mathcal{H}$-matrix arithmetic. Part II: Application to multi-dimensional problems*, Computing, 64 (2000), pp. 21–47.
[25] P. Hénon and Y. Saad, *A parallel multistage ILU factorization based on a hierarchical graph decomposition*, SIAM J. Sci. Comput., 28 (2006), pp. 2266–2293.
[26] V. E. Howle and R. C. Kirby, *Block preconditioners for finite element discretization of incompressible flow with thermal convection*, Numerical Linear Algebra with Applications, 19 (2012), pp. 427–440.
[27] V. E. Howle, R. C. Kirby, and G. Dillon, *Block preconditioners for coupled fluids problems*, SIAM Journal of Scientific Computing, 35 (2013), pp. S368–S385.
[28] I. C. F. Ipsen, *A note on preconditioning nonsymmetric matrices*, SIAM Journal on Scientific Computing, 23 (2001), pp. 1050–1051.
[29] V. Kalantzis, Y. Xi, and Y. Saad, *Beyond AMLS: Domain decomposition with rational filtering*, arXiv preprint arXiv:1711.09487, (2017).
[30] G. Karypis and V. Kumar, *A fast and high quality multilevel scheme for partitioning irregular graphs*, SIAM J. Sci. Comput, 20 (1998), pp. 359–392.
[31] E.-J. Lee and J. Zhang, *Hybrid reordering strategies for ILU preconditioning of indefinite sparse matrices*, Journal of Applied Mathematics and Computing, 22 (2006), pp. 307–316.
[32] R. Li and Y. Saad, *Divide and conquer low-rank preconditioners for symmetric matrices*, SIAM J. Sci. Comput., 35 (2013), pp. A2069–A2095.
[33] ———, *GPU-accelerated preconditioned iterative linear solvers*, J. Supercomput., 63 (2013), pp. 443–466.
[34] R. Li, Y. Xi, and Y. Saad, *Schur complement-based domain decomposition preconditioners with low-rank corrections*, Numerical Linear Algebra with Applications, 23 (2016), pp. 706–729.
[35] M. Magolu Monga Made, R. Beauwens, and G. Warzee, *Preconditioning of discrete Helmholtz operators perturbed by a diagonal complex matrix*, Comm. in Numer. Meth. in Engin., 16 (2000), pp. 801–817.
[36] K.-A. Mardal and R. Winther, *Preconditioning discretizations of systems of partial differential equations*, Numerical Linear Algebra with Applications, 18 (2010), pp. 1–40.
[37] M. F. Murphy, G. H. Golub, and A. J. Wathen, *A note on preconditioning for indefinite linear systems*, SIAM Journal on Scientific Computing, 21 (2000), pp. 1969–1972.
[38] D. Osei-Kuffour, R. Li, and Y. Saad, *Matrix reordering using multilevel graph coarsening for ILU preconditioning*, SIAM J. Sci. Comput., 37 (2015), pp. A391–419.
[39] Daniel Osei-Kuffuor and Yousef Saad, *Preconditioning helmholtz linear systems*, Appl. Numer. Math., 60 (2010), pp. 420–431.
[40] Y. Saad, *A flexible inner-outer preconditioned GMRES algorithm*, SIAM J. Sci. Comput., 14 (1993), pp. 461–469.
[41] ———, *ILUM: A multi-elimination ILU preconditioner for general sparse matrices*, SIAM J. Sci. Comput., 17 (1996), pp. 830–847.
[42] ———, *Iterative Methods for Sparse Linear Systems*, SIAM, Philadelphia, 2nd ed. ed., 2003.
[43] Y. Saad and B. Suchomel, *ARMS: An algebraic recursive multilevel solver for general sparse linear systems*, Numer. Linear Algebra Appl., 9 (2002), pp. 359–378.
[44] G.W. Stewart, *Algorithm 506: Hqr3 and exchng: Fortran subroutines for calculating and ordering the eigenvalues of a real upper Hessenberg matrix*, ACM Transactions on Mathematical Software, 2 (1976), pp. 275–280.
[45] M. B. van Gijzen, Y. A. Erlangga, and C. Vuik, *Spectral analysis of the discrete Helmholtz operator preconditioned with a shifted Laplacian*, SIAM J. Sci. Comput., 29 (2007), pp. 1942–1958.
[46] Y. Xi, R. Li, and Y. Saad, *An Algebraic Multilevel Preconditioner with Low-Rank Corrections for Sparse Symmetric Matrices*, SIAM J. Matrix Anal. Appl., 37 (2016), pp. 235–259.
[47] Y. Xi and Y. Saad, *Computing partial spectra with least-squares rational filters*, SIAM Journal on Scientific Computing, 38 (2016), pp. A3020–A3045.

[48] ———, *A rational function preconditioner for indefinite sparse linear systems*, SIAM Journal on Scientific Computing, 39 (2017), pp. A1145–A1167.

[49] Y. Xi and J. Xia, *On the stability of some hierarchical rank structured matrix algorithms*, SIAM Journal on Matrix Anal. Appl., 37 (2016), pp. 1279–1303.

[50] Y. Xi, J. Xia, S. Cauley, and V. Balakrishnan, *Superfast and stable structured solvers for Toeplitz least squares via randomized sampling*, SIAM J. Matrix Anal. Appl., 35 (2014), pp. 44–72.

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

[52] J. Xia, Y. Xi, S. Cauley, and V. Balakrishnan, *Fast sparse selected inversion*, SIAM Journal on Matrix Anal. Appl., 36 (2015), pp. 1283–1314.