A Robust Hierarchical Solver for Ill-conditioned Systems with Applications to Ice Sheet Modeling

Chao Chen[a], Leopold Cambier[a], Erik G. Boman[b], Sivasankaran Rajamanickam[b], Raymond S. Tuminaro[b], Eric Darve[a]

[a]*Stanford University*
[b]*Sandia National Laboratories*

Abstract

A hierarchical solver is proposed for solving sparse ill-conditioned linear systems in parallel. The solver is based on a modification of the LoRaSp method, but employs a deferred-compression technique, which provably reduces the approximation error and significantly improves efficiency. Moreover, the deferred-compression technique introduces minimal overhead and does not affect parallelism. As a result, the new solver achieves linear computational complexity under mild assumptions and excellent parallel scalability. To demonstrate the performance of the new solver, we focus on applying it to solve sparse linear systems arising from ice sheet modeling. The strong anisotropic phenomena associated with the thin structure of ice sheets creates serious challenges for existing solvers. To address the anisotropy, we additionally developed a customized partitioning scheme for the solver, which captures the strong-coupling direction accurately. In general, the partitioning can be computed algebraically with existing software packages, and thus the new solver is generalizable for solving other sparse linear systems. Our results show that ice sheet problems of about 300 million degrees of freedom have been solved in just a few minutes using a thousand processors.

Keywords: Hierarchical matrix, Sparse matrix, Ice sheet modeling, Parallel computing

1. Introduction

This paper considers the problem of solving large sparse linear systems, which is a fundamental building block but also often a computational bottleneck in many science and engineering applications. In particular, we target linear systems that result from the numerical discretization of elliptic partial differentials equations (PDE) including Laplace, Stokes, Helmholtz equations (in low and middle frequency regime), etc., using local schemes such as finite differences or finite elements. One challenge arises when the condition number of the problem is large, and existing solvers become inefficient. Existing solvers fall into three classes. The first class is sparse direct solvers [1], which leverage efficient ordering schemes to perform Gaussian elimination. However, they generally require $\mathcal{O}(N^2)$ computation and $\mathcal{O}(N^{4/3})$ storage for solving a three-dimensional problem of size N. These large costs seriously limit the application of sparse direct solvers to truly large-scale problems. The second class is iterative solvers such as the conjugate gradient method and the multigrid method. These methods may need only $\mathcal{O}(N)$ work and storage per iteration. However, the number of iterations required to achieve convergence can be quite large when solving ill-conditioned linear systems. Preconditioning is essential to improve the conditioning and convergence.

The third class of methods, which is the focus here, is hierarchical solvers. These methods compute an approximate factorization of a discretized elliptic PDE by taking advantage of the fact that the discretization matrix (and its inverse) has certainty hierarchical low-rank structures, including $\mathcal{H}$- [2, 3], $\mathcal{H}^2$- [4, 5] matrices and hierarchically semiseparable (HSS) [6, 7] matrices, among others [8, 9]. By exploiting this data sparsity of the underlying physical problem, hierarchical solvers have been shown to achieve linear or quasi-linear complexity. However, their efficiency deteriorates when highly ill-conditioned problems are encountered because the rank/costs of approximations must increase dramatically in order to maintain the same accuracy in the final solution. Note that hierarchical solvers can be used either as direct solver (high accuracy) or as a preconditioner (low accuracy) for iterative methods. Our focus is on the latter.

In this paper, we introduce the deferred-compression technique to improve the efficiency of hierarchical solvers for solving sparse ill-conditioned linear systems and demonstrate this improvement by implementing it in a particular hierarchical solver named LoRaSp [10]. In hierarchical solvers such as LoRaSp, off-diagonal matrix blocks are compressed with low-rank approx-

imations if they satisfy the strong admissibility condition [4]. In our new solver, these compressible matrix blocks are first scaled by Cholesky factors of the corresponding diagonal blocks before low-rank approximations are applied. This extra scaling step provably reduces errors in the subsequent step of forming the Schur complement. In addition, it increases the likelihood that the Schur complement remains symmetric positive definite with crude low-rank approximation when the original input matrix is SPD. For many practical applications, using the deferred-compression technique to preserve the SPD property of the underlying physical problem is crucial.

Previous deferred-compression work [11, 12, 13] focused on HSS matrices, which is a type of weakly admissible (as opposed to strongly admissible) hierarchical matrices. These approaches are not directly applicable to strongly admissible hierarchical matrices (e.g., $\mathcal{H}^2$ matrices) such as the LoRaSp solver. Furthermore, prior deferred-compression efforts concentrated on solving dense linear systems, where incorporating the deferred-compression step leads to an extra $\mathcal{O}(N^2)$ amount of computation, and no corresponding parallel solver was developed. Compared to the previously published papers, our deferred-compression technique is novel in three ways:

1. we target hierarchical solvers specialized for strongly admissible hierarchical matrices, and develop an associated general error analysis; the previous analysis can be recovered as a special case of our new analysis.
2. we propose a new solver for sparse linear systems for which it is proved that the computational complexity is $\mathcal{O}(N)$ under some mild assumptions. This nearly optimal complexity implies that we can solve large problems with minimum asymptotic computational cost (up to some constants).
3. we show that incorporating the deferred-compression scheme into the LoRaSp solver does not change the data and task dependencies in the parallel LoRaSp solver [14]. Therefore, we can take advantage of the existing parallel algorithm to solve large-scale problems efficiently (on distributed-memory machines).

In order to demonstrate the performance of our new solver, this paper addresses the challenges of solving linear systems from a real-world problem— ice sheet modeling where the solution of discretized linear systems remains the computational bottleneck. Ice sheet modeling is an essential component needed to estimate future sea-level rise due to climate modeling. As noted in [15, 16] from the Intergovernmental Panel on Climate Change (IPCC),

3

modern ice sheet models must continue to introduce advanced features such as adaptive mesh refinement at sub-kilometer resolutions, optimization, data assimilation and uncertainty quantification for the treatment of numerous unknown model inputs. These advances will likely introduce further computational burdens requiring improvements in the linear solver, which must be repeatedly invoked over the course of the simulation. Given that current ice sheet simulations already consume resources on thousands of processing units on modern supercomputers and can involve up to billions of unknown variables, there is a pressing need for efficient linear solvers to reduce simulation costs and prepare for potentially larger more sophisticated simulations in the future.

However, many existing solvers turn out to deliver rather disappointing performance for solving problems from ice sheet modeling. The most prominent challenge comes from the anisotropic nature of ice sheet models, where the thin vertical scale of the domain is tiny relative to the horizontal scale. This extraordinary contrast is also reflected by the dramatically different magnitudes of entries in the discretization matrix, where large values correspond to strong vertical coupling and tiny ones to weak horizontal coupling. This weak coupling gives rise to oscillatory eigenvectors associated with small eigenvalues and a poorly-conditioned linear system. This can be seen from a simplified model $\epsilon u_{xx} + u_{yy}$, where $\epsilon \ll 1$, where the standard five point finite difference discretization on a $n \times n$ regular grid produces a matrix with many small eigenvalues

$$4(n + 1)^2[\epsilon \sin^2(\pi i/(2n + 2)) + \sin^2(\pi j/(2n + 2))]$$

for *all* values of i and small values of j. Further, the Neumann boundary condition imposed on the top surface and some bottom parts of the domain gives rise to problematic linear systems with nearly singular matrix blocks. Physically, the bottom Neumann boundary condition models large ice shelves, which are vast areas of floating ice connected to land-based ice sheets and are common to Antarctica. The resulting Green's function decays much slower along the vertical direction than that for non-sliding ice at a frozen ice interface [17], again contributing to the poor performance of many existing solvers.

The two solvers (preconditioners) commonly used in ice sheet modeling are the incomplete LU factorization (ILU) and the algebraic multigrid method (AMG). Although the customized ILU with a specific ordering scheme

performs reasonably well for the Greenland ice sheet problem, its performance deteriorates significantly for the Antarctic ice sheet problem. The reason is that ice sheets on the Antarctic problem contain a substantial fraction of floating ice shelves, modeled by imposing Neumann boundary conditions, which leads to aforementioned ill-conditioned linear systems. Another possible approach to solve the ice sheet linear systems is some form of algebraic multigrid (AMG). However, standard AMG methods (e.g., the smoothed aggregation AMG solver [18]) do not generally converge on realistic ice sheet simulations. While some specialized AMG techniques have been successfully developed (e.g., a customized matrix-dependent AMG solver [17]) using tailored semi-coarsening schemes, these approaches required significant non-trivial multigrid adaptions to address ice sheet simulations. These types of adaptations are not generally provided with available AMG packages.

To solve the particular linear systems from ice sheet modeling efficiently, our new solver introduces one customization to efficiently address the ice sheet linear systems. Specifically, the typical meshes employed for ice sheet models are generated by first creating a two-dimensional unstructured horizontal mesh and then extruding this mesh into the vertical dimension to create a three-dimensional grid. This mesh structure is leveraged when building clusters for the hierarchical solver. In particular, the (two-dimensional unstructured) non-extruded mesh is first partitioned with a general graph partitioner and then the horizontal partition results are extended along the third/extruded direction such that mesh vertices lying on the same vertical line belong to the same cluster. Since extruded meshes appear frequently in geophysical problems such as atmospheric and oceanic circulation, oil and gas modeling, etc., our new solver along with the "extruded partitioning" algorithm can be generally applied to other engineering simulations involving thin structures. Compared with the ILU and the AMG methods used for ice sheet modeling, our new solver is robust in the sense that the iteration number stays nearly constant if it is used as a preconditioner for solving linear systems associated with ice sheet modeling, and our new solver is general-purpose in that it can be applied as a "black-box" method with a general partitioning scheme available in several existing software packages, such as METIS/ParMETIS [19], Scotch [20] and Zoltan [21], though a special partitioner can also be easily incorporated. Moreover, it is challenging to parallelize the ILU and the AMG methods on modern many-core architectures such as the GPU. Our new solver, similar to other hierarchical solvers, is mainly based on dense linear algebra subroutines and thus can potentially

be accelerated using many-core processors.

To summarize, the paper presents a parallel hierarchical solver for sparse ill-conditioned linear systems using the deferred-compression technique, and in particular, our work makes the following three major contributions:

1. Error analysis of the deferred-compression scheme for hierarchical solvers based on strongly admissible hierarchical matrices (e.g., $\mathcal{H}^2$-matrices).
2. A parallel/distributed-memory hierarchical solver for sparse ill-conditioned linear systems.
3. Application and analysis of the preconditioner for an ice sheet modeling problem, including numerical comparisons with ILU.[1]

The remainder of this paper is organized as follows. Section 2 introduces the deferred-compression technique and provides an error analysis. Following that is the algorithm of our new solver presented in Section 3. Next Section 4 briefly summarizes the first-order-accurate Stokes approximation model of ice sheets and introduces the "extruded partitioning" algorithm. Finally, in Section 5 numerical results are given demonstrating the performance and scalability of our new solver for ice sheet modeling and also general problems from the SuiteSparse Matrix Collection.[2]

2. Deferred-compression Scheme

This section presents the algorithm for deferred-compression and the corresponding error analysis, targeted at hierarchical solvers that are based on strongly admissible hierarchical matrices. These solvers employ low-rank approximations to compress off-diagonal matrix blocks that satisfy the strong-admissibility condition. A rigorous definition of the strong-admissibility condition can be found in [4]. From a high-level perspective, the strong-admissibility condition states that one block-row in a (appropriately partitioned) strongly admissible hierarchical matrix includes a diagonal block corresponding to "self-interaction," a full-rank off-diagonal block corresponding to "neighbor or near-field interaction," and a (numerically) low-rank off-diagonal block corresponding to "well-separated or far-field interaction." Therefore, a strongly admissible hierarchical matrix A can be partitioned as

---

[1]A high-performance implementation in the Trilinos IFPACK package.
[2]https://sparse.tamu.edu/

the following 3×3 block matrix

$$A = \begin{pmatrix} A_{ss} & A_{sn} & A_{sw} \\ A_{ns} & A_{nn} & A_{nw} \\ A_{ws} & A_{wn} & A_{ww} \end{pmatrix}$$

where "s" is a set of row/column indexes that we seek to eliminate via a Cholesky factorization. "n" stands for the set of indexes for which A_{ns} and A_{sn} are full rank, and "w" is used to denote the low-rank blocks A_{ws} and A_{sw}. We further assume that A is SPD in this paper, so $A_{ns} = A_{sn}^T$, $A_{ws} = A_{sw}^T$, and $A_{wn} = A_{nw}^T$.

Below we first review the classical Cholesky factorization and introduce some notations; then we analyze the errors in forming (approximate) Schur complements when the s block is eliminated with and without using the deferred-compression scheme. In order to measure error, we use the matrix-norm (a.k.a., 2-norm or operator norm) denoted by $\| \cdot \|$.

Cholesky factorization. To carry out one step of (block) Cholesky factorization on the s block in A, we define the following three matrices

$$\mathcal{S}_s = \begin{pmatrix} G_s^{-1} & & \\ & I & \\ & & I \end{pmatrix} \qquad \mathcal{L}_s = \begin{pmatrix} I & & \\ -A_{ns}G_s^{-T} & I & \\ -A_{ws}G_s^{-T} & & I \end{pmatrix} \qquad \mathcal{C}_s = \mathcal{L}_s \mathcal{S}_s$$

where $A_{ss} = G_s G_s^T$ is the Cholesky factorization of A_{ss}. The (exact) Schur complement S_A is found in the lower 2×2 block matrix of $\mathcal{C}_s A \mathcal{C}_s^T$ as

$$S_A = \begin{pmatrix} A_{nn} - A_{ns}A_{ss}^{-1}A_{sn} & A_{nw} - A_{ns}A_{ss}^{-1}A_{sw} \\ A_{wn} - A_{ws}A_{ss}^{-1}A_{sn} & A_{ww} - A_{ws}A_{ss}^{-1}A_{sw} \end{pmatrix} \tag{1}$$

To actually compute the Cholesky factorization of the whole matrix A, the Schur complement S_A needs to be further factorized, which is skipped here since this is not relevant for our current discussion.

Without *deferred-compression scheme.* Suppose the low-rank matrix block A_{sw} can be decomposed as

$$A_{sw} = UV^T = \begin{pmatrix} U_1 & U_2 \end{pmatrix} \begin{pmatrix} V_1^T \\ V_2^T \end{pmatrix} = U_1 V_1^T + U_2 V_2^T, \tag{2}$$

where U is an orthogonal matrix and $\|V_2^T\|_2 = \epsilon$, a small prescribed tolerance. This kind of decomposition can be computed using, e.g., a rank-revealing QR

7

factorization (RRQR). Dropping the $U_2 V_2^T$ term in A leads to the compressed matrix $\tilde{A} = compress(A)$ as follows

$$A \approx \tilde{A} = \begin{pmatrix} A_{ss} & A_{sn} & U_1 V_1^T \\ A_{ns} & A_{nn} & A_{nw} \\ V_1 U_1^T & A_{wn} & A_{ww} \end{pmatrix},$$

where the low-rank approximation can be exploited to compute an approximate factorization of A at a lower cost. Apply one step of the Cholesky factorization on the s block in $\tilde{A}$ with

$$\tilde{\mathcal{C}}_s = \tilde{\mathcal{L}}_s \mathcal{S}_s$$

where

$$\tilde{\mathcal{L}}_s = \begin{pmatrix} I & & \\ -A_{ns} G_s^{-T} & I & \\ -V_1 U_1^T G_s^{-T} & & I \end{pmatrix}.$$

As a result, $\tilde{\mathcal{C}}_s \tilde{A} \tilde{\mathcal{C}}_s^T$ contains $S_{\tilde{A}}$, an approximation of S_A with error $\tilde{E}$:

$$S_{\tilde{A}} = \begin{pmatrix} A_{nn} - A_{ns} A_{ss}^{-1} A_{sn} & A_{nw} - A_{ns} A_{ss}^{-1} (U_1 V_1^T) \\ A_{wn} - (V_1 U_1^T) A_{ss}^{-1} A_{sn} & A_{ww} - (V_1 U_1^T) A_{ss}^{-1} (U_1 V_1^T) \end{pmatrix}, \quad (3)$$

$$\tilde{E} = S_{\tilde{A}} - S_A = \begin{pmatrix} 0 & A_{ns} A_{ss}^{-1} (U_2 V_2^T) \\ (V_2 U_2^T) A_{ss}^{-1} A_{sn} & A_{ws} A_{ss}^{-1} A_{sw} - (V_1 U_1^T) A_{ss}^{-1} (U_1 V_1^T) \end{pmatrix}. \quad (4)$$

Proposition 1. *Assume Eq.(2) holds, the error $\tilde{E}$ between the two Schur complements, namely $S_{\tilde{A}}$ in $\tilde{\mathcal{C}}_s \tilde{A} \tilde{\mathcal{C}}_s^T$ and S_A in $\mathcal{C}_s A \mathcal{C}_s^T$ takes the form in Eq.(4). Moreover, the following error estimates hold*

1. $\|\tilde{E}_{ww}\| \leq 2\epsilon \|A_{sw}\| / \sigma_{\min}(A_{ss}) + O(\epsilon^2)$,

2. $\|\tilde{E}_{nw}\| = \|\tilde{E}_{wn}\| \leq \epsilon \|A_{ns}\| / \sigma_{\min}(A_{ss})$,

3. *and* $\|\tilde{E}\| \leq \|\tilde{E}_{ww}\| + \|\tilde{E}_{nw}\| \leq \epsilon (2\|A_{sw}\| + \|A_{ns}\|) / \sigma_{\min}(A_{ss}) + O(\epsilon^2)$,

where $\tilde{E}_{nw}, \tilde{E}_{ww}$ and $\tilde{E}_{ww}$ stand for the (1,2) block, (2,1) block and (2,2) block in $\tilde{E}$.

Proof. The first part of the proposition is already shown above, so we derive the three error bounds as follows.

1.
$$
\begin{aligned}
\|\tilde{E}_{ww}\| =& \|A_{ws}A_{ss}^{-1}A_{sw} - (V_1 U_1^T)A_{ss}^{-1}(U_1 V_1^T)\| \\
=& \|(V_1 U_1^T + V_2 U_2^T)A_{ss}^{-1}(U_1 V_1^T + U_2 V_2^T) - (V_1 U_1^T)A_{ss}^{-1}(U_1 V_1^T)\| \\
=& \|V_1 U_1^T (A_{ss})^{-1}U_2 V_2^T + V_2 U_2^T (A_{ss})^{-1}(U_1 V_1^T + U_2 V_2^T)\| \\
\leq& \|V_1 U_1^T (A_{ss})^{-1}U_2 V_2^T\| + \|V_2 U_2^T (A_{ss})^{-1}(U_1 V_1^T + U_2 V_2^T)\| \\
\leq& \|V_1 U_1^T\|\|(A_{ss})^{-1}\|\|U_2 V_2^T\| + \|V_2 U_2^T\|\|(A_{ss})^{-1}\|\|(U_1 V_1^T + U_2 V_2^T)\| \\
=& \epsilon(\|A_{sw}\| + \epsilon)/\sigma_{\min}(A_{ss}) + \epsilon\|A_{sw}\|/\sigma_{\min}(A_{ss}) \\
=& 2\epsilon\|A_{sw}\|/\sigma_{\min}(A_{ss}) + \epsilon^2/\sigma_{\min}(A_{ss}) \quad\quad\quad\quad (5)
\end{aligned}
$$

where Eq. (2), $\|U_2 V_2^T\| = \epsilon$ and $\|U_1 V_1^T\| = \|A_{sw} - U_2 V_2^T\| \leq \|A_{sw}\| + \epsilon$ are used, and $\sigma_{\min}$ denotes the smallest singular value of a matrix.

2. $\|\tilde{E}_{nw}\| = \|\tilde{E}_{wn}\| = \|A_{ns}A_{ss}^{-1}(U_2 V_2^T)\| \leq \epsilon\|A_{ns}\|/\sigma_{\min}(A_{ss})$

3.
$$
\begin{aligned}
\|\tilde{E}\| =& \left\|\begin{pmatrix} 0 & \tilde{E}_{nw} \\ \tilde{E}_{wn} & \tilde{E}_{ww} \end{pmatrix}\right\| \\
=& \left\|\begin{pmatrix} 0 & \tilde{E}_{nw} \\ \tilde{E}_{wn} & 0 \end{pmatrix} + \begin{pmatrix} 0 & 0 \\ 0 & \tilde{E}_{ww} \end{pmatrix}\right\| \\
\leq& \left\|\begin{pmatrix} 0 & \tilde{E}_{nw} \\ \tilde{E}_{wn} & 0 \end{pmatrix}\right\| + \left\|\begin{pmatrix} 0 & 0 \\ 0 & \tilde{E}_{ww} \end{pmatrix}\right\| \\
=& \|\tilde{E}_{nw}\| + \|\tilde{E}_{ww}\| \\
\leq& \epsilon(2\|A_{sw}\| + \|A_{ns}\|)/\sigma_{\min}(A_{ss}) + \epsilon^2/\sigma_{\min}(A_{ss})
\end{aligned}
$$

where we used the equality $\left\|\begin{pmatrix} 0 & C \\ C^T & 0 \end{pmatrix}\right\| = \|C\|$ for any matrix C. $\qquad\square$

For ill-conditioned problems such as linear systems from ice sheet modeling the diagonal matrix block A_{ss} can be nearly singular, and so $\sigma_{\min}(A_{ss})$ is very small. As a result, the error $\|\tilde{E}\|$ can be large. Worse still, due to this

9

large error the approximate Schur complement $S_{\tilde{A}}$ may become indefinite and the Cholesky factorization of diagonal blocks can break down. This leads to a poor approximation of the exact Schur Complement S_A, an SPD matrix.

The above error analysis extends to all hierarchical solvers based on strongly admissible hierarchical matrices (with potentially minor modifications) and shows that the low-rank truncation error ϵ needs to decrease at least as fast as $\sigma_{\min}(A_{ss})$ to maintain the same error tolerance on $S_{\tilde{A}}$.

With *deferred-compression scheme.* Before compressing the off-diagonal matrix block A_{sw} directly, we first scale A_{sw} by the inverse of the Cholesky factor of A_{ss}. Specifically, the Cholesky factorization of the diagonal block $A_{ss} = G_s G_s^T$ is used to scale the first block row and column of A as the following

$$\mathcal{S}_s A \mathcal{S}_s^T = \begin{pmatrix} I & G_s^{-1} A_{sn} & G_s^{-1} A_{sw} \\ A_{ns} G_s^{-T} & A_{nn} & A_{nw} \\ A_{ws} G_s^{-T} & A_{wn} & A_{ww} \end{pmatrix},$$

where $\mathcal{S}_s = \begin{pmatrix} G_s^{-1} & & \\ & I & \\ & & I \end{pmatrix}$. Then the block $G_s^{-1} A_{sw}$ is compressed with a low-rank approximation. Similar to Eq. (2), assume a low-rank decomposition of $G^{-1} A_{sw}$ as

$$G_s^{-1} A_{sw} = \hat{U}\hat{V}^T = \begin{pmatrix} \hat{U}_1 & \hat{U}_2 \end{pmatrix} \begin{pmatrix} \hat{V}_1^T \\ \hat{V}_2^T \end{pmatrix} = \hat{U}_1 \hat{V}_1^T + \hat{U}_2 \hat{V}_2^T, \tag{6}$$

where $\hat{U}$ is orthogonal and $\|\hat{V}_2\| = \epsilon$, a small prescribed tolerance. One way to relate Eq. (6) to Eq. (2) is the following. Define $\bar{U} = G_s \hat{U}$, then Eq. (6) is equivalent to $A_{sw} = \bar{U}\hat{V}^T$, where $\bar{U}$ is orthogonal in terms of the inner product defined by the SPD matrix A_{ss}^{-1}.

Replacing $G_s^{-1} A_{sw}$ by $\hat{U}_1 \hat{V}_1^T$ in $\mathcal{S}_s A \mathcal{S}_s^T$ leads to the compressed matrix $\mathcal{S}_s \hat{A} \mathcal{S}_s^T = compress(\mathcal{S}_s A \mathcal{S}_s^T)$ as follows

$$\mathcal{S}_s A \mathcal{S}_s^T \approx \mathcal{S}_s \hat{A} \mathcal{S}_s^T = \begin{pmatrix} I & G_s^{-1} A_{sn} & \hat{U}_1 \hat{V}_1^T \\ A_{ns} G_s^{-T} & A_{nn} & A_{nw} \\ \hat{V}_1 \hat{U}_1^T & A_{wn} & A_{ww} \end{pmatrix},$$

where

$$\hat{A} = \begin{pmatrix} A_{ss} & A_{sn} & G_s \hat{U}_1 \hat{V}_1^T \\ A_{ns} & A_{nn} & A_{nw} \\ \hat{V}_1 \hat{U}_1^T G_s^T & A_{wn} & A_{ww} \end{pmatrix}.$$

10

Carrying out one step of Cholesky factorization on the s block in $\mathcal{S}_s\hat{A}\mathcal{S}_s^T$ with

$$\hat{\mathcal{L}}_s = \begin{pmatrix} I & & \\ -A_{ns}G_s^{-T} & I & \\ -\hat{V}_1\hat{U}_1^T & & I \end{pmatrix}$$

produces the following Schur complement, another approximation of S_A as follows

$$S_{\mathcal{S}_s\hat{A}\mathcal{S}_s^T} = \begin{pmatrix} A_{nn} - A_{ns}A_{ss}^{-1}A_{sn} & A_{nw} - A_{ns}G_s^{-T}(\hat{U}_1\hat{V}_1^T) \\ A_{wn} - (\hat{V}_1\hat{U}_1^T)G_s^{-1}A_{sn} & A_{ww} - \hat{V}_1\hat{V}_1^T \end{pmatrix}. \qquad (7)$$

Proposition 2. *Assume Eq.(6) holds, the error $\hat{E}$ between the two Schur complements, namely $S_{\mathcal{S}_s\hat{A}\mathcal{S}_s^T}$ in $\hat{\mathcal{L}}_s\mathcal{S}_s\hat{A}\mathcal{S}_s^T\hat{\mathcal{L}}_s^T$ and S_A in $\mathcal{C}_sA\mathcal{C}_s^T = \mathcal{L}_s\mathcal{S}_s A\mathcal{S}_s^T\mathcal{L}_s^T$ is the following*

$$\hat{E} = S_{\mathcal{S}_s\hat{A}\mathcal{S}_s^T} - S_A = \begin{pmatrix} 0 & A_{ns}G_s^{-T}\hat{U}_2\hat{V}_2^T \\ \hat{V}_2\hat{U}_2^TG_s^{-1}A_{sn} & \hat{V}_2\hat{V}_2^T \end{pmatrix}. \qquad (8)$$

Moreover, the following error estimates hold

1. $\|\hat{E}_{ww}\| = \epsilon^2$,

2. $\|\hat{E}_{nw}\| = \|\hat{E}_{wn}\| \le \epsilon\|A_{ns}\|/\sigma_{\min}(A_{ss})^{1/2}$,

3. *and* $\|\hat{E}\| \le \|\hat{E}_{ww}\| + \|\hat{E}_{nw}\| \le \epsilon\|A_{ns}\|/\sigma_{\min}(A_{ss})^{1/2} + \epsilon^2$,

where $\hat{E}_{nw}, \hat{E}_{ww}$ and $\hat{E}_{ww}$ stand for the (1,2) block, (2,1) block and (2,2) block in $\hat{E}$.

Proof. We first show Eq. (8) as follows:

$$\begin{aligned}
\hat{E}_{nw} &= A_{ns}(A_{ss}^{-1}A_{sw} - G_s^{-T}(\hat{U}_1\hat{V}_1^T)) \\
&= A_{ns}G_s^{-T}(G_s^{-1}A_{sw} - (\hat{U}_1\hat{V}_1^T)) \\
&= A_{ns}G_s^{-T}\hat{U}_2\hat{V}_2^T
\end{aligned}$$

$$\begin{aligned}
\hat{E}_{ww} &= A_{ws}A_{ss}^{-1}A_{sw} - \hat{V}_1\hat{V}_1^T \\
&= (A_{ws}G_s^{-T})(G_s^{-1}A_{sw}) - \hat{V}_1\hat{V}_1^T \\
&= (\hat{V}_1\hat{U}_1^T + \hat{V}_2\hat{U}_2^T)(\hat{U}_1\hat{V}_1^T + \hat{U}_2\hat{V}_2^T) - \hat{V}_1\hat{V}_1^T \\
&= \hat{V}_2\hat{V}_2^T
\end{aligned}$$

11

where Eq. (6) and the orthogonality of $\hat{U}$ ($\hat{U}_1^T \hat{U}_1 = I$ and $\hat{U}_1^T \hat{U}_2 = 0$) are used. Next, we can prove the following three error bounds easily.

$$\|\hat{E}_{ww}\| = \|\hat{V}_2 \hat{V}_2^T\| = \epsilon^2$$

$$\|\hat{E}_{ww}\| = \|A_{ns} G_s^{-T} \hat{U}_2 \hat{V}_2^T\| \leq \epsilon \|A_{ns}\| / \sigma_{\min}(A_{ss})^{1/2}$$

$$\|\hat{E}\| \leq \|\hat{E}_{ww}\| + \|\hat{E}_{nw}\| \leq \epsilon \|A_{ns}\| / \sigma_{\min}(A_{ss})^{1/2} + \epsilon^2$$

which finishes the proof. □

As the above proposition shows, the approximate Schur complement computed with the deferred-compression scheme is much more accurate than that without the scheme, especially when the problem is highly ill-conditioned. In other words, if the error tolerance is fixed, our new solver can deploy a (much) larger truncation error ϵ reducing the setup/factorization cost of a hierarchical solver significantly. For example, in our numerical experiments we will show that our new hierarchical solver ($\epsilon = 10^{-2}$) performs better than the original LoRaSp solver ($\epsilon = 10^{-4}$). In particular, $\hat{E}_{ww}$ is an order of magnitude smaller than $\tilde{E}_{ww}$ and does not depend on $\sigma_{\min}(A_{ss})$. Furthermore, $\hat{E}_{ww}$ is now symmetric positive semi-definite, which implies the following.

Corollary 1. *The $S_{\mathcal{S}_s \hat{A} \mathcal{S}_s^T}^{(2,2)}$ block, i.e., ww/(2,2) block in $S_{\mathcal{S}_s \hat{A} \mathcal{S}_s^T}$ is SPD.*

Proof. The following equality holds according to Eq. (8).

$$S_{\mathcal{S}_s \hat{A} \mathcal{S}_s^T}^{(2,2)} = S_A^{(2,2)} + \hat{E}_{ww}$$

Since the original matrix A is SPD, the exact Schur complement S_A and the block $S_A^{(2,2)}$ are both SPD. It is also obvious that $\hat{E}_{ww} = \hat{V}_2 \hat{V}_2^T$ is a symmetric positive semi-definite matrix. Therefore, $S_{\mathcal{S}_s \hat{A} \mathcal{S}_s^T}^{(2,2)}$ is SPD. □

In general, the matrix $S_{\mathcal{S}_s \hat{A} \mathcal{S}_s^T}$ itself is not necessarily an SPD matrix for any ϵ. However, we observe that the matrix remains SPD for much larger ϵ (lower cost) with the deferred-compression scheme than that in the original algorithm.

Overall, the differences between computing an approximate Schur complement of S_A with and without the deferred-compression scheme are summarized in the following table.

12

Table 1: Differences between computing an approximate Schur complement of S_A with and without the deferred-compression (DC) scheme. * means corresponding blocks in the computed (approximate) Schur complements.

	Without DC	With DC
Matrix	A	$\begin{pmatrix} G_s^{-1} & & \\ & I & \\ & & I \end{pmatrix} A \begin{pmatrix} G_s^{-T} & & \\ & I & \\ & & I \end{pmatrix}$
Low rank	$A_{sw} = UV$	$G^{-1}A_{sw} = \hat{U}\hat{V}^T$
Approxi-mation	$\begin{pmatrix} A_{ss} & A_{sn} & U_1 V_1^T \\ A_{sn}^T & A_{nn} & A_{nw} \\ V_1 U_1^T & A_{nw}^T & A_{ww} \end{pmatrix}$	$\begin{pmatrix} I & G^{-1}A_{sn} & \hat{U}_1\hat{V}_1^T \\ (G^{-1}A_{ns})^T & A_{nn} & A_{nw} \\ \hat{V}_1\hat{U}_1^T & A_{nw}^T & A_{ww} \end{pmatrix}$
Schur complement	Eq. (3)	Eq. (7)
ww block*	may be indefinite	always SPD
Error for ww blocks*	$2\epsilon\|A_{sw}\|/\sigma_{\min}(A_{ss}) + O(\epsilon^2)$	ϵ^2
Error for nw blocks*	$\epsilon\|A_{ns}\|/\sigma_{\min}(A_{ss})$	$\epsilon\|A_{ns}\|/\sigma_{\min}(A_{ss})^{1/2}$

3. Improved LoRaSp Solver

In this section, we complete the algorithm description of our new hierarchical solver obtained by implementing the deferred-compression technique in the original LoRaSp solver. Our goal is to solve an (ill-conditioned) SPD linear system

$$Ax = b \qquad (9)$$

and our solver is based on a clustering of the unknown variables in Eq. (9).

Matrix Partitioning. Define $G_A = (V, E)$ as the (undirected) graph corresponding to the symmetric matrix A: vertices in G_A correspond to row/column indexes in A, and an edge $E_{p,q} = (p, q)$ exists between vertices p and q if $A(p, q) \neq 0$. A clustering of unknown variables in Eq. (9) is equivalent to a partitioning of the graph G_A. Graph partitioning is a well-studied problem and can be computed algebraically using techniques such as spectral partitioning and multilevel methods in existing high-performance packages, such as METIS/ParMETIS [19], Scotch [20] and Zoltan [21].

Our hierarchical solver computes an approximate factorization of A by compressing fill-in blocks generated during Gaussian elimination. The key observation is that the fill-in blocks have low-rank structures, i.e., their singular values decay fast. Intuitively, the inverse of a diagonal block in the discretization matrix corresponds to the discrete Green's function of a local elliptic PDE, which have numerically off-diagonal matrix blocks. The same low-rank property also carries over to the Schur complement [8, 9, 10, 6, 22].

Below, we first illustrate applying the deferred-compression technique and the "low-rank elimination" step ("scaled low-rank elimination" in the following) to one cluster of unknown variables in Section 3.1. Then we present the whole algorithm in Section 3.2 and complexity analysis in Section 3.3.

3.1. Scaled Low-rank Elimination

Let $\Pi = \cup_{i=0}^{m-1} \pi_i$ denote a clustering of all unknown variables in Eq. (9), and without loss of generality, assume that matrix A is partitioned and ordered accordingly, e.g., the first block row/column corresponds to π_0. Two clusters π_p and π_q are defined as "neighbors" if the matrix block $A(\pi_p, \pi_q) \neq 0$. In other words, the neighbors of a cluster is the set of adjacent clusters in G_A.

To use the "scaled low-rank elimination" step, we partition matrix $A_0 = A$ in the familiar way

$$A_0 = \begin{pmatrix} A_{ss} & A_{sn} & A_{sw} \\ A_{ns} & A_{nn} & A_{nw} \\ A_{ws} & A_{wn} & A_{ww} \end{pmatrix}$$

where the "s" block corresponds to π_0, "n" block corresponds to neighbors of π_0 and "w" block corresponds to the rest. Based on our definition of neighbors above, $A_{sw} = A_{ws} = 0$. In this case and generally if $A_{sw} = A_{ws} = 0$, the "scaled low-rank elimination" step is reduced to normal block Cholesky factorization.

As in Section 2, denote $\mathcal{C}_s$ as the matrix corresponding to one step of block Cholesky factorization and denote A_1 as the Schur complement, i.e.,

$$\mathcal{C}_s A_0 \mathcal{C}_s^T = \begin{pmatrix} I & 0 \\ 0 & A_1 \end{pmatrix}.$$

Again, we can partition A_1 into the following 3×3 block matrix

$$A_1 = \begin{pmatrix} A_{ss}^{(1)} & A_{sn}^{(1)} & A_{sw}^{(1)} \\ A_{ns}^{(1)} & A_{nn}^{(1)} & A_{nw}^{(1)} \\ A_{ws}^{(1)} & A_{nw}^{(1)} & A_{ww}^{(1)} \end{pmatrix},$$

where the "s" block corresponds to π_1, the "n" block corresponds to neighbors of π_1 and the "w" block includes all remaining vertices. Assume $A_{sw}^{(1)} \neq 0, A_{ws}^{(1)} \neq 0$, which contains fill-in generated from previous elimination of π_0. To simplify notations, we will drop the superscription of matrix blocks in A_1.

The "scaled low-rank elimination" step involves three operators: scaling operator $\mathcal{S}$, sparsification operator $\mathcal{E}$ and Gaussian elimination operator $\mathcal{G}$. The scaling operator $\mathcal{S}_s$ is defined as follows

$$\mathcal{S}_s = \begin{pmatrix} G_s^{-1} & & \\ & I & \\ & & I \end{pmatrix}, \tag{10}$$

where $A_{ss} = G_s G_s^T$ is the Cholesky factorization.

After the scaling operator is applied, the off-diagonal block $G_s^{-1} A_{sw}$ in $\mathcal{S}_s A_1 \mathcal{S}_s^T$ is compressed with low-rank approximation, as in Eq. (6). This compression step $\mathcal{S}_s A_1 \mathcal{S}_s^T \approx compress(\mathcal{S}_s A_1 \mathcal{S}_s^T)$ is exactly the same as in the

deferred-compression scheme. Instead of eliminating the "s" block directly, the next step applies the sparsification operator

$$\mathcal{E}_s = \begin{pmatrix} \hat{U}^T & & \\ & I & \\ & & I \end{pmatrix} \tag{11}$$

and introduces a zero block as below

$$\mathcal{E}_s \, compress(\mathcal{S}_s A_1 \mathcal{S}_s^T) \, \mathcal{E}_s^T = \mathcal{E}_s \begin{pmatrix} I & G_s^{-1} A_{sn} & \hat{U}_1 \hat{V}_1^T \\ A_{ns} G_s^{-T} & A_{nn} & A_{nw} \\ \hat{V}_1 \hat{U}_1^T & A_{wn} & A_{ww} \end{pmatrix} \mathcal{E}_s^T$$

$$= \begin{pmatrix} I & & \hat{U}_1^T G_s^{-1} A_{sn} & \hat{V}_1^T \\ & I & \hat{U}_2^T G_s^{-1} A_{sn} & 0 \\ A_{ns} G_s^{-T} \hat{U}_1 & A_{ns} G_s^{-T} \hat{U}_2 & A_{nn} & A_{nw} \\ \hat{V}_1 & 0 & A_{wn} & A_{ww} \end{pmatrix}.$$

Notice $\hat{U}^T \hat{U}_1 = \begin{pmatrix} I \\ 0 \end{pmatrix}$ where the identity has the same size as the number of columns in $\hat{U}_1$, i.e., rank of the low-rank approximation in Eq. (6).

After the sparsification step, a cluster of unknown variables π_s can be split into "coarse" unknown variables π_s^c and "fine" unknown variables π_s^f, where π_s^f involves no fill-in. Then π_s^f is eliminated, which does not propagate any existing fill-in (no level-2 fill-in introduced).

The Gaussian elimination operator

$$\mathcal{G}_s = \begin{pmatrix} I & & & \\ & I & & \\ & -A_{ns} G_s^{-T} \hat{U}_2 & I & \\ & & & I \end{pmatrix} \tag{12}$$

eliminates the "fine" unknown variables π_s^f as follows

$$\mathcal{G}_s \mathcal{E}_s \, compress(\mathcal{S}_s A_1 \mathcal{S}_s^T) \, \mathcal{E}_s^T \mathcal{G}_s^T = \begin{pmatrix} I & & \hat{U}_1^T G_s^{-1} A_{sn} & \hat{V}_1^T \\ & I & & \\ A_{ns} G_s^{-T} \hat{U}_1 & & X_{nn} & A_{nw} \\ \hat{V}_1 & & A_{wn} & A_{ww} \end{pmatrix},$$

where $X_{nn} = A_{nn} - A_{ns} G_s^{-T} \hat{U}_2 \hat{U}_2^T G_s^{-1} A_{sn}$.

16

Last, we introduce an auxiliary permutation operator, P_s, to permute rows and columns corresponding to π_s^c to the end. P_s is defined as

$$P_s = \begin{pmatrix} & I & & \\ & & I & \\ & & & I \\ I & & & \end{pmatrix}. \tag{13}$$

Finally, define the "scaled low-rank approximation" operator $\mathcal{W}_s = P_s \mathcal{G}_s \mathcal{E}_s \mathcal{S}_s$ and $\mathcal{W}_s A_1 \mathcal{W}_s^T$ selects and eliminates the fine DOFs in π_s. To summarize, we have derived

$$\mathcal{W}_s A_1 \mathcal{W}_s^T \approx P_s \mathcal{G}_s \mathcal{E}_s \, compress(\mathcal{S}_s A_1 \mathcal{S}_s^T) \, \mathcal{E}_s^T \mathcal{G}_s^T P_s^T = \begin{pmatrix} I & \\ & A_2 \end{pmatrix}.$$

3.2. Entire Algorithm

We have introduced the "scaled low-rank elimination" step for one cluster. The algorithm repeatedly applies this step on all clusters in $\Pi = \cup_{i=0}^{m-1} \pi_i$. This process is equivalent to computing an approximate factorization of the input SPD matrix A, subject to the error of low-rank approximations. After all clusters are processed, one is left with a linear system consisting of the "coarse" unknown variable $\cup_{i=0}^{m-1} \pi_i^c$, and we can apply the same idea on this coarse system. The entire algorithm is shown in Algorithm 1.

Similar to sparse direct solvers, Algorithm 1 outputs an approximate factorization of the original matrix A, which is used to solve the linear system $Ax = b$. Since $\mathcal{S}_i$ and $\mathcal{E}_i$ are block diagonal matrices, $\mathcal{G}_i$ is a triangular matrix and P_i is a permutation matrix, the solve phase follows the standard forward and backward substitution, as shown in Algorithm 2.

3.3. Complexity Analysis

The computational cost and memory requirement of the original LoRaSp method and the corresponding parallel algorithm are analyzed in [10] and [23], respectively. A key assumption of these analyses is that ranks of the low-rank truncations can be bounded from above. We observe this in practice, but it is not possible to guarantee this boundedness without making additional hypotheses on the input matrix. The behavior of ranks in hierarchical matrices has been studied in several existing papers [24, 25, 26]. Here, we make a similar assumption to earlier works on the boundedness of ranks,

17

Algorithm 1 Hierarchical solver: factorization phase

1: **procedure** HIERARCHICAL_FACTOR(A)
2:     **if** the size of A is small enough **then**
3:         Factorize A with the conventional Cholesky factorization
4:         **return**
5:     **end if**
6:     Partition the graph of A and obtain vertex clusters $\Pi = \cup_{i=0}^{m-1} \pi_i$
             ▷ m is chosen to get roughly constant cluster sizes
7:     $A_0 \leftarrow A$
8:     **for** $i \leftarrow 0$ **to** $m - 1$ **do**
9:         $A_{i+1} \leftarrow$ Scaled_LowRank_Elimination(A_i, Π, π_i)
10:    **end for**   ▷ $A_m = \mathcal{W}_{m-1} \dots \mathcal{W}_1 \mathcal{W}_0 A \mathcal{W}_0^T \mathcal{W}_1^T \dots \mathcal{W}_{m-1}^T$
11:    Extract A_c from the block diagonal matrix $A_m \approx \begin{pmatrix} I & \\ & A_c \end{pmatrix}$
             ▷ A_c is the Schur complement for the coarse DOFs
12:    $A_c^{fac} \leftarrow$ HIERARCHICAL_FACTOR(A_c)
             ▷ Recursive call with a smaller matrix
13:    **return** $A^{fac} = \mathcal{W}_0^{-1} \mathcal{W}_1^{-1} \dots \mathcal{W}_{m-1}^{-1} \begin{pmatrix} I & \\ & A_c^{fac} \end{pmatrix} \mathcal{W}_{m-1}^{-T} \dots \mathcal{W}_1^{-T} \mathcal{W}_0^{-T}$
             ▷ A_c^{fac} is not written out explicitly

14: **end procedure**

15: **procedure** SCALED_LOWRANK_ELIMINATION(A_i, Π, π_i)
16:    Extract $\bar{A}$ from $A_i \approx \begin{pmatrix} I & \\ & \bar{A} \end{pmatrix}$
17:    Compute the low-rank elimination operator $\bar{\mathcal{W}}_i = P_i \mathcal{G}_i \mathcal{E}_i \mathcal{S}_i$ based on $\bar{A}$
             ▷ $\mathcal{E}_i, \mathcal{G}_i$ and P_i are defined in Eq. 10, Eq. 12 and Eq. 13
18:    $\mathcal{W}_i \leftarrow \begin{pmatrix} I & \\ & \bar{\mathcal{W}}_i \end{pmatrix}$
             ▷ $\mathcal{W}_i$ has the same size as A_i
19:    **return** $\mathcal{W}_i A_i \mathcal{W}_i^T$
20: **end procedure**
             ▷ Notation: $a \leftarrow b$ means assign the value b to a, whereas $a = b$ means they are equivalent

---

18

Algorithm 2 Hierarchical solver: solve phase

1: **procedure** HIERARCHICAL_SOLVE(A^{fac}, b)
2: $\quad y \leftarrow$ FORWARD_SUBSTITUTION(A^{fac}, b)
3: $\quad x \leftarrow$ BACKWARD_SUBSTITUTION(A^{fac}, y)
4: $\quad$ **return** x
5: **end procedure**

6: **procedure** FORWARD_SUBSTITUTION(A^{fac}, b)
7: $\quad y \leftarrow b$
8: $\quad$ **for** $i \leftarrow 0$ **to** $m-1$ **do**
9: $\quad\quad y \leftarrow \mathcal{W}_i \, y$ $\hfill \triangleright\ y$ is overwritten
10: $\quad$ **end for** $\hfill \triangleright\ y = (y_c, y_f)$ is of the concatenation of y_f and y_c
11: $\quad$ Extract y_f and y_c from y
$\hfill \triangleright\ y_f$ and y_c correspond to the fine DOFs and the coarse DOFs
12: $\quad y_c \leftarrow$ FORWARD_SUBSTITUTION(A_c^{fac}, y_c) $\hfill \triangleright\ y_c$ is overwritten
13: $\quad$ **return** $y = (y_f, y_c)$ $\hfill \triangleright$ output the concatenation of y_f and y_c
14: **end procedure**

15: **procedure** BACKWARD_SUBSTITUTION(A^{fac}, y)
16: $\quad x \leftarrow y$
17: $\quad$ **for** $i \leftarrow m-1$ **to** 0 **do**
18: $\quad\quad x \leftarrow \mathcal{W}_i^T \, x$ $\hfill \triangleright\ x$ is overwritten
19: $\quad$ **end for** $\hfill \triangleright\ x = (x_f, x_c)$ is of the concatenation of x_c and x_f
20: $\quad$ Extract x_f and x_c from x
$\hfill \triangleright\ x_f$ and x_c correspond to the fine DOFs and the coarse DOFs
21: $\quad x_c \leftarrow$ BACKWARD_SUBSTITUTION(A_c^{fac}, x_c) $\hfill \triangleright\ x_c$ is overwritten
22: $\quad$ **return** $x = (x_f, x_c)$ $\hfill \triangleright$ Output the concatenation of x_f and x_c
23: **end procedure**
$\hfill \triangleright$ Notation: $a \leftarrow b$ means assign the value b to a, whereas $a = b$ means they are equivalent

---

which are based on ideas concerning the underlying Green's function that are related to standard multipole estimates [27, 28].

Below we rephrase Theorem 5.4 in [10] and state that it holds as well for the improved LoRaSp solver when similar assumptions are made as in [10]. Complexity analysis of the corresponding parallel algorithm is summarized in Theorem 2, which is again a rephrase of results in [23]. Note that memory and the solve time have the same complexity. Intuitively, the solve phase touches every nonzero once.

Theorem 1. *In the (improved) LoRaSp algorithm, the computational cost of the factorization is $O(Nr^2)$, and the computational cost of the solve (per iteration) and the memory consumption both scale as $O(Nr)$, where N is the problem size and r is the largest cluster size at the first/finest level (level 0), if the following two conditions hold:*

1. *for every cluster of unknown variables, the number of neighbor clusters is bounded by a constant.*
2. *the largest cluster size at the first level (level 0), r, is bounded by a constant; and r_i, the largest cluster size at level i, satisfies the relationship that $r_i \leq \alpha^i r$, where $0 < \alpha < 2^{1/3}$.*

Theorem 2. *Assume the linear system is evenly distributed among all processors, the conditions in Theorem 1 hold and all clusters have r unknown variables. The computational cost of the factorization and the solve (same as the memory consumption) are $O(Nr^2/p)$ and $O(Nr/p)$ on every processor, where p is the number of processors. Further, for every processor, the amount of communication is*

$$O\left(r^2 \left(\frac{N}{rp}\right)^{2/3}\right) = O\left(\left(\frac{Nr^2}{p}\right)^{2/3}\right)$$

for a 3D underlying subdomain, and the number of messages sent by every processor is

$$O\left(\log\left(\frac{N}{rp}\right)\right) + O(\log p).$$

4. Ice Sheet Model

We focus on the first-order Stokes model [29]. This simplified model preserves sufficient accuracy for simulating the flow over most parts of an

ice sheet and is computationally attractive when compared to a full Stokes model. The ice sheet model is discretized with a Galerkin finite element method using either bilinear or trilinear basis functions on tetrahedral or hexahedral elements, respectively. Further details of that underlying discretization can be found in [29, 30]. Below, we provide details on the partial differential equation (PDE) and the corresponding boundary conditions.

4.1. Stokes Formulation and Discretization

The goal of an ice sheet model is to solve for the x and y components of the ice velocity. These two components are approximated by the following elliptic system of PDEs:

$$\begin{cases} -\nabla \cdot (2u\dot{\boldsymbol{\epsilon}}_1) + \rho g \frac{\partial s}{\partial x} = 0 \\ -\nabla \cdot (2u\dot{\boldsymbol{\epsilon}}_2) + \rho g \frac{\partial s}{\partial y} = 0 \end{cases} \tag{14}$$

where μ is the effective viscosity, ρ is ice density, g is the gravitational acceleration, and $s \equiv s(x,y)$ denotes the upper boundary surface. The $\boldsymbol{\epsilon}_i$ are approximations to the effective strain rate tensors:

$$\dot{\boldsymbol{\epsilon}}_1^T = (2\dot{\epsilon}_{xx} + \dot{\epsilon}_{yy}, \dot{\epsilon}_{xy}, \dot{\epsilon}_{xz}) \quad \text{and} \quad \dot{\boldsymbol{\epsilon}}_2^T = (\dot{\epsilon}_{xy}, \dot{\epsilon}_{xx} + 2\dot{\epsilon}_{yy}, \dot{\epsilon}_{yz}) \tag{15}$$

where

$$\dot{\epsilon}_{xx} = \frac{\partial u}{\partial x}, \quad \dot{\epsilon}_{yy} = \frac{\partial v}{\partial y}, \quad \dot{\epsilon}_{xy} = \frac{1}{2}(\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y}), \quad \dot{\epsilon}_{xz} = \frac{1}{2}\frac{\partial u}{\partial z}, \quad \dot{\epsilon}_{yz} = \frac{1}{2}\frac{\partial v}{\partial z}. \tag{16}$$

Nonlinearity arises from the effective viscosity, which is approximated by

$$\mu = \frac{1}{2}A^{-\frac{1}{n}}\dot{\epsilon}_e^{-\frac{2}{n}}, \tag{17}$$

using *Glen's law* [31, 32] to model the ice rheology. Here, $\dot{\epsilon}_e$ is the effective strain rate given by

$$\dot{\epsilon}_e^2 \equiv \dot{\epsilon}_{xx}^2 + \dot{\epsilon}_{yy}^2 + \dot{\epsilon}_{xx}\dot{\epsilon}_{yy} + \dot{\epsilon}_{xy}^2 + \dot{\epsilon}_{xz}^2 + \dot{\epsilon}_{yz}^2 \tag{18}$$

and A is a temperature-dependent factor that can be described through an Arrhenius relation [31]. In this work, we take $n = 3$, as is commonly done. A combination of Newton's method and continuation generates a sequence of linear systems for the new hierarchical solver.

On the top boundary, a homogeneous Neumann condition is prescribed: $\dot{\boldsymbol{\epsilon}}_1 \cdot \boldsymbol{n} = \dot{\boldsymbol{\epsilon}}_2 \cdot \boldsymbol{n} = 0$, where $\boldsymbol{n}$ is the outward facing normal vector to the upper surface. On the bottom boundary, a Robin condition is used:

$$
\begin{aligned}
2\mu\dot{\boldsymbol{\epsilon}}_1 \cdot \boldsymbol{n} + \beta u = 0 \\
2\mu\dot{\boldsymbol{\epsilon}}_2 \cdot \boldsymbol{n} + \beta v = 0
\end{aligned}
\tag{19}
$$

where $\beta \equiv \beta(x, y) \geq 0$ is the basal sliding (or friction) coefficient that in this paper can be viewed as an already known field. Large β (e.g., $\beta = 10^4$ kPa yr m^{-1}) corresponds to a quasi-no-slip condition, while small β implies a weak frictional force, corresponding to a thawed ice-bed interface that allows for some degree of slip tangential to the bedrock. Under floating ice shelves, β is often taken as identically equal to zero, corresponding to a frictionless boundary. Fig. 1 (left) shows the distribution of β in Antarctica. On the lateral boundary, a dynamic Neumann condition (referred to as open-ocean or floating ice) is used:

$$
\begin{aligned}
2\mu\dot{\boldsymbol{\epsilon}}_1 \cdot \boldsymbol{n} - \rho g(s - z)\boldsymbol{n} = \rho_w g \max(z, 0)\boldsymbol{n} \\
2\mu\dot{\boldsymbol{\epsilon}}_2 \cdot \boldsymbol{n} - \rho g(s - z)\boldsymbol{n} = \rho_w g \max(z, 0)\boldsymbol{n}
\end{aligned}
\tag{20}
$$

where ρ_w denotes the density of water and z is the elevation above sea level. This condition is derived under a hydrostatic equilibrium assumption between the ice shelf and the air (or water) that surrounds it [33].

4.2. Extruded Partitioning for Ice Sheets

The improved LoRaSp method is based on an extruded partitioning of a three-dimensional extruded mesh, which logically corresponds to a tensor product of a two-dimensional unstructured mesh in the x, y directions with a one-dimensional mesh in the z direction. Specifically, one layer of the three-dimensional extruded mesh, i.e., an unstructured two-dimensional mesh, is partitioned using a general graph partitioner, such as METIS/ParMETIS [19], Scotch/PT-scotch [20], and Zoltan [21]; the partitioning result is then extruded in the third dimension such that mesh vertices lying on the same extruded line always belong to the same cluster. The motivation of our extruded partitioning scheme is that a mesh point is closer to its vertical neighbors than its horizontal neighbors because vertical coupling is stronger than its horizontal counterpart in ice sheets modeling. Fig. 1 (right) shows the partitioning result of the mesh used for Antarctic ice sheet modeling (the extruded dimension is not shown).
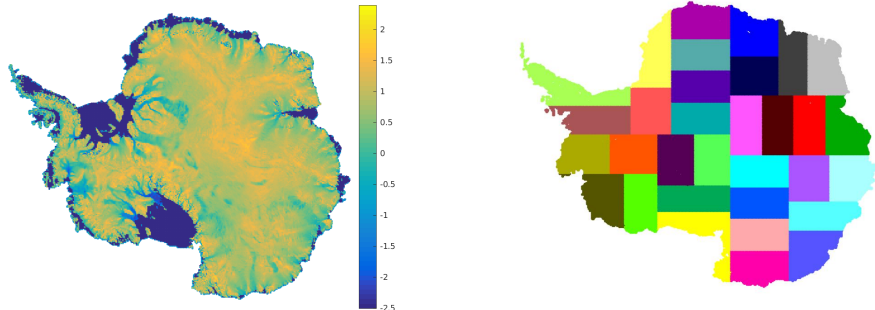
Figure 1: Antarctic ice sheet modeling: (left) distribution of the basal sliding coefficient; (right) partitioning of the two-dimensional mesh, i.e, one layer of the three-dimensional extruded mesh.

Note the extruded partitioning scheme does not assume that the mesh spacing in the extruded dimension is uniform, or that mesh vertexes residing on the same mesh layer have the same z coordinate value. For a number of practical reasons, vertically extruded meshes are commonly employed in ice sheet modeling. In addition to the use in ice sheet modeling, extruded meshes are also heavily used in other geophysical modeling applications (e.g., atmospheric and oceanic, oil/gas, carbon sequestration) and arise frequently in engineering simulations involving thin structures.

5. Numerical Results

This section demonstrates the efficiency and the (parallel) scalability of our hierarchical solver. In particular, we want to answer the following two questions:

1. how does the computation costs, including factorization cost, solve cost per iteration and number of iterations, increase as the problem size increases?
2. how does the running time (factorization cost + solve cost per iteration × iteration count) of our hierarchical solver compare with that of other state-of-the-art methods?

Test problems. We show results for solving linear systems arising from simulating ice sheets on Antarctica. These simulations are carried out on a sequence of increasingly large meshes corresponding to horizontal refinement (fixed number of vertical layers), as is commonly done in practice. The linear systems are solved using the (right) preconditioned GMRES (a restarted GMRES(200) from the Trilinos Belos[3] package) with a stopping tolerance of 10^{-12} and a maximum number of iterations of 1,000.

Parameters in hierarchical solver. In our hierarchical solver, partitions are computed using geometric coordinates of mesh grids by calling the Zoltan [21] library, with cluster sizes around 100, which empirically gives good performance. The (only) other parameter ϵ, i.e., errors of low-rank approximations, is varied to show trade-off between the costs of factorization and solve. When ϵ decreases (more accurate approximations), the factorization cost increases and the number of preconditioned iterations decreases.

Machine. All experiments were run on the NERSC Edison (Cray XC30) supercomputer[4], where every compute node has two 12-core Intel "Ivy Bridge" processors at 2.4 GHz, and nodes are connected with Cray Aries with Dragonfly topology. Our parallel hierarchical solver is implemented using C++ and MPI. The code is compiled with icpc (ICC) 18.0.1 and linked with the Intel MKL library.

5.1. Improved efficiency

This subsection shows the improved efficiency of the hierarchical solver with the vertical partitioning step and the deferred-compression scheme. The focus is on the number of iterations because if we assume the factorization time and the solve time per iteration of the hierarchical solver are both $\mathcal{O}(N)$, then the total running time only depends on the iteration count. The four test problems used in this subsection are the following.

Original LoRaSp method. We first show the poor performance of the original LoRaSp solver, if applied directly to the smallest test matrix corresponding to a resolution of 64km between adjacent mesh points. In the original

---

Table 2: Four test matrices used in Section 5.1

h	N	# of vertical mesh layers
64km	63,126	9
32km	245,646	9
16km	969,642	9
8km	3,848,868	9

h: horizontal mesh resolution/spacing
N: number of unknown variables.

solver, matrix partitioning is computed algebraically with hypergraph parti-
tioning [34] based on the sparsity of the discretization matrix, which ignores
the numerical values in the matrix and would not capture the underlying the
strong/weak coupling. Although more sophisticated partitioning algorithms,
which assign matrix entries to edge weights in the adjacency graph, may lead
to better partitioning results, it is beyond the scope of this paper to explore
such effects.

As Table 3 shows, the original LoRaSp solver did not converge in 100
iterations when $\epsilon \leq 10^{-3}$; when $\epsilon = 10^{-4}$, the solver converged at 69 iterations
with a significant computation time (as compared to results in Table 4).

Table 3: Original LoRaSp solver applied to the linear system corresponding to 64km
resolution.

ϵ	10^{-1}	10^{-2}	10^{-3}	10^{-4}
Factor (s)	12	31	85	134
Solve (s)	—	—	—	44
Iter #	100^a	100^a	100^a	69
Memory (GB)	1	3	6	8

^{a} solver didn't converge in 100 iterations.

Extruded partitioning. Table 4 shows the factorization time, the solve time
(for all iterations), iteration number and the storage cost of the hierarchical
solver using the extruded partitioning scheme. With a pre-processing step of
vertical partitioning, the original LoRaSp solver becomes much more efficient
for solving problems from ice sheet modeling. For example, comparing the
first column in Table 4 with the last column in Table 3, we see that the total

time is about 1 second and 178 seconds for 64km, respectively. Although the performance of LoRaSp has improved significantly with vertical partitioning, the number of iterations doubles as the mesh is refined as shown in Table 4. Suppose the factorization time and the solve time per iteration of the hierarchical solver are both $\mathcal{O}(N)$, the total running time is $\mathcal{O}(N^{3/2})$ as the iteration number grows as $\mathcal{O}(N^{1/2})$.

Table 4: Extruded partitioning. Fixed $\epsilon = 10^{-1}$ in LoRaSp solver.

Resolution	64km	32km	16km	8km
Factor (s)	0.67	2.5	10	41
Solve (s)	0.41	3.7	33	220
Iter #	12	26	52	107
Memory (GB)	0.4	4	7	27

Table 5 shows the number of iterations of different values of ϵ for increasing problem sizes. As shown in the table, the number of iterations decreases as ϵ decreases. When $\epsilon \geq 10^{-3}$, the iteration number roughly doubles as meshes are refined. When $\epsilon = 10^{-4}$, the number of iterations increases relatively slowly. In principle, we could further decrease ϵ and the number of iterations would be further reduced. But the increase of factorization cost with a smaller ϵ may lead to a higher total running time.

Table 5: Extruded partitioning. Number of iterations for different values of ϵ.

ϵ	10^{-1}	10^{-2}	10^{-3}	10^{-4}
64km	12	12	11	11
32km	26	22	21	17
16km	52	44	37	28
8km	107	83	71	35

Deferred compression. Table 6 shows the number of iterations of different values of ϵ when the deferred-compression scheme is used. As the table shows, the number of iterations is reduced significantly. More importantly, the iteration count is almost constant when $\epsilon \leq 10^{-3}$ and increases logarithmically when $\epsilon = 10^{-2}$. Suppose the factorization time and the solve

time per iteration are both $\mathcal{O}(N)$, the total running time would be $\mathcal{O}(N)$ or $\mathcal{O}(N \log(N))$ when $\epsilon \leq 10^{-3}$ or $\epsilon = 10^{-2}$, respectively.

Table 6: Deferred-compression scheme. Number of iterations for different values of ϵ.

ϵ	10^{-1}	10^{-2}	10^{-3}	10^{-4}
64km	14	10	9	5
32km	21	12	8	5
16km	37	14	8	5
8km	54	16	8	6

5.2. Ice sheet problems

In this subsection, we show running time of our hierarchical solver for solving practically large-scale linear systems from ice sheet modeling. Based on previous results, we chose $\epsilon = 10^{-2}$ for our hierarchical solver, which incorporates the deferred-compression scheme and extruded partitioning.

Our reference method is the ILU-preconditioned domain decomposition method used in the Albany package [29] developed at the Sandia National Laboratories for ice sheet modeling. The ILU module is a well-tuned high-performance implementation in Trilinos IFPACK[5]. Since the factorization time of ILU is a tiny fraction in the total runtime, it is not shown explicitly in the following figures and tables.

In the following numerical experiments, we will fix the number of vertical mesh layers at either 6 mesh layers or 11 mesh layers, which are two common ice sheet modeling choices for low- and high-accuracy. Correspondingly, the numbers of unknowns on the same vertical line are 12 and 22 (as there are two unknowns associated with every grid point).

6 vertical mesh layers. Fig. 2 shows the total running time of a weak scaling experiment[6], where a sequence of problems are solved on 1, 4, 15, 64 and 256 processors. As Fig. 2 (left) shows, the running time of ILU blows up as the problem size increases, while that of the hierarchical solver remains almost constant. Fig. 2 (right) shows the decay of residuals, and the convergence of ILU deteriorates significantly as the problem size increases.

---

[5]https://trilinos.org/packages/ifpack/
[6]the problem size increases proportionally to the number of processors used. In other words, the problem size per processor is fixed.
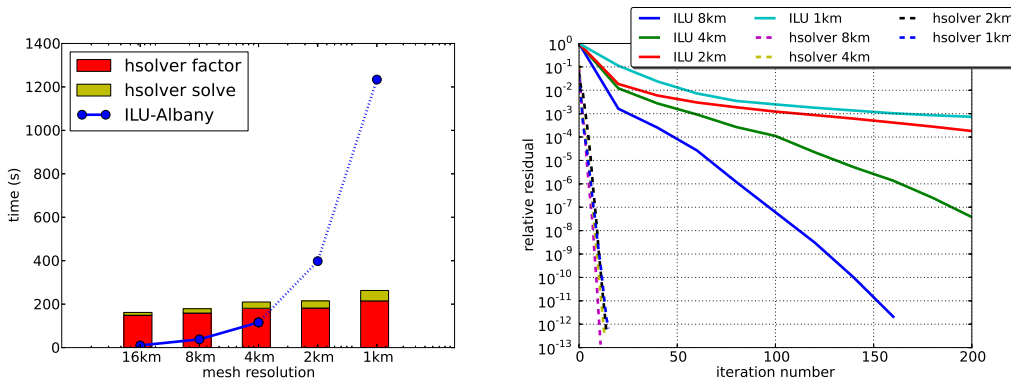
Figure 2: 6 vertical mesh layers. Weak scaling experiment on 1, 4, 16, 64, 256 processors. (Left) comparison of the total runtime (factorization+solve) between ILU (factorization time is negligible and not shown explicitly) and our hierarchical solver (hsolver). Dashed line means extrapolation based on existing data because ILU didn't converge to 10^{-12}. (Right) Decay of residuals in ILU and our hierarchical solver (hsolver).

Table 7: 6 vertical mesh layers: hierarchical solver ($\epsilon = 10^{-2}$) vs. ILU.

h	N	P	ILU iter #	ILU total time	hierarchical solver iter #	hierarchical solver factor	hierarchical solver solve
16km	629K	1	64	10	10	149	13
8km	2.5M	4	170	38	12	159	20
4km	10M	16	498	116	14	181	29
2km	40M	64	1000^a	—	14	182	33
1km	161M	256	1000^b	—	15	215	48

h: horizontal mesh resolution/spacing, N: number of unknown variables, P: number of processors.

[a] ILU didn't converge to 10^{-12}; it took 398 seconds for 1000 iterations (residual $\approx 10^{-10}$).

[b] ILU didn't converge to 10^{-12}; it took 346 seconds for 1000 iterations (residual $\approx 10^{-6}$).

Detailed information about this weak scaling experiment is summarized in Table 7. As the mesh is refined every time, the number of iterations for ILU doubles, whereas it increases by only one or two steps for our hierarchical solver. As a result, we conclude that the computation cost of ILU is $\mathcal{O}(N^{3/2})$
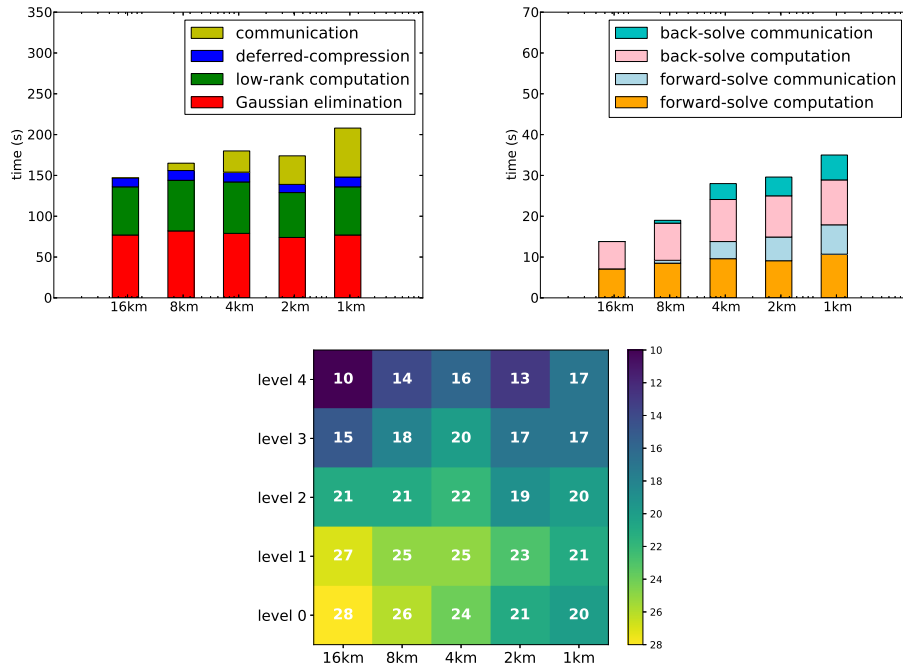
Figure 3: 6 vertical mesh layers. Weak scaling experiment on 1, 4, 16, 64, 256 processors. (First row) breakdown of the factorization time and the solve time (forward substitution+backward substitution for all iterations) in Table 7. (Second row) Average sizes of low-rank compression at different levels ("level 0" is the finest level) in the hierarchical solver ($\epsilon = 10^{-2}$).

as the iteration count increases as $\mathcal{O}(N^{1/2})$ empirically. By contrast, our hierarchical solver achieved $\mathcal{O}(N \log(N))$ computational complexity.

Fig. 3 (first row) shows the breakdown of the factorization time and the solve time (for all iterations) on one processor in the parallel hierarchical solver. In our weak scaling experiment, the deferred-compression time, low-rank compression time, Gaussian elimination time and solve time all stay almost constant as the problem size increases (proportionally to the number of processors used). Moreover, the cost of the deferred-compression scheme is only a small fraction of the total factorization cost.

Fig. 3 (second row) shows the average sizes of low-rank compression at all levels and problem sizes are well-bounded and hence the total running time of the hierarchical solver scales closely to $\mathcal{O}(N)$ as Thm. 1 and Thm. 2 state.

11 vertical mesh layers. A weak scaling study for solving a sequence of increasingly large linear systems on 4, 16, 64, 256 and 1024 processors are shown in Table 8. Again, the number of iterations of ILU increases as $\mathcal{O}(N^{1/2})$ while that of the hierarchical solver increases very slowly. As a result, the computation cost of ILU behaves as $\mathcal{O}(N^{3/2})$, whereas our hierarchical solver scales as $\mathcal{O}(N \log(N))$.

Table 8: 11 vertical mesh layers: hierarchical solver ($\epsilon = 10^{-2}$) vs. ILU.

| h | N | P | ILU | | hierarchical solver | | |
			iter #	total time	iter #	factor	solve
16km	1.1M	4	90	7	18	147	22
8km	4.6M	16	183	21	23	186	38
4km	18.5M	64	468	66	24	213	53
2km	74M	256	1000[a]	—	27	214	65
1km	296M	1024	1000[b]	—	27	243	71

h: horizontal mesh resolution/spacing, N: number of unknown variables, P: number of processors.
[a] ILU didn't converge to 10^{-12}; it took 145 seconds for 1000 iterations (residual $\approx 10^{-9}$).
[b] ILU didn't converge to 10^{-12}; it took 83 seconds for 1000 iterations (residual $\approx 10^{-3}$).

Fig. 4 shows the breakdown of the factorization time and solve time (forward-substitution+backward-substitution) per iteration for different levels. As the figure shows, both the factorization time and the solve time decreases proportionally at coarser levels. The reason for this is that the number of partitions at the coarse level is halved while the size of every partition (twice the compression rank) remains bounded. This type of behavior is commonly observed in the profile of multi-level methods (e.g., the multigrid method and the fast multipole method) and is crucial for achieving parallel scalability.

6. Conclusions and future work

In this paper, we have introduced the deferred-compression technique for developing robust hierarchical solvers based on strongly admissible hierarchical matrices. For these matrices, off-diagonal matrix blocks that satisfy the
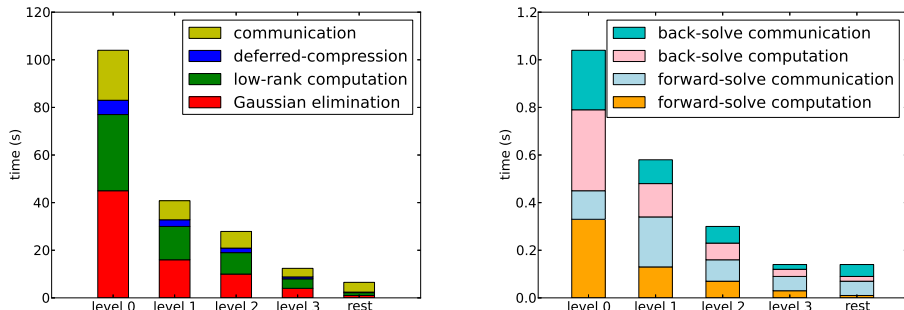
Figure 4: 11 vertical mesh layers. Breakdown of the factorization time and the solve time (forward substitution+backward substitution for one iteration) for the 4km resolution (on 64 processors). Note "level 0" is the finest level in the hierarchical solver ($\epsilon = 10^{-2}$).

strong admissibility condition are numerically low-rank (a.k.a., data-sparse). This low-rank property is leveraged in fast algorithms for computing approximate Cholesky factorizations of an SPD matrix, where (block) Gaussian elimination is applied after low-rank blocks are compressed. In the deferred-compression scheme, by contrast, these matrix blocks are first scaled by the inverse of the Cholesky factor of the corresponding diagonal block before low-rank approximations are applied. This deferred compression provably reduces the error in forming the subsequent Schur complement, especially for ill-conditioned linear systems. Our analysis shows that the ww block in the Schur complement becomes second-order accurate (ϵ^2) with respect to the truncation error ϵ, as opposed to first-order accurate ($\epsilon/\sigma_{\min}(A_{ss})$) in the original algorithm, and more importantly, the block is shown to be SPD.

The effectiveness of the deferred-compression scheme is demonstrated through the newly developed improved LoRaSp solver, which is based on the original LoRaSp method and deploys the new compression technique. The improved LoRaSp solver has linear computational complexity under some mild assumptions, and its parallelization is similar to the original LoRaSp solver. Similar to ILU, the improved LoRaSp solver computes an approximate factorization by compressing fill-in blocks, but its dropping/truncation rule is based on the decay of singular values, which is expected to be more efficient than other level-based or threshold-based rules typically used in ILU. With a general graph partitioner, the improved LoRaSp solver can be used as a "black-box" method to solve general SPD sparse linear systems.

31

The application of ice sheet modeling is studied to benchmark the improved LoRaSp solver against other state-of-the-art methods. The standard smoothed aggregation AMG solver struggles due to difficulties associated with the strong anisotropic phenomena. On the other hand, ILU, a commonly used method in practical ice sheet simulations, has the disadvantage that the number of iterations doubles as the discretization mesh is refined, making it an $\mathcal{O}(N^{3/2})$ method. Compared with existing methods, our improved LoRaSp solver delivers a $\mathcal{O}(N)$ solution for a wide range of meshes. For extruded meshes used in ice sheet modeling, we have developed the extruded partitioning scheme to boost the performance of our solver, and we expect this approach to be effective for other geophysical modeling of thin structures.

Several directions for future research are as follows.

- The deferred-compression technique does not guarantee the subsequent Schur complement to be SPD. The creation of a numerical algorithm that guarantees the SPD property with strong admissibility is currently open.

- The deferred-compression scheme and the improved LoRaSp solver were developed for SPD matrices; their extensions to non-symmetric matrices should be explored. For non-symmetric matrices, the optimal scaling factors for the upper triangular and the lower triangular parts need to be determined.

- If the near-null space (very small singular values) of a physical model is available, it can be taken advantage of by hierarchical solvers to accelerate convergence. Such a scheme [35] respects extra constraints on low-rank approximations and would resolve any vector in the near-null space exactly or very accurately.

7. Acknowledgments

[1] T. A. Davis, S. Rajamanickam, W. M. Sid-Lakhdar, A survey of direct methods for sparse linear systems, Acta Numerica 25 (2016) 383–566.

[2] W. Hackbusch, A sparse matrix arithmetic based on $\mathcal{H}$-matrices. Part I: Introduction to $\mathcal{H}$-matrices, Computing 62 (1999) 89–108.

[3] W. Hackbusch, B. N. Khoromskij, A sparse $\mathcal{H}$-matrix arithmetic., Computing 64 (2000) 21–47.

[4] W. Hackbusch, S. Börm, Data-sparse approximation by adaptive $\mathcal{H}^2$-matrices, Computing 69 (2002) 1–35.

[5] W. Hackbusch, $\mathcal{H}^2$-matrices, in: Hierarchical Matrices: Algorithms and Analysis, Springer, 2015, pp. 203–240.

[6] J. Xia, S. Chandrasekaran, M. Gu, X. S. Li, Fast algorithms for hierarchically semiseparable matrices, Numerical Linear Algebra with Applications 17 (2010) 953–976.

[7] S. Chandrasekaran, M. Gu, T. Pals, A fast ULV decomposition solver for hierarchically semiseparable representations, SIAM Journal on Matrix Analysis and Applications 28 (2006) 603–622.

[8] P. Amestoy, C. Ashcraft, O. Boiteau, A. Buttari, J.-Y. L'Excellent, C. Weisbecker, Improving multifrontal methods by means of block low-rank representations, SIAM Journal on Scientific Computing 37 (2015) A1451–A1474.

[9] A. Aminfar, S. Ambikasaran, E. Darve, A fast block low-rank dense solver with applications to finite-element matrices, Journal of Computational Physics 304 (2016) 170–188.

[10] H. Pouransari, P. Coulier, E. Darve, Fast hierarchical solvers for sparse matrices using extended sparsification and low-rank approximation, SIAM Journal on Scientific Computing 39 (2017) A797–A830.

[11] J. Xia, M. Gu, Robust approximate Cholesky factorization of rank-structured symmetric positive definite matrices, SIAM Journal on Matrix Analysis and Applications 31 (2010) 2899–2920.

[12] J. Xia, Z. Xin, Effective and robust preconditioning of general SPD matrices via structured incomplete factorization, SIAM Journal on Matrix Analysis and Applications 38 (2017) 1298–1322.

[13] X. Xing, E. Chow, Preserving positive definiteness in hierarchically semiseparable matrix approximations, SIAM Journal on Matrix Analysis and Applications 39 (2018) 829–855.

[14] C. Chen, S. Rajamanickam, E. G. Boman, E. Darve, Parallel hierarchical solver for elliptic partial differential equations, Technical Report, Sandia National Laboratories, 2016.

[15] S. Solomon, Climate change 2007-the physical science basis: Working Group I contribution to the fourth assessment report of the IPCC, volume 4, Cambridge University Press, 2007.

[16] T. Stocker, Climate change 2013: the physical science basis: Working Group I contribution to the fifth assessment report of the Intergovernmental Panel on Climate Change, Cambridge University Press, 2014.

[17] R. Tuminaro, M. Perego, I. Tezaur, A. Salinger, S. Price, A matrix dependent/algebraic multigrid approach for extruded meshes with applications to ice sheet modeling, SIAM Journal on Scientific Computing 38 (2016) C504–C532.

[18] P. Vaněk, J. Mandel, M. Brezina, Algebraic multigrid by smoothed aggregation for second and fourth order elliptic problems, Computing 56 (1996) 179–196.

[19] G. Karypis, V. Kumar, A fast and high quality multilevel scheme for partitioning irregular graphs, SIAM Journal on scientific Computing 20 (1998) 359–392.

[20] C. Chevalier, F. Pellegrini, PT-Scotch: A tool for efficient parallel graph ordering, Parallel computing 34 (2008) 318–331.

[21] E. G. Boman, Ü. V. Çatalyürek, C. Chevalier, K. D. Devine, The Zoltan and Isorropia parallel toolkits for combinatorial scientific computing: Partitioning, ordering and coloring, Scientific Programming 20 (2012) 129–150.

[22] K. L. Ho, L. Ying, Hierarchical interpolative factorization for elliptic operators: differential equations, Communications on Pure and Applied Mathematics 69 (2016) 1415–1451.

[23] C. Chen, R. Tuminaro, S. Rajamanickam, E. G. Boman, E. Darve, A hierarchical solver for extruded meshes with applications to ice sheet modeling, in: Center for Computing Research Summer Proceedings 2017, A.D. Baczewski and M.L. Parks, eds., Technical Report SAND2018-2780O, Sandia National Laboratories, pp. 3–18.

[24] M. Bebendorf, W. Hackbusch, Existence of $\langle$-matrix approximants to the inverse fe-matrix of elliptic operators with l^∞-coefficients, Numerische Mathematik 95 (2003) 1–28.

[25] M. Bebendorf, Efficient inversion of the galerkin matrix of general second-order elliptic operators with nonsmooth coefficients, Mathematics of Computation 74 (2005) 1179–1199.

[26] S. Chandrasekaran, P. Dewilde, M. Gu, N. Somasunderam, On the numerical rank of the off-diagonal blocks of schur complements of discretized elliptic pdes, SIAM Journal on Matrix Analysis and Applications 31 (2010) 2261–2290.

[27] L. Greengard, V. Rokhlin, A fast algorithm for particle simulations, Journal of computational physics 73 (1987) 325–348.

[28] L. Greengard, V. Rokhlin, A new version of the fast multipole method for the laplace equation in three dimensions, Acta numerica 6 (1997) 229–269.

[29] I. K. Tezaur, M. Perego, A. G. Salinger, R. S. Tuminaro, S. F. Price, Albany/FELIX: a parallel, scalable and robust, finite element, first-order stokes approximation ice sheet solver built for advanced analysis, Geoscientific Model Development 8 (2015) 1197–1220.

[30] I. K. Tezaur, R. S. Tuminaro, M. Perego, A. G. Salinger, S. F. Price, On the scalability of the Albany/FELIX first-order stokes approximation ice sheet solver for large-scale simulations of the Greenland and Antarctic ice sheets, Procedia Computer Science 51 (2015) 2026–2035.

[31] K. M. Cuffey, W. S. B. Paterson, The physics of glaciers, Academic Press, 2010.

[32] J. Nye, The distribution of stress and velocity in glaciers and ice-sheets, in: Proceedings of the Royal Society of London A: Mathematical, Physical and Engineering Sciences, volume 239, The Royal Society, pp. 113–133.

[33] D. R. MacAyeal, V. Rommelaere, P. Huybrechts, C. L. Hulbe, J. Determann, C. Ritz, An ice-shelf model test based on the Ross Ice Shelf, Antarctica, Annals of Glaciology 23 (1996) 46–51.

[34] K. D. Devine, E. G. Boman, R. T. Heaphy, R. H. Bisseling, U. V. Catalyurek, Parallel hypergraph partitioning for scientific computing, in: Parallel and Distributed Processing Symposium, 2006. IPDPS 2006. 20th International, IEEE, pp. 10–pp.

[35] K. Yang, H. Pouransari, E. Darve, Sparse hierarchical solvers with guaranteed convergence, arXiv preprint arXiv:1611.03189 (2016).