Multiscale Cholesky Preconditioning for Ill-conditioned Problems

Jiong Chen, Florian Schäfer, Jin Huang, Mathieu Desbrun

Multiscale Cholesky Preconditioning for Ill-conditioned Problems

JIONG CHEN, State Key Lab of CAD&CG, Zhejiang University / Inria
FLORIAN SCHÄFER, California Institute of Technology
JIN HUANG, State Key Lab of CAD&CG, Zhejiang University
MATHIEU DESBRUN, Inria / Ecole Polytechnique / Caltech

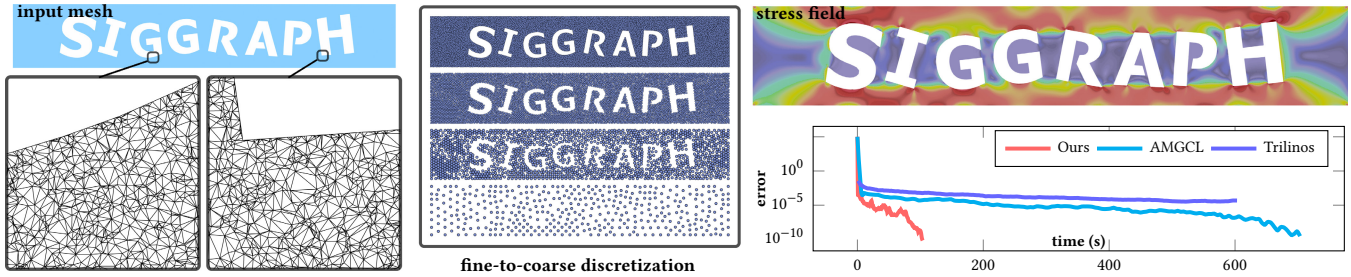


Fig. 1. **Multiscale Cholesky preconditioning**. Given an unstructured Delaunay mesh (left), our method generates a hierarchy of discretizations (middle) to reorder the 816k degrees of freedom and derive a sparsity pattern from which we can efficiently compute a zero fill-in incomplete Cholesky (IC) factorization of, e.g., an elastic stiffness matrix for finite element analysis (right, top). The resulting factorization can be used as a preconditioner for an iterative Preconditioned Conjugate Gradient solver, outperforming (even on this homogeneous case) existing highly-optimized algebraic multigrid preconditioners which typically exhibit slow convergence for ill-conditioned large-scale systems (right, bottom), and without requiring the large amount of memory that direct solvers consume.

Many computer graphics applications boil down to solving sparse systems of linear equations. While the current arsenal of numerical solvers available in various specialized libraries and for different computer architectures often allow efficient and scalable solutions to image processing, modeling and simulation applications, an increasing number of graphics problems face large-scale and ill-conditioned sparse linear systems — a numerical challenge which typically chokes both direct factorizations (due to high memory requirements) and iterative solvers (because of slow convergence). We propose a novel approach to the efficient preconditioning of such problems which often emerge from the discretization over unstructured meshes of partial differential equations with heterogeneous and anisotropic coefficients. Our numerical approach consists in simply performing a fine-to-coarse ordering and a multiscale sparsity pattern of the degrees of freedom, using which we apply an incomplete Cholesky factorization. By further leveraging supernodes for cache coherence, graph coloring to improve parallelism and partial diagonal shifting to remedy negative pivots, we obtain a preconditioner which, combined with a conjugate gradient solver, far exceeds the performance of existing carefully-engineered libraries for graphics problems involving bad mesh elements and/or high contrast of coefficients. We also back the core concepts behind our simple solver with theoretical foundations linking the recent method of operator-adapted wavelets used in numerical homogenization to the traditional Cholesky factorization of a matrix, providing us with a clear bridge between incomplete Cholesky factorization and multiscale analysis that we leverage numerically.

CCS Concepts: • **Mathematics of computing → Solvers**.

Additional Key Words and Phrases: Numerical solvers, preconditioned conjugate gradient, incomplete Cholesky decomposition, wavelets.

ACM Reference Format:
Jiong Chen, Florian Schäfer, Jin Huang, and Mathieu Desbrun. 2021. Multiscale Cholesky Preconditioning for Ill-conditioned Problems. *ACM Trans. Graph.* 40, 4, Article 81 (August 2021), 13 pages. https://doi.org/10.1145/3450626.3459851

1 INTRODUCTION

With the rapid development of visual computing techniques and computational hardware, the graphics community can tackle increasingly complex problems arising from an ever-wider range of applications through large-scale sparse linear algebra. While there is a plethora of scalable numerical algorithms and optimized libraries to offer efficient solutions to a large amount of graphics methods, the situation is far more dire if one needs to solve very large ill-conditioned systems as typically arising from the discretization of high-contrast partial differential equations over unstructured grids: direct factorizations are prohibitively demanding in terms of memory size, while iterative solvers exhibit very poor convergence rates.

In this paper, we propose a novel incomplete Cholesky preconditioner for iteratively solving large-scale ill-conditioned linear systems typically generated by the discretization of high-contrast materials with unstructured meshes. Based on recent numerical homogenization techniques used in graphics and applied mathematics, our method is conceptually simple, and our current implementation already offers orders of magnitude acceleration over optimized, specialized libraries, while keeping the memory size linear in the number of degrees of freedom.

Authors' addresses: J. Chen, J. Huang (corresponding author), State Key Lab of CAD&CG, Zhejiang University, Hangzhou, China; F. Schäfer, Computing + Mathematical Sciences, Caltech, Pasadena, USA; M. Desbrun, Inria Saclay, LIX/DIX, Institut Polytechnique de Paris, Palaiseau, France; on leave from Caltech.

1.1 Related work

Direct and iterative solvers. Linear solvers are often classified into two categories, namely direct and iterative solvers. The former class is typically based on Gaussian elimination, and includes LU factorization and its symmetric variant, Cholesky factorization. Although accurate and stable, these direct solvers do not scale well in terms of both computational time and memory cost, and are thus only used for small matrices or when several solves using a same matrix are called for. Reuse or partial updating of Cholesky factorization to speedup specific applications has also been proposed [Herholz et al. 2017; Herholz and Alexa 2018; Herholz and Sorkine-Hornung 2020]. As for the latter class, Krylov subspace approaches such as the preconditioned conjugate gradient method are widely used. They can be very efficient computationally with small memory requirements, but their convergence properties depend heavily on the efficacy of their preconditioner. Therefore, a lot of effort have been spent on the design of efficient preconditioners to accelerate iterative solves. A popular approach to design preconditioners for geometric problems are so-called *multigrid methods*.

Multigrid methods. A key issue of iterative methods is their lack of efficiency in globally approximating solutions through local updates given by matrix-vector products. Multigrid methods remedy this weakness by alternating between updates on different scales and exchanging information across scales. Geometric multigrid method (GMG) has been widely applied to accelerate solve for its simplicity in hierarchical construction [Zhu et al. 2010]; however, for inhomogeneous problems the geometry-based prolongation and restriction operators cannot properly capture the underlying anisotropy. Numerically, this translates in each level not acting on a restricted range of the operator spectrum, eventually leading to very poor convergence. Algebraic multigrid methods (AMG) can often perform better in practice. For Laplacian matrices on regular meshes in particular, [Krishnan et al. 2013] proposed an AMG-style preconditioner, where coarsening is iteratively done by collapsing weak connections between matrix coefficients. While this approach has very strong preconditioning properties for matrices containing only non-positive off-diagonal entries (a particular instance of M-matrices [Fiedler and Pták 1962]), its performance worsens dramatically otherwise, for instance when the input matrix is the cotangent Laplacian or an elastic stiffness matrix computed on an irregular mesh containing obtuse angles. General AMG preconditioners, such as the smoothed aggregation approach (used, e.g., in [Tamstorf et al. 2015]), are not restricted to M-matrices, but they still struggle when faced with strong anisotropy or high contrast materials: the restriction and prolongation operators are known to lose their efficacy when high contrast variations are present in the material coefficients [Alcouffe et al. 1981; Yavneh 2006; Brandt et al. 2011].

Homogenization. A closely related problem in computational science is *numerical homogenization*, i.e., the computation of *coarse-grained* operators that capture the large scale behavior of the solutions of a given PDE, while preserving its locality as reflected in the sparsity of its matrix representation. For elliptic problems with *smooth* coefficients, one can represent the operator matrix in a wavelet basis of sufficiently high order of vanishing moments. Gines et al. [1998] apply Gaussian elimination to such a representation to obtain a fast solver that can be seen as a direct analog of multigrid methods. The seminal work of Målqvist and Peterseim [2014] proved that the so-called *Local Orthogonal Decomposition* (LOD) achieves near-optimal coarse graining of general second order elliptic operators. Kornhuber et al. [2018] elegantly re-express these results by relating them to the convergence theory of additive Schwarz methods. By casting the design of prolongation operators as a statistical inference problem, Owhadi et al. [2017; 2019] introduce the concept of *gamblets*, a class of operator-adapted wavelets that can be interpreted as a multiscale version of LOD which can be computed in near-linear computational complexity resulting in a fast solver for general elliptic PDE. An interesting relationship between gamblets and Gaussian elimination was pointed out in Schäfer et al. [2021], providing a simpler construction of gamblets via Cholesky factorization with improved asymptotic complexity. Gamblets were also used in computer animation by [Chen et al. 2019] to design material-adapted, refinable basis functions and associated wavelets to offer efficient coarse-graining of elastic objects made of highly heterogeneous materials. Due to its ability to construct space- and eigenspace-localized functional spaces, this recent approach exhibits far improved homogenization properties compared to the long line of work on model reduction in animation [Nesme et al. 2009; Kharevych et al. 2009; Torres et al. 2014; Chen et al. 2017, 2018], allowing for runtime simulations on very coarse resolution grids that still capture the correct physical behavior. However, the initial construction of their basis functions and wavelets was particularly compute- and memory-intensive (even if matching the best known computational complexity), prompting the authors to lament the inability to use their approach to offer a multiscale preconditioner that can boost the efficiency of elasticity simulation. Note that this is, in fact, a common issue with reduction methods: they are often too slow to be useful in applications demanding rapid updates.

1.2 Overview

In this paper, we propose a preconditioner and its practical implementation to efficiently solve sparse, large-scale and ill-conditioned linear systems. Our approach has its roots in the recent work of Schäfer et al. [2021] which showed that after a specific change of basis, the Cholesky factor of any discretized elliptic integral operator is sparse up to exponentially small errors. As noted in their Section 6.2, this result extends to differential operators, generalizing closely-related homogenization and multiresolution concepts [Schröder et al. 1978; Gines et al. 1998; Owhadi and Scovel 2019] and enabling the implementation of fast solvers based on incomplete Cholesky factorization. However, being focused on integral equations, they do not develop this idea further, and in particular do not provide a practical implementation. We build upon their work by relating Cholesky factorization to the construction of basis functions adapted to a given differential operator recently described in [Chen et al. 2019]. As a result, we offer a much faster approach to construct the hierarchy of operator-adapted wavelets than the original algorithmic evaluation. We also devise a practical preconditioner for sparse linear systems that is based on an *incomplete* Cholesky decomposition with a specific fine-to-coarse multiscale reordering of the matrix rows and columns and an adjustable sparsity pattern.

In order to offer peak computational performance, we further propose a supernodal implementation to maximize the utilization of level-3 BLAS routines, use a multicolored ordering to exploit shared memory parallelism and formulate an efficient way to deal with non-positive pivots. Finally, we use our incomplete Cholesky factorization as a preconditioner for conjugate gradient and apply the resulting solver to typical linear algebra problems arising in graphics applications. Compared to optimized algebraic multigrid libraries, our solver exhibits an acceleration of *several orders of magnitude* in solving inhomogeneous PDEs over irregular meshes; moreover, its memory usage is linear in the size of the linear system, in sharp contrast to the superlinear memory consumption of direct solvers. Compared to the work of [Schäfer et al. 2021], we offer for the first time a concrete implementation confirming the numerical superiority of this homogenization-based approach to preconditioning; we further observe that a multiscale ordering (i.e., the use of "lazy wavelets") outperforms averaging based approaches on problems with high contrast even in cases *not* covered by their theoretical foundations, since the former is more resilient to the loss of positive definiteness during the factorization.

Outline. In Sec. 2, we spell out the relationship between Cholesky factorization and operator-adapted wavelets in detail, thus establishing the mathematical foundations of our approach originally proposed in [Schäfer et al. 2021]; readers purely interested in implementation aspects can safely skip this part. In Sec. 3, we provide our algorithms to construct an incomplete Cholesky preconditioner through a specific choice of multiscale reordering and sparsity pattern, as well as important technical extensions to improve its practical performance. We then provide in Sec. 4 a series of numerical examples to demonstrate the efficiency and accuracy of a PCG solver using our preconditioner, and present concrete comparisons with current highly-optimized solvers, exhibiting several orders of magnitude of improvement in computational time using only a memory size proportional to the number of degrees of freedom.

2 CHOLESKY-BASED OPERATOR-ADAPTED WAVELETS

Our new preconditioner has its theoretical roots in a homogenization approach based on operator-adapted wavelets. From a positive-definite stiffness matrix computed via a fine grid, the original approach of [Budninskiy et al. 2019; Chen et al. 2019] proposes a variational construction to compute a hierarchy of refinable basis functions best adapted to a given operator $\mathcal{L}$ for scalar- and vector-valued PDEs. Equipped with these resulting nested functional spaces, a Galerkin projection is performed to re-express the stiffness matrix, from which a solution is efficiently found by solving small, well-conditioned and independent linear systems corresponding to the various levels of the hierarchy. In contrast, our approach bypasses this sequential construction, directly generating the hierarchy after a basis transformation. As we show in this section, this new interpretation is theoretically equivalent to previous methods, but far simpler to implement since there is no need to explicitly construct the adapted basis refinement, wavelets and associated system matrices in a fine to coarse fashion: all these operators can be extracted or expressed via sub-blocks of the resulting factors. In order to motivate our practical approach, we first review the concept of operator-adapted wavelets, before explaining its fundamental link to the famous Cholesky factorization.

2.1 Recap of operator-adapted wavelets

Given a partial differential equation $\mathcal{L}u = g$, the usual Galerkin approach defines a set of basis functions forming a finite dimensional functional space $\mathcal{V}$, then discretizes the equation by assuming that its solution lies in $\mathcal{V}$, leading to a linear system of the form $\mathbf{A}\mathbf{u}=\mathbf{g}$. In order to better capture the multiscale nature of many physical phenomena, the wavelet-Galerkin approach further decomposes the whole functional space into a series of q nested subspaces $\{\mathcal{V}^k\}_{k=1..q}$ where $\mathcal{V}^q \equiv \mathcal{V}$ is the original finest space, that is,

$$\mathcal{V}^1 \subset \mathcal{V}^2 \subset \ldots \subset \mathcal{V}^{q-1} \subset \mathcal{V}^q \equiv \mathcal{V}.$$

We call the subspace $\mathcal{W}^k$ the complement of $\mathcal{V}^k$ in $\mathcal{V}^{k+1}$ in the L_2 sense, thus satisfying

$$\mathcal{V}^{k+1} = \mathcal{V}^k \oplus \mathcal{W}^k, \quad k = 1..q-1, \tag{1}$$

where $\mathcal{V}^k$ is spanned by n_k basis functions φ_i^k, $i=1,..,n_k$ and $\mathcal{W}^k$ is spanned by $\eta_k \equiv n_{k+1}-n_k$ wavelets ψ_j^k, $j=1,..,\eta_k$. The basis functions and wavelets are therefore L_2-orthogonal by definition, implying $\int_\Omega \varphi_i^k \psi_j^k \, d\mathbf{x} = 0$. We thus get a multiscale decomposition of the original, fine functional space, where, hopefully, each level captures a certain range of spectrum. However, as soon as one uses spatial grids that are not regular or if the PDE involves heterogeneous and anisotropic coefficients, this general wavelet approach offers little to no improvement: it fails to properly separate the multiscale structure of the solution space [Sudarshan 2005].

If one wants to construct a multiscale decomposition which will lead to a discrete *block-diagonal* version $\mathbb{A}$ of the operator in order to facilitate the resulting numerical solve, the subspaces should be instead *adapted* to the differential operator $\mathcal{L}$ itself: only then can the decomposition leverage the spectral properties of the operator. So a series of recent works [Owhadi 2017; Owhadi and Scovel 2019] proposed to reformulate the orthogonal decomposition using operator-adapted subspaces $\mathcal{V}^k$ and $\mathcal{W}^k$ that must satisfy

$$\mathcal{V}^{k+1} = \mathcal{V}^k \oplus_{\mathcal{L}} \mathcal{W}^k, \quad k = 1,..,q-1, \tag{2}$$

instead, with $\mathcal{V}^q \equiv \mathcal{V}$ and where orthogonality is no longer w.r.t. the L_2-inner product of functions $\int_\Omega f g \, d\mathbf{x}$ but w.r.t. the $\mathcal{L}$-inner product $\int_\Omega f \mathcal{L} g \, d\mathbf{x}$. Then, given any prescribed refinement relationship $\mathbf{M}^k$ (defining multiscale basis functions $\varphi_i^k = \sum_{l=1}^{n_{k+1}} \mathbf{M}_{il}^k \varphi_l^{k+1}$) and its associated refinement kernel $\mathbf{W}^k$ such that $\mathbf{M}^k \mathbf{W}^{k,T} = \mathbf{0}$ (and defining multiscale wavelets $\psi_j^k = \sum_{l=1}^{n_{k+1}} \mathbf{W}_{jl}^k \varphi_l^{k+1}$), *$\mathcal{L}$-adapted basis functions and wavelets* spanning respectively $\mathcal{V}^k$ and $\mathcal{W}^k$ are then constructed via:

$$\boldsymbol{\varphi}_i^k = \sum_{l=1}^{n_{k+1}} \mathbb{M}_{il}^k \boldsymbol{\varphi}_l^{k+1}, \quad \boldsymbol{\psi}_j^k = \sum_{l=1}^{n_{k+1}} \mathbf{W}_{jl}^k \boldsymbol{\varphi}_l^{k+1}, \tag{3}$$

where $\mathbb{M}^k$ is the new $\mathcal{L}$-adapted refinement between scaling functions of two adjacent levels. Exploiting the two conditions

$$\text{(a) } \mathbb{M}^k \mathbb{A}^{k+1} \mathbf{W}^{k,T} = \mathbf{0}, \quad \text{(b) } \mathbb{M}^k \mathbf{M}^{k,T} = \mathbb{I},$$

representing, respectively, (a) the $\mathcal{L}$-orthogonality between adapted basis functions and their wavelets and (b) the locality of adapted basis functions, the adapted refinement is found in closed form as:

$$\mathbb{M}^k = \mathbf{M}^{k,\dagger} \left[\mathbb{I} - \mathbb{A}^{k+1} \mathbf{W}^{k,T} \left(\mathbb{B}^k \right)^{-1} \mathbf{W}^k \right], \tag{4}$$

where $\mathbf{M}^{k,\dagger} = (\mathbf{M}^k\mathbf{M}^{k,T})^{-1}\mathbf{M}^k$ is the (left) Moore-Penrose pseudoinverse and $\mathbb{B}^k = \mathbf{W}^k\mathbb{A}^k\mathbf{W}^{k,T}$. The stiffness matrix of level $k+1$ can then be decomposed into two $\mathcal{L}$-orthogonal parts, with $\mathbb{A}^k$ for the stiffness matrix of level-k adapted basis functions (with $\mathbb{A}^k_{ij} = \int_\Omega \boldsymbol{\varphi}^k_i \mathcal{L}\boldsymbol{\varphi}^k_j d\mathbf{x}$) and $\mathbb{B}^k$ for the stiffness matrix of the associated wavelets (with $\mathbb{B}^k_{ij} = \int_\Omega \boldsymbol{\psi}^k_i \mathcal{L}\boldsymbol{\psi}^k_j d\mathbf{x}$). Starting from the finest level q, the adapted hierarchy can thus be constructed by sequentially applying Eq. (4), with which the adapted basis functions, wavelets and stiffness matrices on each level are assembled. The resulting stiffness matrix is now block-diagonal by construction, with $\mathbb{A} = \mathrm{diag}(\mathbb{B}^{q-1}, ..., \mathbb{B}^2, \mathbb{B}^1, \mathbb{A}^1)$. Details regarding sparsification through thresholding and invariance enforcement through normalization can be found in [Chen et al. 2019].

2.2 Cholesky-based reformulation

While the theoretical time complexity of adapted hierarchical construction is $O(n_q \log^{2d+1} n_q)$ [Owhadi and Scovel 2019], the actual implementation presented in [Chen et al. 2019] involves intensive linear algebra operations throughout the sequential construction based on Eq. (4) with little opportunity for massive parallelization, thus putting a very large constant in front of the time complexity. In this section, we show that one can reformulate the construction of operator-adapted wavelets via Cholesky factorization.

Schur-inspired factorization. Given a square matrix $\mathbf{A}$ whose associated row/column indices can be classified into two disjoint sets f (for fine) and c (for coarse) with $\mathrm{size}(\mathbf{A}) = |\mathsf{f}| + |\mathsf{c}|$, the crucial link connecting operator-adapted wavelet and Cholesky factorization has its roots in the following factorization

$$\underbrace{\begin{pmatrix}\mathbf{A}_{\mathsf{ff}} & \mathbf{A}_{\mathsf{fc}}\\ \mathbf{A}_{\mathsf{cf}} & \mathbf{A}_{\mathsf{cc}}\end{pmatrix}}_{\mathbf{A}} = \underbrace{\begin{pmatrix}\mathbb{I} & \mathbf{0}\\ \mathbf{A}_{\mathsf{cf}}\mathbf{A}_{\mathsf{ff}}^{-1} & \mathbb{I}\end{pmatrix}}_{\bar{\mathbf{L}}} \underbrace{\begin{pmatrix}\mathbf{A}_{\mathsf{ff}} & \mathbf{0}\\ \mathbf{0} & \mathbf{A}_{\mathsf{cc}} - \mathbf{A}_{\mathsf{cf}}\mathbf{A}_{\mathsf{ff}}^{-1}\mathbf{A}_{\mathsf{fc}}\end{pmatrix}}_{\bar{\mathbf{D}}} \underbrace{\begin{pmatrix}\mathbb{I} & \mathbf{A}_{\mathsf{ff}}^{-1}\mathbf{A}_{\mathsf{fc}}\\ \mathbf{0} & \mathbb{I}\end{pmatrix}}_{\bar{\mathbf{L}}^T}. \quad (5)$$

That is, the original matrix $\mathbf{A}$ can be decomposed as the product of a lower triangular matrix $\bar{\mathbf{L}}$, a block-diagonal matrix $\bar{\mathbf{D}}$ and an upper triangular matrix $\bar{\mathbf{L}}^T$, where one recognizes the Schur complement of $\mathbf{A}$ in the middle matrix. By multiplying $\mathbf{A}$ on the left by the inverse $\bar{\mathbf{L}}$ and on the right by the inverse of $\bar{\mathbf{L}}^T$, we thus block diagonalize it, achieving the same effect as what adapted basis functions and wavelets have done for the operator $\mathcal{L}$ in the previous section. Note that the inverse of $\bar{\mathbf{L}}$ has a simple form which will be useful later on:

$$\bar{\mathbf{L}}^{-1} = \begin{pmatrix}\mathbb{I} & \mathbf{0}\\ -\mathbf{A}_{\mathsf{cf}}\mathbf{A}_{\mathsf{ff}}^{-1} & \mathbb{I}\end{pmatrix}. \quad (6)$$

Cholesky factorization. Now let's use the Schur factorization described above to a specific matrix, defined as:

$$\mathbf{A} = \mathbf{P}\mathbb{A}^{k+1}\mathbf{P}^T, \quad \text{with } \mathbf{P} = \begin{pmatrix}\mathbf{W}^k\\ \mathbf{M}^{k,\dagger}\end{pmatrix}, \quad (7)$$

One recognizes the $\mathcal{L}$-adapted stiffness matrix $\mathbb{A}^{k+1}$ of level $k+1$ on which we performed a specific basis transform based on the refinement kernel $\mathbf{W}^k$ and the pseudo-inverse $\mathbf{M}^{k,\dagger}$. From Eq. (5), we need to multiply this newly-assembled matrix on the left by $\bar{\mathbf{L}}^{-1}$ and on the right by its transpose in order to diagonalize it. However, we note through simple calculation that:

$$\bar{\mathbf{L}}^{-1}\mathbf{P} = \begin{pmatrix}\mathbb{I} & \mathbf{0}\\ -\mathbf{M}^{k,\dagger}\mathbb{A}^{k+1}\mathbf{W}^{k,T}(\mathbf{W}^k\mathbb{A}^{k+1}\mathbf{W}^{k,T})^{-1} & \mathbb{I}\end{pmatrix}\begin{pmatrix}\mathbf{W}^k\\ \mathbf{M}^{k,\dagger}\end{pmatrix} = \begin{pmatrix}\mathbf{W}^k\\ \mathbb{M}^{k,\dagger}\end{pmatrix}.$$

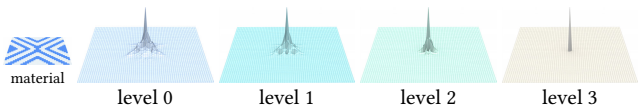


Fig. 2. **Adapted basis functions**. The multilevel Cholesky approach results in an equivalent hierarchy as [Chen et al. 2019], with $\mathcal{L}$-adapted basis functions of decreasing spatial scale from coarsest to finest levels.

Therefore, diagonalizing $\mathbb{A}^{k+1}$ after a basis transform using $\mathbf{P}$ results in $\mathrm{diag}(\mathbb{B}^k, \mathbb{A}^k)$. If the matrix $\bar{\mathbf{D}}$ in Eq. (5) is further decomposed as

$$\bar{\mathbf{D}} = \underbrace{\begin{pmatrix}\widehat{\mathbf{L}}_{\mathsf{f}} & \mathbf{0}\\ \mathbf{0} & \widehat{\mathbf{L}}_{\mathsf{c}}\end{pmatrix}}_{\widehat{\mathbf{L}}}\underbrace{\begin{pmatrix}\widehat{\mathbf{L}}_{\mathsf{f}}^T & \mathbf{0}\\ \mathbf{0} & \widehat{\mathbf{L}}_{\mathsf{c}}^T\end{pmatrix}}_{\widehat{\mathbf{L}}^T}, \quad (8)$$

then the Cholesky factorization of $\mathbf{P}\mathbb{A}^{k+1}\mathbf{P}^T = \mathbf{L}\mathbf{L}^T$ must thus satisfy:

$$\mathbf{L} = \bar{\mathbf{L}}\widehat{\mathbf{L}} = \begin{pmatrix}\widehat{\mathbf{L}}_{\mathsf{f}} & \mathbf{0}\\ \mathbf{A}_{\mathsf{cf}}\mathbf{A}_{\mathsf{ff}}^{-1}\widehat{\mathbf{L}}_{\mathsf{f}} & \widehat{\mathbf{L}}_{\mathsf{c}}\end{pmatrix}. \quad (9)$$

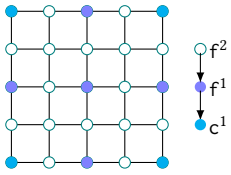
because of Eqs. (5) and (8). Hence, **from the Cholesky factor $\mathbf{L}$ of $\mathbf{P}\mathbb{A}^{k+1}\mathbf{P}^T$, we can directly extract $\mathbb{A}^k$ and $\mathbb{B}^k$** through:

$$\mathbb{A}^k = \widehat{\mathbf{L}}_{\mathsf{c}}\widehat{\mathbf{L}}_{\mathsf{c}}^T, \quad \mathbb{B}^k = \widehat{\mathbf{L}}_{\mathsf{f}}\widehat{\mathbf{L}}_{\mathsf{f}}^T. \quad (10)$$

Note finally that the term $-\mathbf{A}_{\mathsf{cf}}\mathbf{A}_{\mathsf{ff}}^{-1}$ of $\bar{\mathbf{L}}^{-1}$ can be computed through partial back-substitution: from the non-diagonal part $\mathbf{A}_{\mathsf{cf}}\mathbf{A}_{\mathsf{ff}}^{-1}\widehat{\mathbf{L}}_{\mathsf{f}}$ of the Cholesky factor $\mathbf{L}$, a simple back substitution on the top diagonal part will get rid of the term $\widehat{\mathbf{L}}_{\mathsf{f}}$. As a consequence, computing a Cholesky decomposition of $\mathbb{A}^{k+1}$ after a basis transform using $\mathbf{P}$ is equivalent to one level of coarsening in [Chen et al. 2019].

2.3 Multilevel Cholesky

Although we limited our exposition thus far to a single coarsening from a level $k+1$ and a coarser level k, the method can be seamlessly extended to a multilevel construction with a single Cholesky factorization, from which we directly extract the multiscale hierarchy. To achieve this goal, all the finest DoFs used to compute the initial stiffness matrix need to be labeled as multiple nested index sets to form a coarse to fine hierarchy. Let c^1 be the index set denoting the DoFs of the coarsest scale, and c^q the index set of all DoFs of the finest scale, we need to construct nested index sets such that $\mathsf{c}^1 \subset \mathsf{c}^2 \subset ... \subset \mathsf{c}^q$, which can be achieved through farthest point sampling for instance (or through regular subsampling for regular grids, see inset). By defining $\mathsf{f}^k \equiv \mathsf{c}^{k+1} \setminus \mathsf{c}^k$ (where "\" indicates set difference), all DoFs are thus rearranged as a sequence of unrepeated indices through



$$\mathsf{c}^q = \mathsf{f}^{q-1} \cup ... \cup \mathsf{f}^1 \cup \mathsf{c}^1.$$

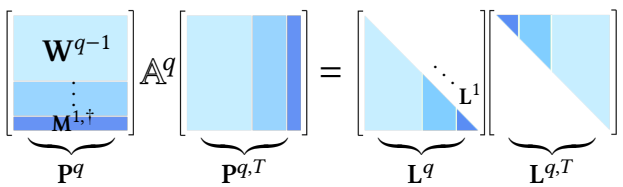


Fig. 3. Multiresolution structure in Cholesky factorization.

Based on this choice of fine-to-coarse hierarchy, we can assemble a transformation matrix $\mathbf{P}^q$ expressed as

$$\mathbf{P}^q = \begin{pmatrix} \mathbf{W}^{q-1} \\ \mathbf{W}^{q-2} \\ \vdots \\ \mathbf{M}^{1,\dagger} \end{pmatrix}, \tag{11}$$

where each row of $\mathbf{W}^k$ encodes the coefficients of basis averaging from nodes in c^q to construct the wavelet on one of the nodes in f^k, and $\mathbf{M}^{1,\dagger}$ for the last level DoFs c^1. With this global transformation matrix, the Cholesky factorization of the fine stiffness matrix $\mathbf{A}^q$ can be done once as

$$\mathbf{A}^q := \mathbf{P}^q \, \mathbb{A}^q \, \mathbf{P}^{q,T} = \mathbf{L}^q \mathbf{L}^{q,T}, \tag{12}$$

where $\mathbf{L}^q$ is the Cholesky factor on the finest level, and factors for all other coarser levels can be extracted from the lower bottom part of $\mathbf{L}^q$, see Fig. 3. Notice that the ordering within levels has been arbitrary until now. Yet, adjusting inter-level ordering is very useful to reduce fill-ins [Amestoy et al. 1996] or improve parallelism via multicolored balancing, as we will exploit in practice in Sec. 3.

Once $\mathbf{A}^q$ has been factorized, we can solve the linear equation $\mathbf{A}^q \mathbf{u}^q = \mathbf{g}^q$ for any right-hand inputs $\mathbf{g}^q$ to get a solution via two back-substitutions; and using the link between Cholesky factorization and operator-adapted wavelets, we can also *directly* extract the hierarchy of $\mathcal{L}$-adapted basis functions and/or wavelets, along with the corresponding block-diagonal stiffness matrix.

2.4 Discussion

The link between Cholesky factorization and operator-adapted wavelets allows us to map the construction of homogenized basis functions which share both spatial and eigenspatial locality [Chen et al. 2019] to the classical Cholesky decomposition, and identifies exactly how all the ingredients of the operator-adapted wavelet approach can be extracted from a Cholesky factor. Additionally, it provides a fine-to-coarse flavor to this well-known construction. But this link, identified and exploited for dense matrices in [Schäfer et al. 2021], opens up a powerful computational basis for a *radically different* way to approach wavelet-based solvers: we can now efficiently construct an *approximate* multiresolution analysis of a given sparse, symmetric, and positive-definite stiffness matrix through an incomplete Cholesky (IC) factorization, a much friendlier algorithm in terms of execution time and memory cost. By computing the sparsity pattern based on the overlap of basis functions from various levels instead of through thresholding to avoid branching, we can significantly accelerate the practical construction as it amounts to stopping the recursive evaluation when the accuracy is good enough. Moreover, we can benefit from a number of computational mechanisms particularly appropriate to perform IC factorization to further improve the wall-clock time: from red-black coloring to allow for parallelism, to the left-looking variant of IC to increase data reuse, to the grouping of columns with identical non-zero structures (supernodes) to leverage highly-optimized level-2 and level-3 BLAS routines, there exists a plethora of ways to increase the performance of sparse IC. Overall, the new construction will be able to coalesce memory access patterns and increasing computational intensity so as to minimize memory access and "squeeze" as much computations out of each access to a matrix element.

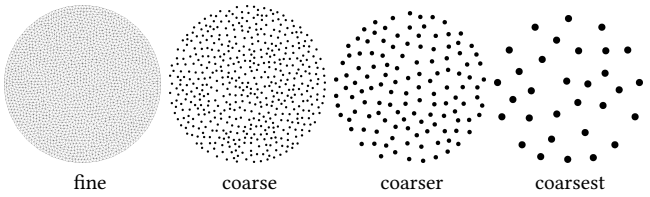


Fig. 4. **Geometric structure of hierarchy**. The maximin ordering is generated by sequentially selecting the farthest DoF from already-selected ones.

3 MULTISCALE INCOMPLETE CHOLESKY

The most straightforward instantiation of the material presented in Section 2 is obtained by using so-called *"lazy wavelets"*, argued in [Chen et al. 2019; Budninskiy et al. 2019] to be most suitable for graphics given its simplest refinement stencil (they referred to it as "Dirac refinement"). We also adopt this choice here as the transformation matrix $\mathbf{P}^q$ for this refinement ends up being a mere *reordering* of the degrees of freedom. One can then perform a specific incomplete Cholesky factorization by the following procedure:

① We reorder the n degrees of freedom of the initial fine stiffness matrix, such that for all $k < n$, the last k degrees of freedom are as far from one another as possible; the smallest distance between two such degrees of freedom is denoted as ℓ_k.

② For $\rho > 1$, we construct a sparsity pattern $S_\rho \in n \times n$ for the IC factors that contains non-zero elements (i, j) with $i \geq j$ for which the distance between the i-th and j-th degree of freedom is smaller than the minimum of $\rho\ell_i$ and $\rho\ell_j$.

③ We then compute the zero-fill-in Cholesky factorization L_ρ of the original stiffness matrix $\mathbf{A}$, ignoring all fill-in occurring outside of the chosen sparsity pattern S_ρ.

The result of this simple and efficient procedure will directly form a preconditioner to be used with, e.g., a conjugate gradient solver. Note that our approach does not use the typical sparsity strategies such as IC(n) based on the initial sparsity structure of the input matrix or thresholding of matrix coefficients: leveraging the multiscale ordering and its implied interaction between basis function supports directly distributes fill-ins for effective coarse-graining and conditioning improvement. Moreover, it can then be reused for any solve on the same initial mesh, and according to our tests that we will report in Sec. 4, it already outperforms previous preconditioners in terms of overall efficiency. We now describe the algorithmic steps of our multiscale IC factorization in detail.

3.1 Step 1: Multiscale ordering

The ordering of all degrees of freedom used by our method is obtained by *reversing* the *maximin ordering* [Schäfer et al. 2021; Guinness 2018] that is sequentially generated through farthest point sampling, i.e., the next index i_k selected is the one satisfying:

$$i_k := \underset{i \in C^q \setminus \{i_1,\ldots,i_{k-1}\}}{\arg\max} \underbrace{\min_{j \in \{i_1,\ldots,i_{k-1}\}} \operatorname{dist}(x_i, x_j)}_{:= \ell_{i_k}}, \tag{13}$$

where $\operatorname{dist}(.,.)$ measures *distances* (and geodesic distances if a curved domain is used). The maximin ordering can be computed in complexity $O(n \log^2 n)$ by using [Schäfer et al. 2021, Alg. 4.1]; alternatively,

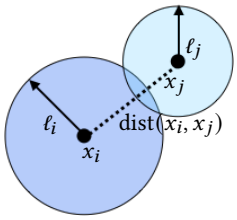
it can be efficiently approximated by using the farthest-sampling procedure described in Alg. 2, which we use in our implementation. Once reversed, the resulting ordering defines an implicit fine-to-coarse multiresolution basis transformation or "*lazy wavelet*": after choosing an arbitrary $h \in (0,1)$, a level can be defined by the set of indices with a value of ℓ_i (defined in Eq. 13) within the same log factor base h, i.e., the l-th level corresponds to the indices $\{i_k : \log_h(\ell_k) \in [l, l+1)\}$, see [Schäfer et al. 2021, Fig. 3.5] for details. If the spatial distribution in $\mathbb{R}^d$ of the n degrees of freedom is quite uniform, one can just pick the first $\lfloor n - n/2^d \rfloor$ consecutive indices of the reversed maximin ordering, then the next $\lfloor n/2^d - n/4^d \rfloor$ ones, etc, to construct a fine-to-coarse hierarchy of the domain.

3.2 Step 2: Sparsity pattern

Following [Schäfer et al. 2020, 2021], we also use a geometric criterion to determine the sparsity pattern of our factorization, i.e., the set of non-zero elements allowed in the Cholesky factor. For a given parameter $\rho > 1$, the sparsity pattern S_ρ is defined through:

$$S_\rho := \{(i,j) \in C^q \times C^q \mid \operatorname{dist}(x_i, x_j) \le \rho \min(\ell_i, \ell_j)\}, \qquad (14)$$

see inset. Here, the parameter ρ allows us to change the trade-off between size of the sparsity pattern and quality of the preconditioner: forcing a sparser Cholesky factor will reduce the computational time needed to perform the incomplete factorization, but decrease the quality of its homogenization properties. In our implementation, the length scales of all nodes within the same level are uniformly specified as the *average spacing* between these in-level points, roughly estimated based on the volume of domain and the number of samples on a level. The covering region of each node will then grow with ρ at the same speed as the estimated support of their adapted wavelets. To avoid unnecessary loss in the raw fill-ins of the input matrix, if a node's estimated basis scale is not large enough to cover all its neighbors, we slightly expand the radius such that no adjacent fill-in is dropped. A similar coarsening-aware length adaption was used in [Chen et al. 2019] for wavelet sparsification, with the same key intuition that the basis support is adapted to the scale of the level, as observed in Fig. 2. The assembly of the sparsity pattern is highly parallelizable and can be performed in near-linear complexity through fast range search algorithms (our implementation uses k-d trees). Once computed, it can be reused for *any* other homogenization task using the same mesh. As illustrated in Fig. 5, the factor becomes denser as ρ increases, but it also more accurately approximates the true stiffness matrix. [Schäfer et al. 2021, Thm. 6.1] shows that for elliptic partial differential operators of order larger than half the spatial dimension d of the problem, truncating the exact Cholesky factor to S_ρ leads to an error that decreases as $O(\log(n)\exp(-C\rho))$ as ρ and/or n increase. On the other hand, the number of nonzeros in the sparsity pattern S_ρ only grows as $O(n\rho^d)$. While their rigorous results for the Laplace operator in two or three dimensions require the use of Haar (as opposed to lazy) wavelets, we do empirically observe the same exponential accuracy of the approximation.



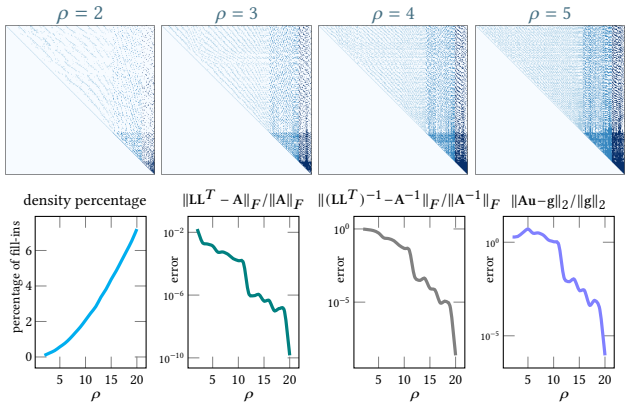


Fig. 5. **Sparsity pattern, error and factor density**. For a Laplacian matrix on a regular grid of size $16k \times 16k$, the general fill-in distribution regarding ρ is illustrated. Due to multiscale spatial supports of basis functions, coarser levels (darker blue) are always denser due to their large supports. With ρ increasing, errors of Cholesky approximation, inverse operator approximation, and solution approximation decay exponentially.

3.3 Step 3: Zero fill-in incomplete Cholesky factorization

Now that a prescribed sparsity pattern S_ρ is provided that records the position of fill-in elements based on the control parameter ρ, the incomplete Cholesky (IC) factorization can directly be performed according to Alg. 1. The factorization is done column by column from left to right in a sequential order. Computationally, it mainly consists of inner products between sparse rows (Alg. 1 Line 6), leading to an asymptotic complexity reduced from $O(n^3)$ to $O(n\rho^{2d})$ compared to a full dense Cholesky factorization, where d is the spatial dimension of the problem (see Fig. 6).

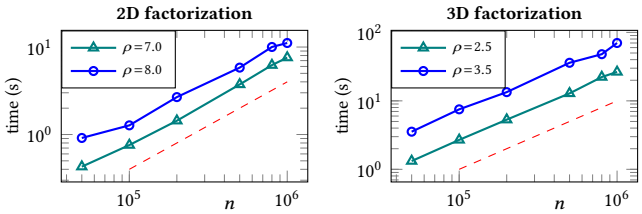
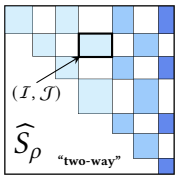


Fig. 6. **Scalability.** In 2D and 3D, our IC factorization time matches the expected $O(n\rho^{2d})$ time complexity for a matrix size $n \times n$ and a sparsity parameter ρ; the dashed line indicates a slope of 1 in this log-log plot.

3.4 Fast implementation

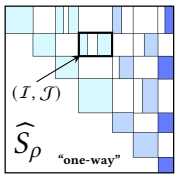
The main objective of this work (and its main difference with [Schäfer et al. 2021]) being to apply our IC algorithm to derive a fast and robust solver for huge, sparse linear problems, we discuss three additional implementation details which significantly improve computational efficiency on modern hardware. *First*, we use supernodes, allowing us to re-express the factorization in terms of BLAS operations. *Second*, we apply a multicolored ordering which maximizes the number of operations that can be performed in parallel. *Third*, we propose a numerical procedure to apply if a non-positive pivot is encountered that does not affect the preconditioning properties of the resulting factorization significantly.

Supernodal implementation. Performing IC column by column involves indirect addressing, slowing down the factorization of large scale problems. A supernodal implementation is one of the most effective ways to improve cache coherence — and thus computational performance. In essence, supernodal factorization just amounts to a block-wise version of the column-by-column scheme: multiple columns with similar non-zero structure (called supernodes) are processed together. The crucial issue here is the efficient construction of supernodes (sets of DoFs that appear consecutively in the elimination ordering) so that the columns belonging to each supernode have almost the same sparsity pattern. In our implementation, we tested two possible supernodal approaches. The first one, which we will refer to as the *two-way supernodal approach*, identifies the supernodes through distance-based clustering: for each level, if two nodes i and j satisfy $\text{dist}(x_i, x_j) < \frac{\rho}{2} \min(\ell_i, \ell_j)$ then we cluster these two nodes into a same supernode. After identifying all supernodes on each level, the sparsity pattern is finally modified to form dense blocks, resulting in a supernodal sparsity pattern $\widehat{S}_\rho$:



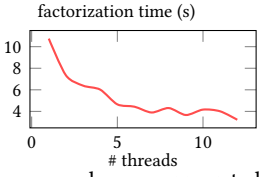
$$\widehat{S}_\rho := \{(\mathcal{I}, \mathcal{J}) | \exists i \in \mathcal{I}, \exists j \in \mathcal{J} \text{ and } (i, j) \in S_\rho\}.$$

Finally, in order to avoid inserting too many fill-ins that might not pay off in terms of convergence acceleration, we set an upper bound on the supernodal size (64 for all tests). With this supernodal structure, the scalar product, scalar square root and scalar inversion in Alg. 1 Line 6, Line 10 and Line 12 are respectively replaced by matrix-matrix product, dense Cholesky factorization, lower- and upper-triangular solves, using highly optimized BLAS routines (we use the Intel MKL library). As a result, indirect addressing is limited to the few supernodal columns and the ratio of FLOPS per memory access increases, drastically reducing computational time for larger values of ρ. While this approach is relatively easy to incorporate, we can further improve performance and memory use by implementing what we will refer to as the *one-way supernodal approach*, partially inspired by the supernodes used in Cholmod [Davis and Hager 2009]. Instead of considering each rectangular block $(\mathcal{I}, \mathcal{J})$ as a dense block, our one-way supernodal approach only fills dense subblocks of columns: storing the upper triangular Cholesky factor as a sparse matrix with a column-major layout, the one-way approach traverses all non-empty supernodal blocks and fully fills a column *iff* there is at least one nonzero fill-in in the raw sparsity S_ρ for that column. All fill-ins in each supernodal row can then be stored in a contiguous memory block and each supernodal cell only needs to additionally record how global column indices are mapped to local ones. While additional copies of memory blocks are needed to scatter the result of BLAS matrix products during factorization, this one-way treatment saves about half of the fill-ins needed by the two-way approach, thus improving both efficiency and memory usage. Consequently, this option is the recommended default choice in practice.



Multi-threading. Due to sparsity, supernodal columns without dependency can be processed in parallel, which is often referred to level scheduled parallelization. (Note that the term "level" here is conceptually very different from the notion of "mesh level" used in multiscale representation.) To identify which nodes can be eliminated in parallel, a direct acyclic graph is constructed based on the supernodal sparsity pattern $\widehat{S}_\rho$. In the symbolic stage of factorization, parallelizable nodes are grouped level by level. Both numerical factorization and substitution can now be run in parallel. For more detailed explanations on multi-threaded Cholesky factorization, see [Naumov 2012]. In practice, our parallelized algorithm scales almost linearly in the size of the input matrix, see Fig. 6.

Multicolored ordering. To increase parallelism of nodes without altering the original multiscale structure we constructed, the ordering of supernodes are adjusted individually *within each scale* through a classical greedy multicoloring algorithm: for each scale, the inner-level supernodes are aggregated into multiple clusters according to their pre-assigned color, as long as any two adjacent nodes have different colors. While classical fill-reducing orderings such as nested dissection greatly limit the use of multicolored orderings, our approach gives complete freedom on the ordering within each scale, resulting in a highly parallel algorithm. The time cost of factorization changes with the number of used threads, as depicted in the inset.



Fixing breakdowns. For strictly symmetric positive definite matrices, a complete Cholesky factorization will never run into a non-positive pivot. However, this is no longer true for IC as a possibly large amount of fill-in elements are dropped. While there exist classes of matrices where IC is guaranteed not to encounter negative pivots (M-matrices [Meijerink and van der Vorst 1977] and H-matrices, also called generalized diagonal dominant matrices [Scott and Tůma 2014c] most notably), the vast majority of matrices used in graphics problems, such as Laplacian matrices or elasticity stiffness matrices computed on irregular grids, do not fall into one of these particular classes. A few existing strategies can be applied to remedy this issue of IC, the simplest instance being to add a global diagonal shift matrix $\alpha \mathbb{I}$ ($\alpha > 0$) and reboot the factorization whenever a negative pivot is detected [Scott and Tůma 2014b]. If it fails again, α is further increased, until success. In practice, we find that global diagonal shifts are not appropriate in our case as they drastically reduce the efficacy of the preconditioner due its effect on the spectrum of the factor. Instead, when we encounter a negative pivot, we only add shifts to the diagonal entries of those columns that affect the diagonal entry of the failed column. Only these diagonally modified columns as well as their descendant columns have to be re-factorized, while the remaining factorized ones can be safely left as they were—a idea similar to the partial updates from [Herholz and Sorkine-Hornung 2020]. If other negative pivots are found later on, the above procedure is repeated and each relevant column is incrementally shifted by a factor of $2^{p_i}\alpha$ until success, where p_i is the number of times we tried to modify column i to fix breakdowns. Another known approach to reduce the number of reboots is to apply a diagonal scaling to the matrix before factorization. To this end, scaling factors such as the inverse of the diagonal elements or the L_2-norm of rows have been investigated in the literature [Scott and Tůma 2014a]. In our implementation, we prescale the matrix by the inverse of the diagonal entries, and α is set to 10^{-4}.

Input: SPD matrix $\mathbf{A}$ and sparsity pattern S_ρ.
Output: Incomplete Cholesky factor $\mathbf{L}$ such that $\mathbf{A} \approx \mathbf{L}\mathbf{L}^T$

```
place non-zeros of A into L according to S_ρ;
for i ← 1 to n do
    for j ← i to n do
        for k ← 1 to i − 1 do
            if (j,i), (j,k), (i,k) ∈ S_ρ then
                L(j,i) ← L(j,i) − L(j,k)L(i,k);
            end
        end
    end
    L(i,i) ← √L(i,i);
    for j ← i + 1 to n do
        L(j,i) ← L(j,i)/L(i,i);
    end
end
```

Algorithm 1: Left-looking incomplete Cholesky factorization.

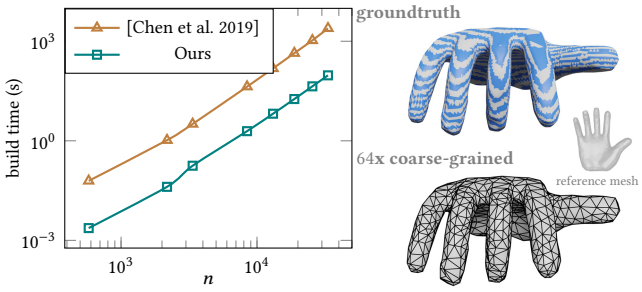


Fig. 7. **Cholesky-based numerical coarsening.** Compared to the hierarchical approach of [Chen et al. 2019], a full Cholesky factorization can significantly improve the computational time required to precompute all the adapted basis functions and wavelets for a given operator: on this elastic deformation example using a relatively small tetrahedral mesh of a hand, two orders of magnitude improvement is already observed.

4 RESULTS AND DISCUSSION

In this section, we show that our multiscale Cholesky factorization provides a much faster alternative to the level-by-level (sparsified or complete) construction of operator-adapted wavelets, and that the multiscale IC version we introduced results in a preconditioner which, once integrated in a preconditioned conjugate gradient solver, drastically outperforms existing solvers for large-scale, ill-conditioned problems, while using a memory size that is only linear in the number of DoFs in contrast to direct solvers. Our algorithms were implemented in C++, and our numerical tests were run on a laptop with two 6-core 2.3GHz Intel© Xeon© E5-2630 CPUs with 128 Gb of RAM. In all comparisons, the convergence condition for PCG is $\|Ax - b\|_2/\|b\|_2 < 10^{-12}$, a typical tolerance for most applications.

4.1 Applicablity

First, we quickly show a few applications of our Cholesky-based wavelet construction. By switching between a full and an incomplete factorization, our method can be customized to solve common tasks arising in graphics applications.

Fig. 8. **Non-linear dynamics**. The speedup obtained from solving a single ill-conditioned system with our method eventually leads to significant accelerations when applied to nonlinear elasticity simulation, involving repeated linearizations and solves for time integration.

Operator-adapted wavelet construction with full Cholesky. The equivalence between Cholesky factorization and the closed-form approach to constructing an operator-adapted hierarchy of basis functions and wavelets guarantees that they share the same ability when applied for effective coarsening. However, the original sequential construction proposed in [Chen et al. 2019; Budninskiy et al. 2019] becomes inefficient for large scale problems in terms of both memory and computational time, and thus various strategies to induce basis sparsification have to be applied. Unfortunately, sparsification will inevitably hurt accuracy, and has very little effect on memory requirements. Additionally, while these two methods have the same asymptotic complexity, their effective complexity (i.e, the magnitude of the constant in front of the asymptotic complexity) is drastically different: the Cholesky approach offers a way to avoid explicitly calculating and storing all relevant adapted operators, which greatly optimizes the construction of the operator-adapted hierarchy both in terms of computational time and in terms of memory consumption. As demonstrated by Fig. 7 on a 3D elasticity problem, the Cholesky factorization optimizes the performance of the complete hierarchy construction by at least two orders of magnitude of acceleration, even on rather small problems. Since a complete Cholesky factorization is used, we can apply any known permutation of columns and rows to reduce fill-ins, and we used the approximate maximal degree (AMD) ordering within each level in this case. The resulting benefit on performance allows our algorithm to directly conduct coarse-graining of complex composite materials with high contrast of materials for much larger models than previously reported. Note that even for aggressive sparsification in the original construction of operator-adapted wavelets, direct Cholesky factorization is still about one order of magnitude faster.

Solving linear and non-linear systems in graphics. Combining an approximate multiscale solver with a conjugate gradient solver is widely recognized as an efficient way to solve large scale problems in graphics. Our multiscale incomplete Cholesky factorization introduced in Sec. 3 can also be used for this purpose given its good asymptotic accuracy for small ρ values (see Fig. 5). As a result, many applications in computer graphics can benefit from this novel preconditioning technique, as long as symmetric positive definite matrices are involved. Here we discuss two concrete examples for illustration: elasticity simulation and image processing.

- *Fast elastic simulation.* Non-linear elasticity simulation of inhomogeneous materials is notoriously difficult from a numerical standpoint. Given that time integration usually involves repeatedly solving very large linearized ill-conditioned systems, preconditioned iterative solvers are the only viable approach but they

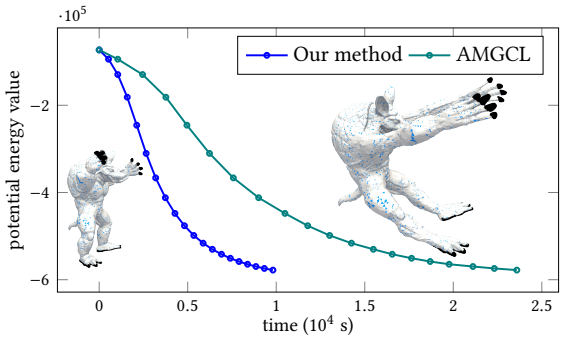


Fig. 9. **Non-linear quasi-statics**. An armadillo is stretched via lateral gravity with a few nodes (marked in black) are fixed at their rest positions. A trust-region nonlinear optimization is implemented to determine the direction and length for each step, involving a linear system solve using for our IC preconditioner or AMGCL; timing of the first 20 iterations are plotted.

Fig. 10. **Edge-preserving decomposition**. Our method can also generate edge-preserving multiscale decompositions of an image, which can be further used for applications such as tone mapping, detail exaggeration, etc. All these images presented here have a size of 1280×800, and use gradient-based diffusion coefficient a_x and a_y, for two different diffusion times λ.

suffer from slow convergence in practice, particularly for problems on unstructured grids or/and for heterogeneous materials. Our preconditioner offers significant improvement in computational timings: for the non-linear (compressible neo-Hookean) elastodynamic example in Fig. 8 for instance, we simulate a bird model with $490k$ cells and $300k$ degree of freedoms, using two different materials with a contrast ratio of Young's moduli equal to 10^4. The simulation takes about 120 seconds to solve each frame with an IC-preconditoned conjugate gradient solver, while a typical AMG solver, e.g. AMGCL we used for later comparisons, is about 5 times slower on this example. Besides dynamics, our preconditioner can also be used to accelerate quasi-statics: in Fig. 9, we stretch an armadillo models with about $340k$ nodes (thus over $1M$ degrees of freedom), for which each tetrahedron is randomly assigned one of two neo-Hookean materials whose contrast between Young's moduli is 10^4. In each step of our Newton descent to find the final shape, we project the neo-Hookean elastic stiffness matrix to enforce semi-positive definiteness [Teran et al. 2005]. The linear solves of the Newton descent exhibit a 2.5× speedup on average compared to AMGCL, with larger speed-ups for smaller error tolerances.

- *Image processing*. Our method can also benefit image processing tasks, or even video stream processing. For instance, a number of approaches have been designed to compute edge-preserving decompositions of images [Farbman et al. 2008; Fattal 2009; Paris et al. 2015] in order to perform dynamic-range compression, tone mapping, multiscale detail enhancement or edge-preserving smoothing to cite a few common examples. One way to provide such a scale-space decomposition is through inhomogeneous diffusion [Perona and Malik 1990] at different scales, where the diffusion coefficients are based on the log illuminance of the input image. Our method can process any number of different images of the same size once our sparsity pattern has been precomputed. We can produce hierarchical edge-preserving decompositions by simply performing linear solves corresponding to an implicit integration of the non-linear diffusion for various times λ as demonstrated in Fig. 10. In this case, our solver pays off compared to multigrid methods because the multiscale dynamics of the non-linear coefficients of the Laplacian prevent the traditional coarsening methods from providing adequate homogenization.

4.2 Quantitative analysis of IC

We now focus on providing detailed evaluations of various aspects of our preconditioner, to better understand when and how it excels.

Empirical choice of ρ. As the most important parameter in our algorithm, ρ is responsible for balancing robustness, factorization speed, and convergence. To be more specific, with an increase of either the problem size n or the material contrast, the algorithm tends to have more breakdowns when using too small a ρ as depicted in Fig. 11, which can slow down the efficacy of our preconditioner. However, slightly increasing ρ removes most breakdowns, and the few remaining negative pivots can be effectively fixed via our partial diagonal shifting strategy. For the sake of accuracy, the parameter ρ should be selected as large as possible, but considering time and storage complexity in ρ as well as the overall balance between factorization and convergence, we suggest to pick ρ in the 6.5 to 8.5 range in 2D and 2.5 to 4.0 in 3D in practice, which leads to peak performance across all the examples we tried.

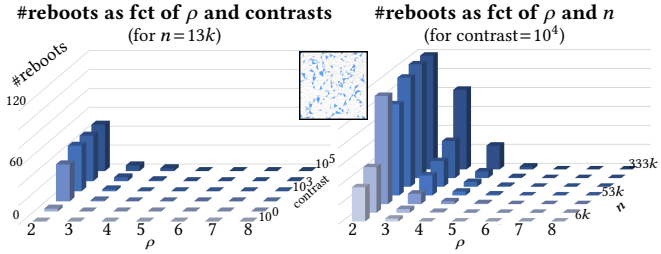


Fig. 11. **Reboot frequency**. For 2D elasticity, as the problem size n or material contrast increases, too small a sparsity parameter ρ can lead to frequent breakdowns (i.e., non-positive pivots) during incomplete factorizations, requiring reboots using our partial diagonal shifting strategy. However, their frequency drops to zero as long as ρ is chosen to be not too small.

Table 1. **Statistics**. We provide timings (t_{factor}) of our zero fill-in incomplete Cholesky factorization for various examples in this paper, along with their spatial dimension, number n of degrees of freedom, and sparsity parameter ρ. Precomputations involve sparsity pattern construction (t_{spars}), supernode aggregation and multicoloring (t_{super}) and level scheduling for parallelism (t_{sched}), which are done once and for all for a given mesh. Note that nnz(**A**) (resp., nnz(**L**)) is the number of non-zeros of the original operator **A** (resp., its IC factor **L**), while memory indicates the total amount of RAM used.

example	dim.	n	ρ	nnz(**A**)	nnz(**L**)	t_{spars} (s)	$t_{\text{super}}(s)$	t_{sched} (s)	t_{factor} (s)	memory
Poisson	2D	5×10^5	7.5	3493289	226089245	5.434	17.81	1.04	4.175	1.69G
Poisson	2D	1×10^6	7.5	6992002	456876935	11.93	40.90	2.56	7.841	3.42G
Poisson	3D	5×10^5	3.2	7246747	468544044	11.11	36.73	2.44	20.33	3.50G
Poisson	3D	1×10^6	3.2	14761198	980299947	26.06	83.82	6.05	42.94	7.32G
Elasticity	2D	5×10^5	7.5	6984008	432016172	5.368	23.48	1.68	9.661	3.23G
Elasticity	2D	1×10^6	7.5	13973156	871565600	11.75	55.12	4.26	22.09	6.52G
Elasticity	3D	5×10^5	3.2	21813147	1225328286	13.70	58.76	5.42	83.67	9.14G
Elasticity	3D	1×10^6	3.2	43327773	2527263530	28.45	116.7	12.3	171.9	18.85G

Statistics. In Table 1, we provide statistics for our IC preconditioner, including the number of fill-ins collected from all supernodes. We also include the wall clock times for the construction of the sparsity pattern (Sec. 3.2), the aggregation and coloring of supernodes and the level scheduling for improved parallelism (Sec. 3.4), the actual numerical factorization time, as well as the memory size. Both time and storage costs are linear in the problem size n given a specific ρ. Small symbolic precomputations are involved in constructing the sparsity pattern and scheduling; their time complexity grows almost linearly with n, but they are reusable for *any* other linear solves issued from the same mesh.

4.3 Comparisons with existing solvers

We compare computational costs of various linear solvers with our approach. Note that we count the execution time of both our IC factorization and PCG iterations, while ignoring precomputations since these are only needed once for a given mesh, independent on the number or nature of problems solved on them.

With Cholmod. A highly optimized implementation of the full Cholesky factorization for direct solving of symmetric positive definite systems, Cholmod [Chen et al. 2008] is widely used in graphics. For small to intermediate size problems, Cholmod is vastly preferable for its computational efficiency and stable numerical behaviors regardless of the matrix conditioning. However, it has a computational complexity in $n^{3/2}$ in 2D and n^2 in 3D, and a memory requirement of $n \log n$ in 2D and $n^{4/3}$ in 3D [Davis 2006][Section 7.6]. Consequently, Cholmod cannot be applied to huge problems, which is particularly limiting in 3D. Instead, our method is linear in complexity (see Fig. 6) and memory usage. In Fig. 12, we solve a 3D inhomogeneous Poisson equation on irregular Delaunay meshes of increasing sizes, using PCG with our preconditioner vs. Cholmod. Although our approach cannot compete with Cholmod for small sizes, the curves clearly demonstrate that these two methods have very different complexity trends, making our approach preferable for both speed and storage. Moreover, when n exceeds a million, Cholmod simply fails to factorize the matrix because the number of non-zero fill-ins needed by the full Cholesky factor has already exceeded the upper bound set by the library for array indexing.

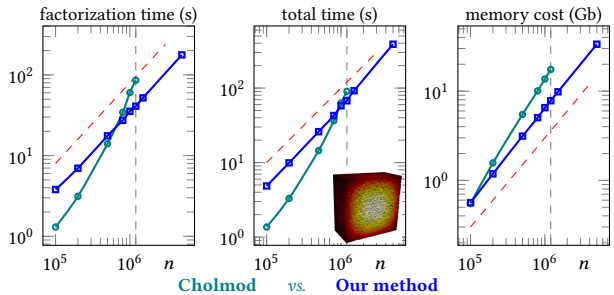


Fig. 12. **Direct vs. iterative solves.** For a 3D Poisson solve, Cholmod scales non-linearly in the linear system size n for factorization time, total solve time (which include factorization and back-substitution), and memory use, and fails for $n > 1$M; instead, a PCG-based iterative solve using our preconditioner exhibits consistent linear behaviors on all three measurements.

With AMG preconditioners. We also compare our method with two commonly-used AMG implementations: AMGCL [Demidov 2019] and Trilinos [Trilinos 2020]. As a benchmark, we use a classical Poisson problem as well as a linear elasticity problem:

$$\begin{cases} \mu(x)\Delta u(x) + \dfrac{\mu(x)}{1-2\nu}\nabla(\nabla\cdot u(x)) = g(x), \\ u(x) = 0 \quad \forall x \in \mathcal{B}, \end{cases} \tag{15}$$

where μ is the spatially variant Young's modulus and ν is the Poisson ration. $\mathcal{B}$ is a user-imposed constrained DoFs to ensure a unique solution. In these experiments, the Trilinos solver is set to use the smoothed aggregation coarsening strategy [Vaněk et al. 1996] to construct its prolongation operator, and a symmetric Gauss-Seidel relaxation. For AMGCL, we found that its smooth aggregation implementation can lead to singular systems, and is thus not numerically stable; instead, we picked the Ruge-Stüben method [Stüben 2000] for coarsening, and sparse inverse approximation [Grote and Huckle 1997] for relaxation. The total number of levels built by the AMG preconditioners is automatically determined by their respective code based on the input matrix; our method automatically produces 4 to 9 levels, depending only on the example size. For both AMG solvers, pre- and post-relaxation are performed once as using more relaxations does not pay off in terms of convergence for ill-conditioned systems. The bimaterial is generated by randomly assigning stiff regions, occupying around 1/8 in 2D (resp., 1/64 in 3D) of the domain. Depending on the contrast between soft and stiff regions, the system's condition number can vary widely. Overall, our Cholesky-based preconditioner is faster than the two AMG implementations

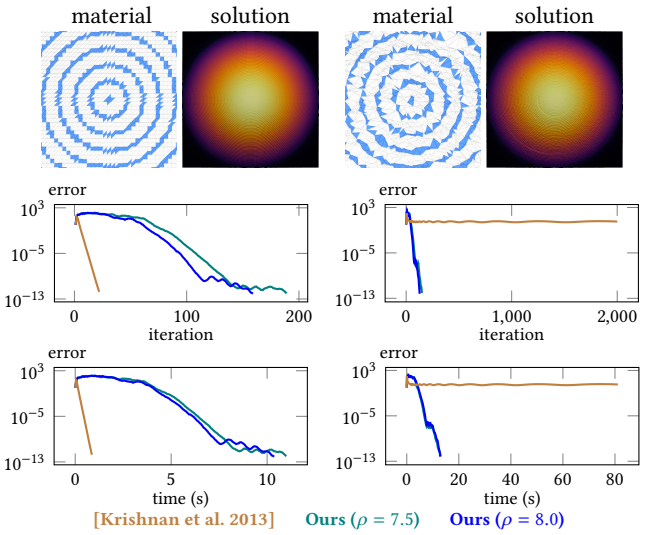


Fig. 13. **Comparisons with [Krishnan et al. 2013]**. For $160k$ points, either on a regular grid (left) or uniformly distributed in a square (right), a Delaunay triangulation of these points is computed, and an inhomogeneous Laplace operator based on a heterogeneous medium made of stiff (blue) and soft (white) materials (with a stiffness contrast of 10^4) is discretized, then a Poisson problem is finally solved. While the regular grid case leads to a M-matrix on which their solver excels, as soon as the mesh is unstructured, performance and accuracy are severely affected.

we tried in all examples except when the linear operator is a M-matrix, which seems to happen only for Poisson problems on regular grids. As mesh size and contrast increase, our preconditioner leads to orders of magnitude improvement in computational time for a given error tolerance, especially for elasticity where the stiffness matrix is more involved than the Laplacian. It confirms a known flaw of multigrid methods: their restriction and prolongation operators lose efficacy when high contrast is present in the PDE coefficients. In sharp contrast (pun intended), the homogenization roots of our approach pay off handsomely in such cases.

With [Krishnan et al. 2013]. For Laplacian matrices, a preconditioner was proposed by Krishnan et al. [2013]. For comparisons, we solve on different meshes an inhomogeneous Poisson equations

$$\begin{cases} -\nabla \cdot (a(x)\nabla u(x)) = g(x), \\ u(x) = 0 \quad \forall x \in \partial\Omega. \end{cases} \tag{16}$$

On a regular grid, the resulting Laplacian matrix only contains non-positive off-diagonal entries, which is a particular instance of M-matrices. In this very specific case, their method guarantees that the condition number of the resulting preconditioned operator is nicely bounded by simply iteratively dropping the weakest connections. Their solver thus achieves remarkably fast convergence, see Fig. 13. However, once the Laplacian matrix has positive off-diagonal entries, their method can even fail to converge. In practice, positive off-diagonal entries are generally hard to prevent since arbitrarily-shaped element can occur quite often, even for Delaunay triangulations as demonstrated in Fig. 13. In contrast, our method is not quite sensitive to the matrix coefficients and works much more consistently for different meshes. In both problems, our method takes less than 20 seconds to get a solution on this vein-structured material. Although we are far from rivaling with coefficient-aware methods such as [Krishnan et al. 2013] for solving problems with M-matrices, our method can deal with a far wider range of problems without significant loss of efficiency.

When to use our IC-based preconditioner? In sum, our multiscale preconditioner is particularly recommended (and often times, the only viable solution) when the linear system is very large (as direct solvers require too much memory), or when the operator is ill-conditioned, but not a M-matrix. Therefore, our contribution adds to the arsenal of linear solves available for graphics applications.

4.4 When does it pay off to be lazy for a wavelet?

The theoretical results of [Schäfer et al. 2021] on the exponential accuracy of the Cholesky factorization and the optimal approximation property of the coarse grained operator require the use of *averaging* as opposed to the traditional *subsampling* approach when the order of the PDE does not exceed the spatial dimension. Therefore, it does not technically cover the use of lazy wavelets applied to Laplace operators (or variants thereof) in two or three dimensions. However, as illustrated in Fig. 5, we numerically observe exponential decay of the approximation error of the sparse factorization using lazy wavelets, even though this result has not been proven in the literature outside of the translation-invariant setting treated by [Schröder et al. 1978]. Since lazy wavelets are also easier to construct and seem to be much less susceptible to encountering negative pivots during the factorization, we recommend their use when employing our approach as a fast solver. On the other hand, the results of [Schäfer et al. 2021] on near-optimal accuracy of the coarse-grained operator appear to be sharp, meaning that its approximation property deteriorates when applying lazy wavelets to second-order equations. For large Poisson problems with homogeneous coefficient fields we also observe (see inset) that using wavelets based on bilinear basis functions, as opposed to subsampling, improves the convergence speed of conjugate gradient. It is therefore desirable to develop methods for improving the robustness of the incomplete Cholesky factorization in averaging-based multiresolution basis. Efficient implementation of supernodes in this setting may require using a non-standard form of Cholesky factorization as introduced by [Gines et al. 1998], but we leave further investigation to future work.

ρ	#iter (bilinear)	#iter (lazy)
2.0	184	433
2.5	135	278
3.0	85	134
3.5	80	117
4.0	67	72

5 CONCLUSIONS

In this paper, we described the practical implementation of a novel preconditioner to efficiently solve large-scale and ill-conditioned linear systems. Despite its roots in the literature on numerical homogenization and operator adapted multiresolution analysis [Schäfer et al. 2021][Section 6.2], it can be cast as a standard zero fill-in incomplete Cholesky factorization. This simple formulation allows for implementations that readily exploit parallelism and level-three BLAS routines. Our current implementation, available at `https://gitlab.inria.fr/geomerix/ichol`, is solely on CPU and is undoubtedly improvable, but it already outperforms highly-optimized libraries for concrete examples that we took straight out of usual

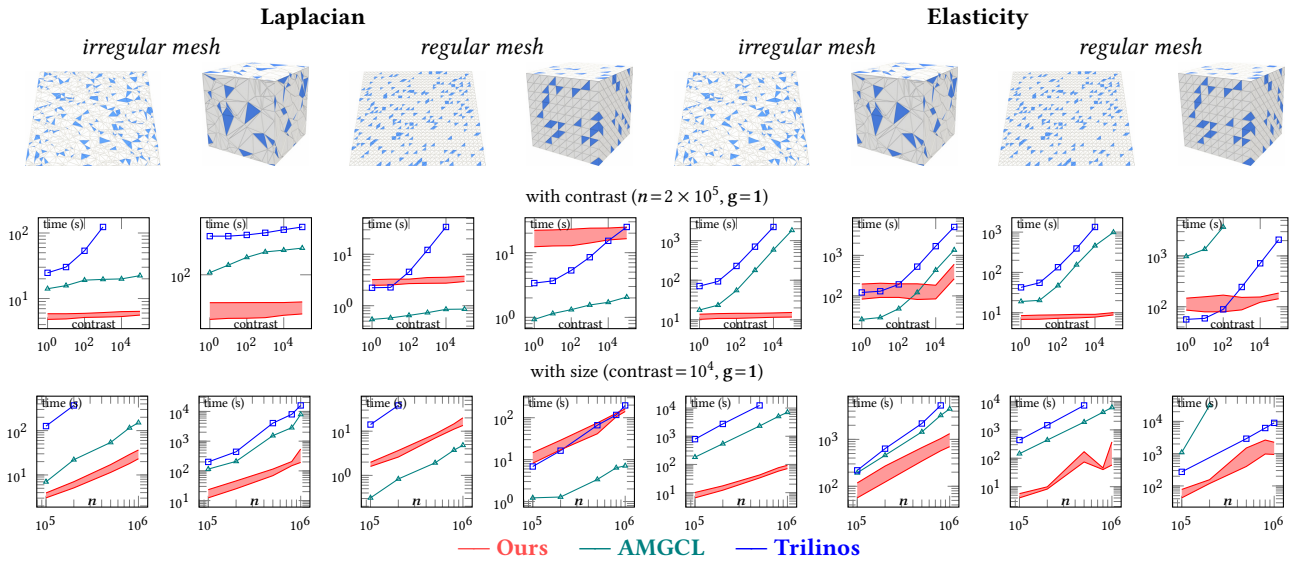


Fig. 14. **Comparisons with AMG libraries**. We take two implementations of AMG from two libraries, namely AMGCL [Demidov 2019] and Trilinos [Trilinos 2020], with a number of pre- and post-sweeps set to 1. Figures indicate time costs (including factorization and PCG iteration times) as a function of material contrast. Our method is much less sensitive to contrast and problem size, and is particularly efficient when the size n becomes large and/or for bad condition numbers. For our method, timings are within the red region depending on the actual value of ρ, for which we used the range $[6.5, 8.5]$ in 2D and $[2.5, 4.0]$ in 3D. All meshes are generated by Delaunay triangulation.

graphics applications. Fine-grained control of thread allocations between parallelizable levels may further help with efficiency, while leveraging GPUs for massive parallelization of both incomplete Cholesky preconditioning and matrix-vector product in PCG would be attractive to further boost the performance of our solver. Although increasing the sparsity parameter ρ shows a trend towards less frequent breakdowns during the factorization, finding effective methods to either improve the efficiency of our current treatment or to completely avoid the occurrence of negative pivots would also add to the efficiency of our approach. We also limited our evaluation to lazy wavelets, but the approach is fundamentally valid on arbitrary refinement stencils, and a clear analysis of the pros and cons of using higher order wavelets remains needed. Finally, while we purposely adopted an approach that is totally blind to the matrix coefficients, other solvers managed to be extremely efficient on M-matrices by exploiting these coefficients instead. Finding ways to construct a hybrid version that uses our multiscale precomputation and a runtime adaptation to the matrix coefficients may maintain our clear efficiency advantage even when specific types of matrices are encountered. Finally, applying our method to preconditioning non-symmetric matrices via incomplete LU factorization is also an interesting future research direction, which can be used to efficiently solve systems arising from advection-diffusion problems.

ACKNOWLEDGMENTS

The authors wish to thank Houman Owhadi for supporting this project from its onset, and Rasmus Tamstorf for early discussions. Partial funding for JC and JH was provided by National Key R&D Program of China (No. 2020AAA0108901) and NSFC (No. 61732016). FS was partially supported by AFOSR grant FA9550-18-1-0271 and ONR grant N00014-18-1-2363. MD gratefully thanks the GeoViC team for their hospitality, and acknowledges additional funding from Inria. Images of Fig. 10 are courtesy of `pixabay.com`, the Armadillo model from Fig. 9 is courtesy of Stanford University, the hand model from Fig. 7 is courtesy of AIM@SHAPE, while the bird model from Fig. 8 is courtesy of the authors of [Liu et al. 2018].

REFERENCES

Raymond Alcouffe, Achi Brandt, Joel Dendy, Jr., and James W. Painter. 1981. The Multi-Grid Method for the Diffusion Equation with Strongly Discontinuous Coefficients. *SIAM J. Sci. Stat. Comput.* 2, 4 (1981), 430–454.

Patrick R Amestoy, Timothy A Davis, and Iain S Duff. 1996. An approximate minimum degree ordering algorithm. *SIAM J. Matrix Analysis and Applications* 17, 4 (1996), 886–905.

Achi Brandt, James Brannick, Karsten Kahl, and Irene Livshits. 2011. Bootstrap AMG. *SIAM J. Sci. Comput.* 33, 2 (2011), 612–632.

Max Budninskiy, Houman Owhadi, and Mathieu Desbrun. 2019. Operator-adapted wavelets for finite-element differential forms. *J. Comput. Phys.* 388 (2019), 144–177.

Desai Chen, David IW Levin, Wojciech Matusik, and Danny M Kaufman. 2017. Dynamics-aware numerical coarsening for fabrication design. *ACM Trans. Graph.* 36, 4, Article 84 (2017).

Jiong Chen, Hujun Bao, Tianyu Wang, Mathieu Desbrun, and Jin Huang. 2018. Numerical Coarsening Using Discontinuous Shape Functions. *ACM Trans. Graph.* 37, 4, Article 120 (July 2018).

Jiong Chen, Max Budninskiy, Houman Owhadi, Hujun Bao, Jin Huang, and Mathieu Desbrun. 2019. Material-adapted refinable basis functions for elasticity simulation. *ACM Trans. Graph.* 38, 6 (2019), 161.

Yanqing Chen, Timothy A Davis, William W Hager, and Sivasankaran Rajamanickam. 2008. Algorithm 887: CHOLMOD, supernodal sparse Cholesky factorization and update/downdate. *ACM Trans. Math. Software* 35, 3 (2008), 1–14.

Timothy A Davis. 2006. *Direct methods for sparse linear systems.* SIAM.

Timothy A. Davis and William W. Hager. 2009. Dynamic Supernodes in Sparse Cholesky Update/Downdate and Triangular Solves. *ACM Trans. Math. Softw.* 35, 4, Article 27 (2009).

Denis Demidov. 2019. AMGCL: An efficient, flexible, and extensible algebraic multigrid implementation. *Lobachevskii Journal of Mathematics* 40, 5 (2019), 535–546.

Zeev Farbman, Raanan Fattal, Dani Lischinski, and Richard Szeliski. 2008. Edge-preserving decompositions for multi-scale tone and detail manipulation. *ACM Trans. Graph.* 27, 3 (2008), 1–10.

Raanan Fattal. 2009. Edge-Avoiding Wavelets and Their Applications. In *ACM SIGGRAPH Proceedings*. Article 22.

Miroslav Fiedler and Vlastimil Pták. 1962. On matrices with non-positive off-diagonal elements and positive principal minors. *Czechoslovak Mathematical Journal* 12, 3 (1962), 382–400.
D Gines, G Beylkin, and J Dunn. 1998. LU factorization of non-standard forms and direct multiresolution solvers. *Applied and Computational Harmonic Analysis* 5, 2 (1998), 156–201.
Marcus J. Grote and Thomas Huckle. 1997. Parallel preconditioning with sparse approximate inverses. *SIAM J. Sci. Comput.* 18, 3 (1997), 838–853.
Joseph Guinness. 2018. Permutation and grouping methods for sharpening Gaussian process approximations. *Technometrics* 60, 4 (2018), 415–429.
Philipp Herholz and Marc Alexa. 2018. Factor once: reusing cholesky factorizations on sub-meshes. *ACM Trans. Graph.* 37, 6 (2018), 1–9.
Philipp Herholz, Timothy A Davis, and Marc Alexa. 2017. Localized solutions of sparse linear systems for geometry processing. *ACM Trans. Graph.* 36, 6 (2017), 1–8.
Philipp Herholz and Olga Sorkine-Hornung. 2020. Sparse Cholesky Updates for Interactive Mesh Parameterization. *ACM Trans. Graph.* 39, 6, Article 202 (2020).
Lily Kharevych, Patrick Mullen, Houman Owhadi, and Mathieu Desbrun. 2009. Numerical coarsening of inhomogeneous elastic materials. *ACM Trans. Graph.* 28, 3, Article 51 (2009).
Ralf Kornhuber, Daniel Peterseim, and Harry Yserentant. 2018. An analysis of a class of variational multiscale methods based on subspace decomposition. *Math. Comp.* 87, 314 (2018), 2765–2774.
Dilip Krishnan, Raanan Fattal, and Richard Szeliski. 2013. Efficient preconditioning of Laplacian matrices for computer graphics. *ACM Trans. Graph.* 32, 4 (2013), 142.
Ligang Liu, Chunyang Ye, Ruiqi Ni, and Xiao-Ming Fu. 2018. Progressive Parameterizations. *ACM Trans. Graph.* 37, 4, Article 41 (2018).
Axel Målqvist and Daniel Peterseim. 2014. Localization of elliptic multiscale problems. *Math. Comp.* 83, 290 (2014), 2583–2603.
J. Andvandervorst Meijerink and Henk A. van der Vorst. 1977. An iterative solution method for linear systems of which the coefficient matrix is a symmetric M-matrix. *Mathematics of computation* 31, 137 (1977), 148–162.
Maxim Naumov. 2012. *Parallel Incomplete-LU and Cholesky Factorization in the Preconditioned Iterative Methods on the GPU.* Technical Report. NVIDIA.
Matthieu Nesme, Paul G Kry, Lenka Jeřábková, and François Faure. 2009. Preserving topology and elasticity for embedded deformable models. *ACM Trans. Graph.* 28, 3, Article 52 (2009).

Input: A point set $V = \{x_i\}_{i=0}^{n-1}$ and a starting point x_{i_0}
Output: Coarse-to-fine maximin order $\mathcal{P} = [i_k]_{k=0}^{n-1}$

$\mathcal{P} \leftarrow [i_0]$;
$d[i_0] = 0$;
valid$[i_0] \leftarrow$ `False`;
h$\leftarrow \emptyset$; `// Initialize a max-heap`
for $x_{i_k} \in V$ **and** valid$[i_k]$ **do**
  h.push$(i_k, \texttt{dist}(x_{i_0}, x_{i_k}))$;
end
while not h.empty() **do**
  $i_p \leftarrow$ h.pop();
  append$(\mathcal{P}, i_p)$;
  $d[i_p] \leftarrow 0$;
  valid$[i_p] \leftarrow$ `False`;
  for $i_k \in$ rangeSearch(x_{i_p}, ℓ_{i_p}) **and** valid(i_k) **do**
    if $\texttt{dist}(x_{i_k}, x_{i_p}) < d[i_k]$ **then**
      $d[i_k] \leftarrow \texttt{dist}(x_{i_k}, x_{i_p})$;
      h.decrease$(i_k, d[i_k])$;
    end
  end
end
return $\mathcal{P}$

Algorithm 2: Approximate farthest-first sampling for coarse-to-fine reordering. A k-d tree is used to perform range searches to prune out candidate points efficiently.

Houman Owhadi. 2017. Multigrid with rough coefficients and multiresolution operator decomposition from hierarchical information games. *SIAM Rev.* 59, 1 (2017), 99–149.
Houman Owhadi and Clint Scovel. 2019. *Operator-Adapted Wavelets, Fast Solvers, and Numerical Homogenization: From a Game Theoretic Approach to Numerical Approximation and Algorithm Design.* Vol. 35. Cambridge University Press.
Sylvain Paris, Samuel W. Hasinoff, and Jan Kautz. 2015. Local Laplacian Filters: Edge-Aware Image Processing with a Laplacian Pyramid. *Commun. ACM* 58, 3 (2015), 81–91.
Pietro Perona and Jitendra Malik. 1990. Scale-space and edge detection using anisotropic diffusion. *IEEE Trans. Pattern Anal. Mach. Intell.* 12, 7 (1990), 629–639.
Florian Schäfer, Matthias Katzfuss, and Houman Owhadi. 2020. Sparse Cholesky factorization by Kullback-Leibler minimization. *arXiv preprint arXiv:2004.14455* (2020).
Florian Schäfer, T. J. Sullivan, and Houman Owhadi. 2021. Compression, inversion, and approximate PCA of dense kernel matrices at near-linear computational complexity. *Multiscale Modeling & Simulation* 19, 2 (2021), 688–730.
Johann Schröder, Ulrich Trottenberg, and Kristian Witsch. 1978. On fast Poisson solvers and applications. In *Numerical Treatment of Differential Equations.* Springer, 153–187.
Jennifer Scott and Miroslav Tůma. 2014a. HSL_MI28: An efficient and robust limited-memory incomplete Cholesky factorization code. *ACM Trans. Math. Software* 40, 4 (2014), 1–19.
Jennifer Scott and Miroslav Tůma. 2014b. On positive semidefinite modification schemes for incomplete Cholesky factorization. *SIAM J. Sci. Comput.* 36, 2 (2014), A609–A633.
Jennifer Scott and Miroslav Tůma. 2014c. On signed incomplete Cholesky factorization preconditioners for saddle-point systems. *SIAM J. Sci. Comput.* 36, 6 (2014), A2984–A3010.
Klaus Stüben. 2000. Algebraic Multigrid (AMG): an Introduction with Applications. *Multigrid* (2000).
Raghunathan Sudarshan. 2005. *Operator-adapted Finite Element Wavelets: theory and applications to a posteriori error estimation and adaptive computational modeling.* Ph.D. Dissertation. MIT, Department of Civil and Environmental Engineering.
Rasmus Tamstorf, Toby Jones, and Stephen F. McCormick. 2015. Smoothed Aggregation Multigrid for Cloth Simulation. *ACM Trans. Graph.* 34, 6, Article 245 (2015).
Joseph Teran, Eftychios Sifakis, Geoffrey Irving, and Ronald Fedkiw. 2005. Robust Quasistatic Finite Elements and Flesh Simulation. *Symp. on Comp. Anim.* (2005), 181–190.
Rosell Torres, Jose M. Espadero, Felipe A. Calvo, and Miguel A. Otaduy. 2014. Interactive Deformation of Heterogeneous Volume Data. *Lecture Notes in Computer Science* 8789 (2014).
Trilinos. 2020. *The Trilinos Project Website.* https://trilinos.github.io
Petr Vaněk, Jan Mandel, and Marian Brezina. 1996. Algebraic multigrid by smoothed aggregation for second and fourth order elliptic problems. *Computing* 56, 3 (1996), 179–196.
Irad Yavneh. 2006. Why Multigrid Methods Are So Efficient. *Computing in Science Engineering* 8, 6 (2006), 12–22.
Yongning Zhu, Eftychios Sifakis, Joseph Teran, and Achi Brandt. 2010. An efficient multigrid method for the simulation of high-resolution elastic solids. *ACM Trans. Graph.* 29, 2 (2010), 1–18.