Parallel accelerated cyclic reduction preconditioner for three-dimensional elliptic PDEs with variable coefficients

Gustavo Chávez[a,*], George Turkiyyah[b], Stefano Zampini[a], David Keyes[a]

[a]*King Abdullah University of Science and Technology, Thuwal, Saudi Arabia*
[b]*American University of Beirut, Beirut, Lebanon*

Abstract

We present a robust and scalable preconditioner for the solution of large-scale linear systems that arise from the discretization of elliptic PDEs amenable to rank compression. The preconditioner is based on hierarchical low-rank approximations and the cyclic reduction method. The setup and application phases of the preconditioner achieve log-linear complexity in memory footprint and number of operations, and numerical experiments exhibit good weak and strong scalability at large processor counts in a distributed memory environment. Numerical experiments with linear systems that feature symmetry and nonsymmetry, definiteness and indefiniteness, constant and variable coefficients demonstrate the preconditioner applicability and robustness. Furthermore, it is possible to control the number of iterations via the accuracy threshold of the hierarchical matrix approximations and their arithmetic operations, and the tuning of the admissibility condition parameter. Together, these parameters allow for optimization of the memory requirements and performance of the preconditioner.

Keywords: Preconditioning, Cyclic reduction, Hierarchical matrices.

1. Introduction

This work focuses on the iterative solution of large-scale block tridiagonal linear systems of equations that arise from the discretization of elliptic partial differential equations on structured grids. Specifically, we demonstrate a parallel and scalable preconditioner based on an approximate factorization generated by the cyclic reduction algorithm [1]. Cyclic reduction uses a sequence of Schur complement reduction steps, with each step eliminating half of the unknowns. While an exact cyclic reduction would result in prohibitively expensive dense matrix blocks, we exploit the data-sparsity of these resulting blocks by approximating them in a hierarchically low-rank form featuring log-linear storage.This work builds on [2, 3], where a fast direct solver was introduced based on the synergy of parallel cyclic reduction and hierarchical matrices, and named accelerated cyclic reduction (ACR).

Iterative methods are advantageous for large-scale scientific computing since they feature tractable complexity and scalability, but their convergence is problem dependent. Direct methods, in contrast, guarantee a solution at the expense of higher complexity. Similar to the way in which incomplete factorizations such as the incomplete Cholesky factorization [4] or the incomplete LU factorization [5] accelerate the convergence, of Krylov methods, we propose a variable-accuracy ACR factorization that serves as a preconditioner to Krylov methods.

Since ACR is entirely algebraic, its range of applicability extends to problems with arbitrary coefficient structure, up to the amenability of rank compression. Furthermore, the ACR factorization and solve stages require only log-linear work and memory, which is particularly beneficial at large-scale. In addition, ACR exhibits substantial concurrency due to two separate characteristics: hierarchical matrix arithmetic operations expose substantial concurrency at the node level [6, 7], and the amount of distributed memory concurrency in cyclic reduction, which is based on red/black ordering of the block rows of the linear system, is proportional to the square root of the problem size in two-dimensions and to the cube root of the problem size in three-dimensions.

Numerical experiments document the robustness, performance, and memory consumption of the ACR preconditioner on a set of elliptic PDEs with heterogeneous coefficients. In particular, we study the variable-coefficient Poisson equation, the convection-diffusion equation, and the wave Helmholtz equation in heterogeneous media. For these equations, the numerical results show that ACR is a broadly applicable preconditioner that, without any equation-specific customizations. can be used to accelerate the convergence of Krylov methods very effectively even

---

*Corresponding author
Email address: `gustavo.chavezchavez@kaust.edu.sa` (Gustavo Chávez)

as the size of the linear system increases, the contract in the coefficients gets more pronounced, or the eigenvalue structure of the matris becomes more irregular.

This rest of this paper is organized as follows. Section 2 reviews the literature on hierarchical matrices for the solution of elliptic PDEs. Section 3 reviews the basic elements of hierarchical matrix representations. Section 4 describes the accelerated cyclic reduction algorithm for the generation of the preconditioner, where hierarchical matrix representations are used for storing and manipulating the formally dense blocks that arise in the factorization process and shows the effect of tuning parameters on the hierarchical matrix structure. Section 5 describes the parallel version of the algorithm including the distributed memory parallelism of the overall factorization process and the shared memory parallelism of the inner arithmetic operations on hierarchical matrices, and shows the weak and strong scalability of resulting algorithm. Sections 6 through 8 present detailed performance results on the effectiveness and near-optimal complexity of the ACR preconditioner on three problem categories of engineering interest: a variable coefficient Poisson problem, a non-symmetric convection-diffusion problem, and an indefinite Helmholtz equation. Section 9 presents the key conclusions.

2. Literature review

The last two decades have witnessed an increasing interest in the use of data-sparse approximations for the solution of linear systems. Leveraging an underlying hierarchically low-rank structure has been a successful strategy for improving the arithmetic complexity and memory footprint of direct solvers. As a result, direct solvers—as well as the closely related preconditioners obtained by aggressively truncating the rank of low-rank blocks—are becoming feasible candidates for tackling large-scale problems that traditional direct solvers are not able to handle due to memory requirements. In this section, we briefly discuss the two major directions towards fast direct solvers and preconditioners for the solution of sparse linear systems.

2.1. Compression of dense frontal matrices

The seminal work of Chandrasekaran et al. [8] showed that the off-diagonal blocks of the Schur complement of discretized elliptic PDEs can be efficiently represented with a hierarchical low-rank approximation. Using this property, methods such as the multifrontal solver [9], and other variants based on Schur complementation, can represent and perform arithmetic operations—of otherwise dense frontal matrices—using data-sparse formats.

An instance of the synergy of the multifrontal method with the hierarchical semiseparable (HSS) format [10] can be found in [11, 12, 13, 14, 15, 16]. However, other formats can be used to accelerate the multifrontal method, such as the hierarchical off-diagonal low rank (HODLR) format [17] which lead to the multifrontal-HOLDR solver [18], or the block low-rank (BLR) format [19] which lead to the multifrontal-BLR solver [20], among others. For further discussion of the differences of each variant, we refer the reader to [21].

2.2. Compression of the entire triangular factors

An alternative technique that, rather than compressing individual blocks within the decomposition process, focuses on approximating the entire triangular factors as one hierarchical matrix has also been proposed. Instances of such strategy can be seen in the work of what is known as $\mathcal{H}$-Cholesky by Ibragimov et al. [22] and $\mathcal{H}$-LU by Grasedyck et al. [23, 24]. The main idea is to create an $\mathcal{H}$-Matrix approximation of the sparse system with a clustering based on a nested dissection ordering of the unknowns. The nonzero blocks are approximated with a low-rank approximation, and an LU factorization is performed under the appropriate $\mathcal{H}$ arithmetic operations.

As with the previous section, different hierarchical formats can be used to approximate dense blocks. The work of Xia et al. [25] also proposes the construction of a rank-structured Cholesky factorization via the HSS hierarchical format, whereas the work of Pouransari et al. [26] approximates fill-in via low-rank approximations with the $\mathcal{H}^2$ format. We refer the reader to [21] for a discussion of the differences of each strategy and a discussion of the implications of the choice of different hierarchical format regarding arithmetic operations count and memory requirements.

3. Hierarchical matrix representations

A hierarchical matrix is a data-sparse representation that enables fast linear algebraic operations by using a hierarchy of off-diagonal blocks, each represented by a low-rank approximation or a small dense matrix, that can be tuned to guarantee a desired precision. The approximation, sometimes referred to as compression, is performed via singular value decomposition, or related methods that deliver low-rank approximations with fewer arithmetic operations than the traditional SVD method. For the representation to be effective in terms of arithmetic operations

and memory requirements, the numerical rank must be significantly smaller than the sizes of the various matrix blocks that they replace.

3.1. Overview of the $\mathcal{H}$-matrix format

Formally, a hierarchical matrix in the $\mathcal{H}$-format [27, 28, 29], can be constructed from four components: an index set, a cluster tree, a block cluster tree, and the specification of an admissibility condition.

3.1.1. Index set

The index set $\mathcal{I} = \{0, 1, \ldots, n-1\}$ represents the the nodal points of the grid under a certain ordering, such as the natural ordering.

3.1.2. Cluster tree

The cluster tree, denoted by $\mathcal{T}_{\mathcal{I}}$, recursively subdivides the index set $\mathcal{I} \times \mathcal{I}$ until exhaustion. For simplicity, consider a binary cluster tree of cardinality 8 as shown in Figure 1.
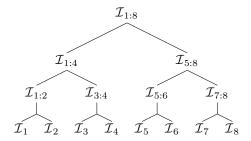

Figure 1: Binary cluster tree $\mathcal{T}_{\mathcal{I}}$ of cardinality 8.

3.1.3. Block cluster tree

Once the cluster tree is defined, the block cluster tree maps matrix sub-blocks over the partitioning of the index set $\mathcal{I} \times \mathcal{I}$. An example of a clustering, frequently used by other data-sparse formats as we discuss in the next section, is a flat block-subdivision of the matrix in l levels, as depicted in Figure 2. The $\mathcal{H}$-format, however, uses a discriminant to determine which blocks are further subdivided with the so-called admissibility condition.
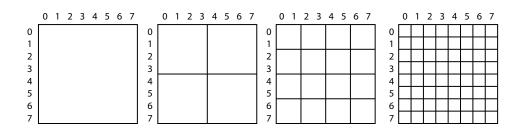

Figure 2: Flat block partitioning at different levels l, without admissibility condition.

3.1.4. Admissibility condition

Besides determining which blocks are further partitioned, the admissibility condition also determines which blocks are represented as a low-rank block (green) or a dense block (red), see Figure 3.
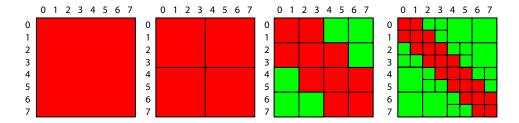
Figure 3: Hierarchical block partitioning with standard admissibility condition. Green blocks are represented by low-rank approximations and red block with dense matrices.

A *weak* admissibility criterion results in a coarse partitioning of the $\mathcal{H}$-matrix format where the off-diagonal blocks at every level of the hierarchy are all represented as low rank blocks. A *standard* admissibility criterion allows a more refined blocking of the matrix given by the inequality:

$$min(diameter(\tau), diameter(\sigma)) \leq \eta \cdot distance(\tau, \sigma) \tag{1}$$

where τ and σ denote two geometric regions defined as the convex hulls of two separate point sets t and s (nodes in the cluster tree). A matrix block A_{ts} satisfying the previous inequality is represented in a low-rank form. The tuning parameter η controls the weight of the distance function. Larger η values admit larger blocks in the off-diagonal regions of the matrix as we will illustrate in Section 4.

3.1.5. Compression of low-rank blocks

The last step for the construction of an $\mathcal{H}$-matrix is the choice of an algorithm to compute low-rank approximations for each of the blocks tagged as low-rank blocks as the product of two matrices of the from UV^T. Given a block of size $n \times n$, an effective compression leads to a tall and narrow matrix U of size $n \times k$, and a short and wide matrix V of size $k \times n$, where k is the numerical rank of the block at some truncated accuracy $\mathcal{H}_\epsilon$. An effective compression means that the numerical rank k is $k \ll n$. An efficient use of a hierarchical matrix to compress a given matrix has a balance between the numerical low-rank k and a moderate number of low-rank blocks.

3.2. Benefits of $\mathcal{H}$-matrix approximations

$\mathcal{H}$-matrix approximations are especially useful for the particular class of matrices that arise from the discretization of elliptic operators with methods such as the boundary element method (BEM), finite-difference (FD), finite volumes (FV), or the finite element method (FEM). The resulting matrices and their Schur complements have blocks with bounded ranks that provide algorithmic gains while using $\mathcal{H}$-matrix storage and the set of algebraic operations that are available within the $\mathcal{H}$-format. In terms of storage, storing a dense matrix requires $\mathcal{O}(N^2)$ memory footprint, while its $\mathcal{H}$-matrix approximation counterpart can be stored in $O(N \log N)$ units of memory. For a comprehensive discussion of the construction of $\mathcal{H}$-matrices and their arithmetic operations, we refer the reader to [29].

4. Accelerated cyclic reduction

In this section we briefly review the cyclic reduction algorithm and describe the tunable accuracy accelerated cyclic reduction variant that improves its arithmetic and memory complexity estimates to near-optimal complexity for the variable-coefficient case.

4.1. Cyclic reduction

Cyclic reduction was introduced by Hockney in 1965 [1], and then formalized by Buzbee and Golub in 1970 [30]. Cyclic Reduction is a recursive algorithm for (block) tridiagonal linear systems. The algorithm consists of two phases: elimination and back-substitution. Elimination is equivalent to block Gaussian elimination without pivoting on a permuted system $(PAP^T)(Pu) = Pf$. The permutation matrix P corresponds to a red-black ordering.

The red-black ordering *slices* the domain into lines or planes, depending on whether the underlying problem comes from a 2D or 3D problem respectively, as depicted on Figure 4. This decomposition bears a similarity to the slice decomposition as reported in [31]. This decomposition is also used in the sweeping preconditioner [32, 33].
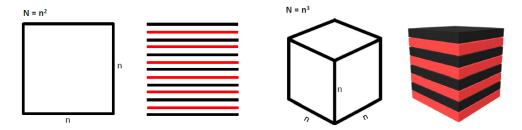
Figure 4: Plain grid in 2D and 3D, and its corresponding red/black coloring. In 2D $N = n^2$, each block row represents a line of size $n \times n$., whereas in 3D $N = n^3$, each block row represents a plane of size $n^2 \times n^2$.

Permutation decouples the system, and the computation of the Schur complement successively reduces the problem size by half. This process is recursive, and it finishes when a single block is reached, although the recursion can be stopped early if the system is small enough to be solved directly. The second phase performs a forward and backward substitution to find the solution. A graphical representation of the progression of elimination is shown in Figure 5. We refer to the reader to [2] for an extended description of the cyclic reduction method.
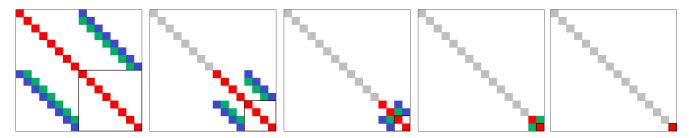

Figure 5: Cyclic reduction preserves a block tridiagonal structure through elimination. Red blocks depict diagonal blocks, green blocks depict the innermost of the bidiagonal blocks, and blue blocks depict the outermost of the bidiagonal blocks. Gray blocks denote blocks in which elimination is completed.

4.2. Accelerated cyclic reduction (ACR)

In [3], we proposed the use of hierarchical matrices and their corresponding algebraic operations to improve on the computational complexity and the memory requirements of the classical cyclic reduction method, and named the resulting method accelerated cyclic reduction (ACR).

In generating the structure of the hierarchical matrix representations of the matrix blocks, we exploit the fact that, for a 3D problem, the domain is subdivided into n planes each consisting of n^2 grid points. As a result, block rows of the matrix are identified with the planes of the discretization grid. We consider this geometry and use a two-dimensional planar bisection clustering when constructing each $\mathcal{H}$-matrix. In other words, ACR deals with $\mathcal{H}$-matrices with one dimension less than the original problem.

A standard admissibility condition was chosen, as opposed to a weak admissibility condition that the $\mathcal{H}$-matrix format also allows, because it provides the flexibility of selecting a range of coarser to finer blocks.

4.3. Tuning parameters

There are three tuning parameters in the construction of an $\mathcal{H}$-matrix that can be leveraged to optimize memory requirements and performance: $\mathcal{H}_\epsilon$, η, and n_{min}. These, in turn, allow for a tunable accuracy ACR factorization which we use in this work as a preconditioner to Krylov methods. The cyclic reduction method was originally conceived as a direct solver; however, extensions of the use of CR as preconditioner have appeared in the literature [34, 35], although to the best of our knowledge, none of them use hierarchical matrices.

The first parameter $\mathcal{H}_\epsilon$ controls the specified block-wise relative accuracy of the $\mathcal{H}$-matrix blocks tagged as low-rank. This parameter resembles the cut-off tolerance ϵ of the truncated SVD that disregards singular values to achieve an approximation accuracy of ϵ.

The second parameter is η, from the admissibility condition criterion. The case for choosing a standard admissibility condition (small η) is that, by further refining off-diagonals blocks, it is possible to achieve the same relative accuracy as with a weak admissibility (large η) but with smaller numerical ranks, albeit with more off-diagonal

5

blocks (see Figure 6a vs. 6d). Numerical low-ranks are crucial to ensure economic memory consumption and overall high-performance.

Consider the computation of the approximate inverse in the $\mathcal{H}$-matrix format of a 2D variable-coefficient Poisson problem, with an error tolerance of three digits of accuracy in the Frobenius norm ($\left\|AA^{-1}-I\right\|_F$), and a fixed accuracy parameter $\mathcal{H}_\epsilon$. The variable of interest in this experiment is the admissibility condition parameter η, which controls the block refinement as depicted in Figure 6.

(a) $\eta = 2$ (Strong admissibility)   (b) $\eta = 64$

(c) $\eta = 128$ (d) $\eta = 256$ (Weak admissibility)

Figure 6: $\mathcal{H}$-matrix structure for different parameter η with fixed $\mathcal{H}_\epsilon$ and leaf size n_{min}=32. Matrix depicts a 2D variable coefficient Poisson problem with four orders of magnitude of contrast in the coefficient discretized with $N = 128^2$ degrees of freedom. The numbers inside the green low-rank blocks denote the required numerical rank for the specified accuracy.

Table 1 documents the memory requirements of each approximate inverse as a function of η. As shown, the optimal η parameter resides in between strong admissibility (η=2) and weak admissibility (η=256). This tuning is a significant advantage for data-sparse formats that are not limited to the choice of weak admissibility, such as the $\mathcal{H}$-format. As the table shows, the most economic inverse regarding memory is not necessarily the representation with the smallest rank, since an aggressive refinement leads to a larger number of blocks and deeper cluster trees.

N	η	$\left\|AA^{-1}-I\right\|_F$	Max. Rank	Memory (Bytes)
128^2	2	5.0e-3	16	6.76e+7
128^2	**64**	**7.2e-3**	**34**	**6.64e+7**
128^2	128	9.1e-3	64	8.64e+7
128^2	256	9.1e-3	126	1.01e+8

Table 1: Memory consumption as a function of the tuning parameter η for the computation of the approximate inverse in the $\mathcal{H}$-matrix format of a 2D variable-coefficient Poisson problem with four orders of magnitude of contrast in the coefficient, discretized with $N = 128^2$ degrees of freedom. Parameter η=2 depicts strong admissibility, while η=256 depicts weak admissibility; regarding memory requirements, η=64 is optimal.

Since the memory consumption of ACR is determined by the sum of the memory consumption of each $\mathcal{H}$-matrix involved in elimination, an economical storage of each $\mathcal{H}$-matrix directly translates into savings to the overall ACR memory footprint. As shown in Figure 7, tuning η across a range of problem sizes has nuanced benefits. For linear systems in the order of a few millions of degrees of freedom a coarse block partitioning (close to weak admissibility) minimizes the overall memory consumption. However, for problems larger than a dozen of millions of unknowns

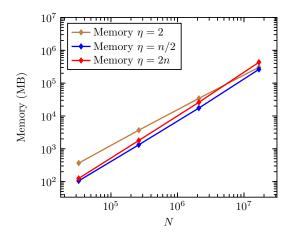block partitioning closer to strong admissibility is optimal to reduce memory requirements.

Figure 7: Effect of tunable parameter η on total memory consumption of ACR for a 3D variable-coefficient problem. The memory complexity estimate of $\mathcal{O}(N \log N)$ is achieved for $\eta = 2$ which corresponds to strong admissibility. $\eta = 2n$ which correspond to weak admissibility uses the most memory asymptotically, and an intermediate value of $\eta = n/2$ achieves the least amount of memory within the range of problem sizes considered.

The third tuning parameter determines which blocks with less than or equal to n_{min} rows or columns are stored as dense matrices, as it is more efficient to operate on them in dense rather than in low-rank form. It also alleviates unnecessarily deep binary trees in the structure of $\mathcal{H}$-matrices. Figure 6 depicts these blocks in red.

5. Hybrid distributed-shared parallelism

The concurrency features of cyclic reduction have been evaluated in both distributed memory [36, 37, 38, 39, 40], and shared memory environments [41, 42, 43], although to the best of our knowledge, none of them use hierarchical matrices. We propose a hybrid model with MPI across the nodes and task-based parallelism across the cores in a node.

The distribution of parallel work was designed to accommodate the architecture of a modern supercomputer, such as the Shaheen Cray XC40 supercomputer at the King Abdullah University of Science & Technology. Shaheen is composed of 6,144 compute nodes, with each node holding 128GB of RAM and two Intel Haswell processors with 16 cores clocked at 2.3Ghz. The nodes are connected with a Dragonfly network. All our reported numerical experiments were performed on this machine.

Since the supercomputer architecture features multiple fast individual nodes, physically connected trough a high-speed network interconnect, we seek to maximize computation within nodes and minimize communication across nodes. Schur complementation is calculated locally within the nodes with a task-based parallel programming model. Dependencies to perform elimination and solve are fulfilled via a distributed memory programming model in which only the missing matrix blocks or vectors are provided with the message passing interface (MPI).

5.1. Distributed memory parallelism

Each plane of the computational domain is assigned to an MPI rank, but to minimize communication within the nodes, we allocate as many planes per node as memory allows. Let p be the number of compute nodes and n be the number of planes; therefore each node stores n/p planes at the beginning of the factorization. Since ACR eliminates half of the planes at each step, after r steps each node holds $n/(2^r p)$ planes. At level $r = \log(n/p)$, every node holds a single plane only. In the multigrid literature, this level is known as the C-level, i.e. the coarse level, illustrated in blue in Figure 8. The remaining $\log p$ steps beyond the C-level leave some compute nodes idle; fortunately, most of the remaining block operations have been completed by this step.

Distributed memory communication occurs only at inter-node boundaries, as the elimination of plane i only requires planes $i - 1$ and $i + 1$. Thus up to the C-level, there are $O(p)$ messages per step, each transmitting planes of size $O(k\, n^2 \log n)$. Beyond the C-level, there are $O(p/2 + \cdots + 1) \approx O(p)$ communication messages, adding up to a total communication volume of $O(k\, p\, n^2 \log n\, (\log \frac{n}{p} + 1))$, where n is the size of the linear dimension and k is the rank of the low-rank approximation.
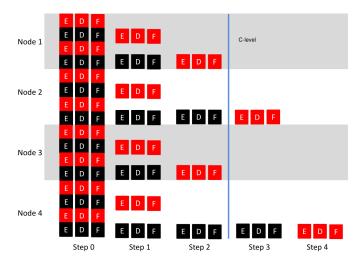
7

Figure 8: Distribution of multiple planes per node for an example with $n = 16$ planes and $p = 4$ nodes.

5.2. Shared memory parallelism

We exploit the concurrency of hierarchical matrix algebra at node level [6], in particular, through a task-based programming model [7]. The HLibPro package relies on the Intel Threading Building Blocks library [44] to build directed acyclic graphs for the dependencies between tasks, which might involve recursion. The allocation of cores to perform hierarchical matrix algebra within the node depends on the number of planes per node.

For instance, for a node with thirty-two processors and four planes per node, we set four MPI processes, and for each plane we allocate eight processors to perform task-based parallelism. Resource allocation for either block row processing (communication of planes) or parallel task-based hierarchical arithmetics (computation of Schur complement) can be tuned to maximize performance or memory availability.

We refer to the reader to [3] for an extended discussion of the parallel features of ACR, and the derivation of its parallel complexity estimates.
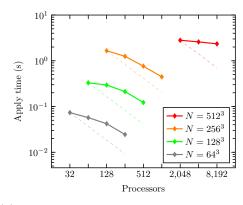
5.3. Parallel scalability

The parallel scalability of the ACR preconditioner is demonstrated below on a constant-coefficient Poisson equation with homogeneous Dirichlet boundary conditions in the unit cube, discretized with the 7-point finite-difference star stencil. We use the conjugate gradient method [45] with a convergence criterion in the 2-norm of the relative residual down to 10^{-8}. This problem results in a symmetric positive definite matrix whose factors exhibit rapid decay of the singular values of off-diagonal blocks and offers an ideal testbed to show the parallel scalability of the hierarchically low-rank algorithmic computations. The following experiments improve on previous timings for the setup phase performed at smaller accuracies [3], demonstrating that the use of ACR as a preconditioner with looser tolerances, i.e. smaller ranks, also benefits scalability.

5.3.1. Strong scaling

Figures 9a and 9b show the total time in seconds for the setup and solve per iteration of the ACR preconditioner in a strong scaling setting; dashed lines indicate ideal scaling, a reduction in time by a factor of two as we double the number of processors.
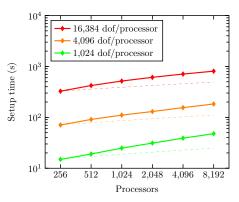
8

(a) Strong scaling of the preconditioner setup.   (b) Strong scaling of the preconditioner apply.
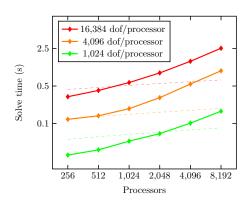
Figure 9: ACR preconditioner strong scalability for the solution of the constant-coefficient Poisson equation.

The most time-consuming phase of the ACR preconditioner benefits the most as the number of processors increases for a variety of problem sizes. Nonetheless, the ideal scaling of the solve stage deteriorates at large processor counts as factors such as hardware latency play a significant role in this computationally lightweight kernel solely based on $\mathcal{H}$ matrix-vector multiplications.

5.3.2. Weak scaling

Figures 10a and 10b depict the results of a weak scaling experiments for the ACR preconditioner fixing a different numbers of degrees of freedom per processor, along with the ideal weak scaling reference lines depicted as dashed curves considering that the estimates of setup is of $O(k^2 N \log^2 N)$ operations and the solve stage per iteration are of $O(kN \log N)$ operations.

(a) Weak scaling of the preconditioner setup.   (b) Weak scaling of the preconditioner apply.

Figure 10: Weak scalability of the ACR preconditioner application for the solution of the constant-coefficient Poisson equation.

The setup stage follows the ideal trending line as we increase the number of processors. The solve phase deviates from the ideal scaling due to the communication latency which is more noticeable due to its lower arithmetic intensity, the scalability of the Krylov method (conjugate gradient in this case), and the load imbalance in the late stages of the recursive bisection of cyclic reduction.

6. Variable-coefficient Poisson equation

The solution of variable-coefficient PDEs is an essential engineering problem, as the coefficient structure typically corresponds to material properties of the problem under consideration. This section documents the behavior of the ACR preconditioner from an increasingly challenging coefficient structure with up to six orders of magnitude of contrast.

9

The problem under consideration in this section is the symmetric positive definite discretization of the 3D variable-coefficient Poisson equation with Dirichlet boundary conditions. In particular, the second-order accurate 7-point finite-difference star stencil with harmonic average of the coefficient $\kappa(x)$ [46]:

$$-\nabla \cdot \kappa(x)\nabla u = 1, \quad \mathbf{x} \in \Omega = [0,1]^3, \quad u(\mathbf{x}) = 0, \mathbf{x} \in \Gamma, \tag{2}$$

6.1. Generation of random permeability fields

The generation of random permeability field $\kappa(x)$ that closely represents a porous medium for the modeling of water or oil flow is a well-defined task on its own. The experiments in this section are based on the parallel framework for the multilevel Monte Carlo approach (MLMC) described in [47], via the Distributed and Unified Numerics Environment DUNE [48]. The random permeability fields are defined with covariance function of the form:

$$\mathcal{C}(h) = \sigma^2 \exp(-||h||_2/\lambda), \quad h \in [0,1]^3 \tag{3}$$

Gaussian random fields are set to a correlation length $\lambda = 3h$, where $h = \frac{1}{n-1}$ and $N = n^3$. The variance σ is set to deliver a particular contrast in the coefficient measured in orders of magnitude. Figure 11 depicts four random fields realizations at different number of degrees of freedom and contrast of the coefficient.

(a) $N = 32^3$ One order of magnitude of contrast.   (b) $N = 64^3$ Two orders of magnitude of contrast.   (c) $N = 128^3$ Four orders of magnitude of contrast.   (d) $N = 256^3$ Six orders of magnitude of contrast.
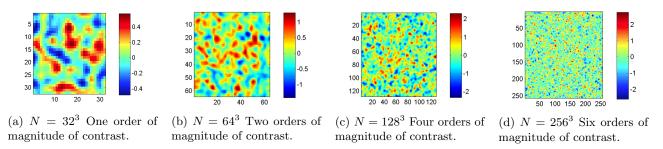
Figure 11: Different realizations of random permeability fields $\kappa(x)$ at different resolutions and contrast of the coefficient. Images depict the middle slice of each 3D permeability field.
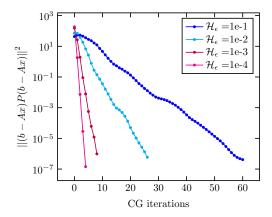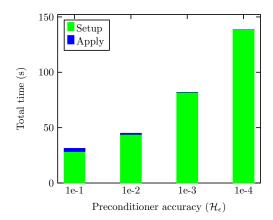
6.2. Tuning parameters

The main parameter that controls the accuracy of the ACR preconditioner is $\mathcal{H}_\epsilon$. As discussed in section 3, $\mathcal{H}_\epsilon$ controls the accuracy of the $\mathcal{H}$-matrix approximations and their arithmetic operations. This global threshold, in turn, controls the relative accuracy of the solution for a given right-hand side.

It is expected that as we adjust $\mathcal{H}_\epsilon$, we can control the required number of iterations to reach convergence with a Krylov method. One could set $\mathcal{H}_\epsilon$ to the sought after accuracy of the solution and not require any iteration at all. However, the performance and memory requirements, although asymptotically-optimal, become impractical at high-accuracy for 3D problems. For the ACR preconditioner, the sweet spot for achieving the fastest time to solution is not the one corresponding to the least number of iterations. It is generally the case that the inexpensive ACR preconditioners provide the fastest time to solution. There is a trade-off between the accuracy of the preconditioner and the number of Krylov iterations as numerical experiments show below.

The effect on the required number of iterations as a function of the preconditioner accuracy $\mathcal{H}_\epsilon$ to solve a $N = 128^3$ problem with coefficient contrast of four orders of magnitude can be seen in Figure 12a. The largest $\mathcal{H}_\epsilon$ requires the most number of iterations, while the smallest $\mathcal{H}_\epsilon$ requires the least number of iterations.

As can be seen from Figure 12b, even though setting a large $\mathcal{H}_\epsilon$ required the greatest number of CG iterations, this is the recommended value of $\mathcal{H}_\epsilon$ to optimize for time to solution in our current implementation. Although there are more iterations than with a smaller $\mathcal{H}_\epsilon$, the application of the preconditioner is fastest at large $\mathcal{H}_\epsilon$ since the ranks are the smallest, see Figure 13a. Figure 13b depicts how $\mathcal{H}_\epsilon$ directly determines the memory footprint of the preconditioner, and shows why it is desirable to set $\mathcal{H}_\epsilon$ as large as possible to also optimize for memory requirements. Larger values of the preconditioner accuracy could deliver better time to solution, although at the expense of more synchronizing iterations, which might be undesirable for some applications.
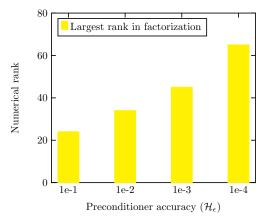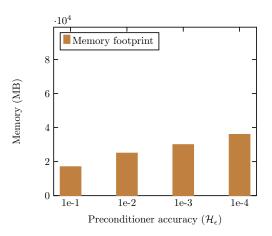
(a) Number of CG iterations as a function of the preconditioner accuracy $\mathcal{H}_\epsilon$ for the variable-coefficient Poisson equation. The preconditioner with the smallest $\mathcal{H}_\epsilon$ requires the least number of iterations.

(b) Time requirements while refining the preconditioner accuracy $\mathcal{H}_\epsilon$ for the variable-coefficient Poisson equation. The preconditioner with the largest $\mathcal{H}_\epsilon$ delivers the best time to solution.

Figure 12: Number of iterations and preconditioning accuracy for the variable-coefficient Poisson equation with $N = 128^3$ degrees of freedom and coefficient contrast of four orders of magnitude.

(a) Largest rank in the factorization at different preconditioner accuracy $\mathcal{H}_\epsilon$. The preconditioner with the largest $\mathcal{H}_\epsilon$ requires the smallest numerical rank.

(b) Memory requirements at different preconditioner accuracy $\mathcal{H}_\epsilon$. The preconditioner with the largest $\mathcal{H}_\epsilon$ requires the least amount of memory.

Figure 13: Effect of the preconditioner accuracy $\mathcal{H}_\epsilon$ for the variable-coefficient Poisson equation with $N = 128^3$ degrees of freedom and coefficient contrast of four orders of magnitude.

6.3. Sensitivity with respect to high contrast coefficient

As the problem difficulty increases, i.e. the contrast of the coefficient sharpens, there are cases for which the most economical preconditioner (e.g. $\mathcal{H}_\epsilon$=1e-1) might not reach convergence within an acceptable number of iterations, see Table 2.

N	$\mathcal{H}_\epsilon$	CG Iterations
32^3	1e-1	27
64^3	1e-1	51
128^3	1e-1	95
256^3	1e-1	100+
	1e-2	73

Table 2: Number of iterations required by CG for the variable-coefficient Poisson equation with coefficient contrast of six orders of magnitude. The most economical preconditioner for the hardest problem did not reach convergence within 100 iterations, thus requiring a more accurate version of the preconditioner to reach convergence.

11

Therefore, to reach convergence, a more accurate preconditioner is necessary. Figure 14 shows the required number of iterations to achieve convergence for a preconditioner with accuracy $\mathcal{H}_\epsilon$=1e-2 at increasing problem size and contrast of the coefficient. For comparison, the baseline case (zero contrast) depicts a constant-coefficient Poisson equation with $\kappa(x)$=1.
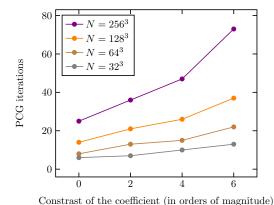

Figure 14: Required number of iterations for an ACR preconditioner accuracy of $\mathcal{H}_\epsilon$=1e-2 as both the problem size and the contrast of the coefficient increases. A larger number of iterations is necessary as the contrast of the coefficient increases.

One of the established methods for the solution of these type of problems is algebraic multigrid. As the contrast of the coefficient changes, Table 3 shows the number of iterations required by the conjugate gradient method (CG) without preconditioning, and the preconditioned conjugate gradient method using algebraic multigrid (AMG) as a preconditioner, accessed here via the hypre library [49, 50]. To compare at a similar number of iterations, the ACR preconditioner was tuned to require less than ten iterations, in this case at $\mathcal{H}_\epsilon$=1e-4. Experiments show that at moderate contrast of the coefficients and using 512 cores, the solve time of the ACR preconditioner is comparable to the AMG preconditioner, albeit at a large enough number of right-hand sides so that the setup time of ACR gets amortized since the preconditioner can be reused. Another dimension of comparison is a traditional direct solver, such as the LU factorization, here accessed through the SuperLU DIST package [51]. For instance, for the problem of four orders of magnitude of contrast, the LU factors are computed in 27.16 seconds and require 3.5E4 MB of memory, whereas the ACR preconditioner at $\mathcal{H}_\epsilon$=1e-1 is computed in 27.71 seconds, but it only requires 1.7E4 MB of memory to store its factors.
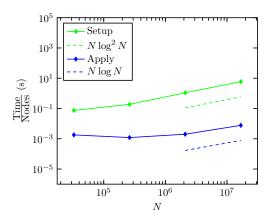
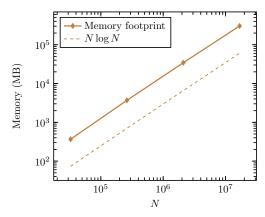| Coefficient	CG + No Prec.		CG + AMG		CG + ACR	
contrast	Iterations	Solve	Iterations	Solve	Iterations	Solve
0	257	0.10	6	0.53	3	0.30
2	975	0.24	6	0.35	4	0.34
4	2210	0.44	6	0.28	4	0.37
6	6968	1.35	7	0.26	7	0.59

Table 3: Number of iterations and solve time for the solution of a sequence of Poisson problems $N = 128^3$ with a variable coefficient at increasing order of magnitude of contrast of the coefficient. Methods under consideration include algebraic multigrid (AMG), and accelerated cyclic reduction (ACR).

6.4. Operation count and memory footprint

The complexity estimates of the number of operations in the setup and application phases of the preconditioner are bounded by $O(k^2 N \log^2 N)$ and $O(kN \log N)$ respectively, while its memory footprint is bounded by $O(kN \log N)$; where N is the number of degrees of freedom and k is the numerical rank of the approximation. To demonstrate that these estimates hold for a variable-coefficient problem, Figure 15 shows the behavior of the preconditioner in terms of operations count and memory footprint as we increase the number of degrees of freedom N for the variable-coefficient Poisson equation with a coefficient of four orders of magnitude of contrast.

The vertical axis of Figure 15a, normalized by the number of processors used in each case, reports the measured performance of the setup and application phases of the preconditioner while comparing it with their theoretical complexity. Figure 15b reports the total memory requirements as the problem size increases and also compares it with the theoretical complexity demonstrating a fair agreement.

(a) Comparison of the preconditioner setup and application with their corresponding theoretical estimates.

(b) Comparison of the preconditioner memory footprint with its theoretical estimate.

Figure 15: Measured performance and memory footprint for the solution of an increasingly larger variable-coefficient Poisson equation with a random field of four orders of magnitude of contrast in the coefficient. The preconditioner accuracy for this experiments is set to $\mathcal{H}_\epsilon = 1\text{e-}1$.

7. Convection-diffusion equation with recirculating flow

In this section we show the effectiveness of the ACR preconditioner on the convection-diffusion equation with a variable and recirculating flow $b(\mathbf{x})$, i.e. a flow with vanishing normal velocities at the boundary.

$$-\nabla \cdot \kappa(x)\nabla u + \alpha b(\mathbf{x}) \cdot \nabla u = f(\mathbf{x}), \quad \mathbf{x} \in \Omega = [0,1]^3,$$
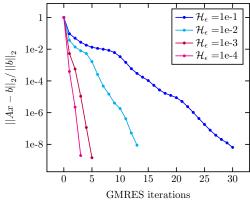
$$b(\mathbf{x}) = \begin{bmatrix} \sin(a\,2\pi x)\sin(a\,2\pi(1/8+y)) + \sin(a\,2\pi(1/8+z))\sin(a\,2\pi x) \\ \cos(a\,2\pi x)\cos(a\,2\pi(1/8+y)) + \cos(a\,2\pi(1/8+y))\cos(a\,2\pi z) \\ \cos(a\,2\pi x)\cos(a\,2\pi(1/8+z)) + \sin(a\,2\pi(1/8+y))\sin(a\,2\pi z) \end{bmatrix}, \quad (4)$$
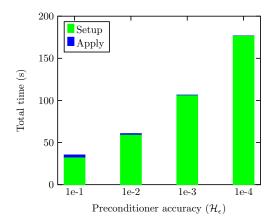
$$b_x + b_y + b_z = 0.$$

The equation is discretized with a 7-point upwind finite difference scheme, which leads to a nonsymmetric linear system. When the convection term dominates, $\alpha > 1$, this equation is known to be challenging for classical iterative solvers.

7.1. Tuning parameters

In a regime of convection dominance, Figure 16a shows how the ACR preconditioner can control the number of GMRES iterations by tuning the preconditioner accuracy $\mathcal{H}_\epsilon$.
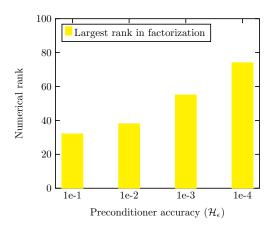

(a) Number of iterations as a function of the preconditioner accuracy $\mathcal{H}_\epsilon$. As $\mathcal{H}_\epsilon$ decreases, the preconditioner requires fewer iterations.
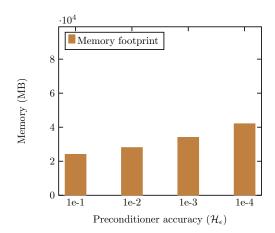
(b) Time requirements while refining the preconditioner accuracy $\mathcal{H}_\epsilon$. The largest $\mathcal{H}_\epsilon$ delivers the best time to solution.

Figure 16: This experiment depicts a convection-diffusion problem with recirculating flow with eight vortices, $\alpha = 8$, discretized with $N = 128^3$ degrees of freedom.

Regarding absolute time to solution, experiments with our latest implementation show that the preconditioner with the largest $\mathcal{H}_\epsilon$ led to the best time to solution, albeit with the most iterations, as Figure 16b shows. As a result, this preconditioner configuration featured the lowest numerical rank, as shown in Figure 17a, enabling a fast application at each iteration. Furthermore, the fastest preconditioner had the least memory requirements, as shown in Figure 17b.

(a) Largest rank in factorization at different $\mathcal{H}_\epsilon$ for the convection-diffusion equation. The preconditioner with the largest $\mathcal{H}_\epsilon$ features the lowest numerical ranks.

(b) Memory requirements while refining the preconditioner accuracy $\mathcal{H}_\epsilon$. The largest $\mathcal{H}_\epsilon$ delivers the preconditioner with the least memory footprint.

Figure 17: Effect on the preconditioner accuracy $\mathcal{H}_\epsilon$ for a convection-diffusion problem with recirculating flow with eight vortices, $\alpha = 8$, and discretized with $N = 128^3$ degrees of freedom.

7.2. Sensitivity with respect to vortex wavenumber

Consider an increasing number of vortices in the flow $b(\mathbf{x})$, as Figure 18 shows. At the corners, and in center of each vortex, there are saddle points which are known to be challenging for multigrid methods to resolve [52]. Figure 19 demonstrates that the ACR preconditioner remains robust as the number of vortices increases.

(a) Two vortices.   (b) Four vortices.   (c) Six vortices.   (d) Eight vortices.
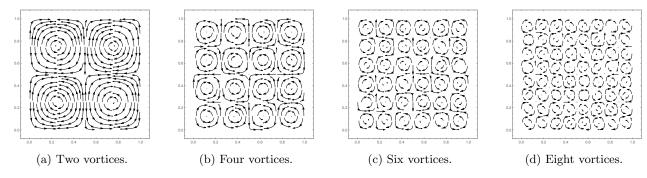
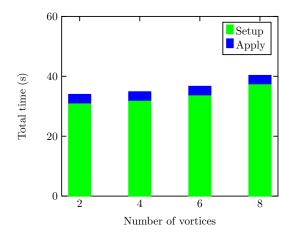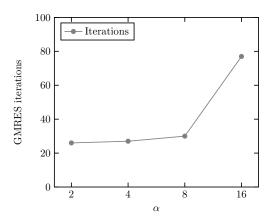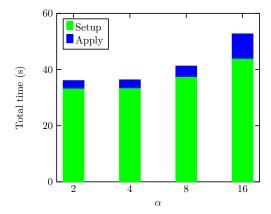Figure 18: Increasing number of vortices in the flow $b(\mathbf{x})$.

Figure 19: Time distribution of the preconditioner as the number of vortices in $b(\mathbf{x})$ increases. Increasing the number of vortices had a minor effect on the effectiveness of the preconditioner.

7.3. Sensitivity with respect to convection dominance

Consider a fixed-accuracy ACR preconditioner with $\mathcal{H}_\epsilon$ =1e-1, and an increasingly convection dominated problem, achieved by gradually increasing α in Equation 4. As expected, Figure 20a shows that a low-accuracy preconditioner requires more iterations as the convective term dominates. Furthermore, given that the accuracy of the preconditioner is fixed, there is a noticeable effect on the application phase of the preconditioner, which is proportional to the number of iterations. A graphical representation of such behavior can be seen in Figure 20b. Evidently, as shown in the section 7.1, it is possible to control, and decrease, the number of iterations by building a more accurate preconditioner. But the key point here is that the ACR preconditioner in combination with GMRES is demonstrated to be robust for convection dominated problems.

As a matter of comparison, Table 4 shows the number of iterations of GMRES without preconditioner, and GMRES in combination with the incomplete LU factorization. In this case, the problem with the strongest convection dominance did not reach a solution with the incomplete LU factorization as a preconditioner of GMRES. In terms of memory, the ACR preconditioner for the problem with the strongest convection dominance required 30.76 seconds and 2.3E4 MB of memory with $\mathcal{H}_\epsilon$=1e-1, whereas SuperLU DIST required 44.59 seconds and 3.7E4 MB of memory.

(a) Number of iterations as the convection term gains dominance. An increase in the dominance of the convection term requires mores iterations.

(b) Time requirements as the convection term gains dominance. The overall time to solution has a moderate increase.

Figure 20: Effect on the preconditioner accuracy $\mathcal{H}_\epsilon$ for the convection-diffusion equation with recirculating flow discretized with $N = 128^3$ degrees of freedom as the convective becomes more significant than the diffusion term.

15

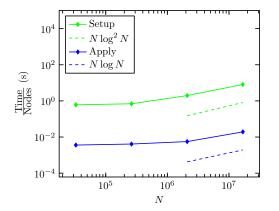| α	GMRES + No Prec.		GMRES + ILU(0)		GMRES + ACR	
	Iterations	Solve	Iterations	Solve	Iterations	Solve
0	1,132	0.38	201	0.20	26	2.93
2	1,242	0.51	226	0.22	27	3.10
4	1,765	0.60	368	0.30	30	4.02
6	100,000+	-	100,000+	-	77	8.94

Table 4: Number of iterations and solve time for the solution of a sequence of convection-diffusion problems with $N = 128^3$ and increasingly convection dominance. Methods under consideration include the incomplete LU factorization (ILU), and accelerated cyclic reduction (ACR).
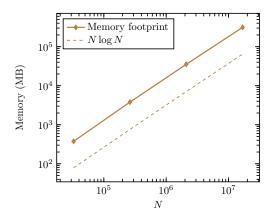
7.4. Operation count and memory footprint

Figure 21 presents a comparison between the measured performance and memory requirements of the preconditioner, and their corresponding theoretical complexity estimates for the convection-diffusion problem described in Equation 4, as the problem size increases.

The vertical axis of Figure 21a, normalized with the number of compute nodes used in each case, reports the measured performance of the setup and application phases of the preconditioner while demonstrating a fair agreement with the asymptotic complexity estimates for the large-scale experiments.

Figure 21b reports the total memory requirements as the problem size increases and also compares it with its corresponding theoretical complexity demonstrating a fair agreement across all experiments.

(a) Comparison of the preconditioner setup and application with their corresponding theoretical estimates.

(b) Comparison of the preconditioner memory footprint with its theoretical estimate.

Figure 21: Measured performance and memory footprint for the solution of the convection-diffusion equation with recirculating flow.

8. Indefinite Helmholtz equation in heterogeneous media

The numerical solution of the indefinite Helmholtz equation offers one of the greatest challenges for iterative and direct solvers at large-scale [53]. There is a significant interest in the development of optimal methods as several engineering applications use the Helmholtz equation to model time-harmonic propagation of acoustic waves. Inversion techniques based on full-waveform inversion (FWI) for instance, involve heterogeneous velocity models and the solution of multiple right-hand sides at a wide range of frequencies. Therefore, the introduction of an efficient forward solver directly contributes to expanding the limits of what can be modeled computationally.

Consider the indefinite Helmholtz equation in a variable velocity field $c(\mathbf{x})$, at frequency f, and Dirichlet boundary conditions in the unit cube:

$$-\nabla^2 u - \frac{(2\pi f)^2}{c(\mathbf{x})^2} u = f(\mathbf{x}), \ \Omega = [0,1]^3, \ \mathbf{x} \in \Gamma,$$

$$c(\mathbf{x}) = 1.25(1 - 0.4e^{-32(|x-0.5|^2 + |y-0.5|^2)})$$

$$u(\mathbf{x}) = \sin(\pi x)\sin(\pi y)\sin(\pi z)$$

(5)

The velocity field models a waveguide over the unit cube as proposed in [33], and depicted in 22. The forcing term $f(\mathbf{x})$ is adjusted to satisfy the proposed exact solution $u(\mathbf{x})$. The equation is discretized with the 27-point

16

trilinear finite element scheme on hexahedra with the software library PetIGA [54]. Since the linear system arising from the discretization is indefinite in the high-frequency Helmholtz case, we use ACR to accelerate the convergence of GMRES.
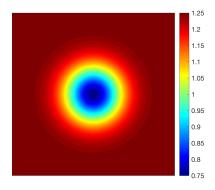

Figure 22: Wave velocity field $c(\mathbf{x})$. The image depicts the middle slice of the 3D wave velocity field.

8.1. Tuning parameters

We illustrate the effectiveness of the ACR preconditioner on a moderately high-frequency Helmholtz problem as described in Equation 5, discretized with $N = 128^3$ degrees of freedom and 12 points per wavelength. As Figure 23 shows, we can control the number of iterations that GMRES requires to reach convergence by adjusting the accuracy of the preconditioner $\mathcal{H}_\epsilon$. Notice that the preconditioner accuracy $\mathcal{H}_\epsilon$ is smaller than what was chosen for diffusive problems. The need of higher relative accuracy is due to the fact that the Helmholtz equation, in the high-frequency regime, has off-diagonal block ranks that asymptotically grow with problem size ($k \sim \mathcal{O}(n)$). This theoretical estimate is reported in the literature [8]. Evidently, rank growth impacts hierarchical-matrix based solvers. Nonetheless, the complexity estimates of the ACR preconditioner are still lower than traditional exact sparse factorizations, as demonstrated in section 8.3.
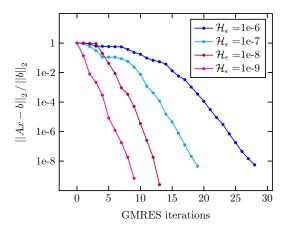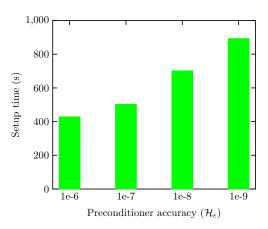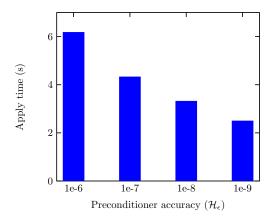

Figure 23: Number of iterations as a function of the preconditioner accuracy $\mathcal{H}_\epsilon$. As $\mathcal{H}_\epsilon$ decreases, the preconditioner requires fewer iterations.

Even though the timings of the preconditioner shows an increase in the setup time as compared to diffusive problems, it still features an economical solve stage (Figure 24b). As mentioned in the introduction of this section, for inverse problems which require the solution of a large number of right-hand sides (typically up to a few thousands), the setup phase (Figure 24a) is typically regarded as an off-line phase that gets amortized if the solve stage is relatively fast, which is the case for the ACR preconditioner.
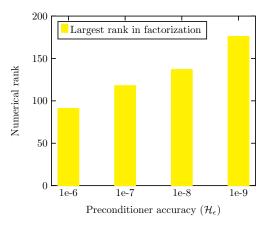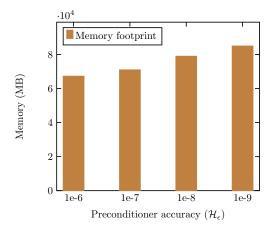
17

(a) Setup phase at increasing accuracy of the preconditioner.

(b) Application phase at increasing accuracy of the preconditioner.

Figure 24: Time requirements while refining the preconditioner accuracy $\mathcal{H}_\epsilon$. The loosest $\mathcal{H}_\epsilon$ delivers the fastest time to solution for a single right-hand side, whereas the tightest $\mathcal{H}_\epsilon$ delivers the best time to solution for a large number of right-hand sides, since the preconditioner setup is computed only once.

The growth in the setup phase as the accuracy of the preconditioner is tightened is due to increased numerical ranks, as shown in Figure 25a. Rank growth has a direct impact on the memory footprint of the preconditioner, as shown in Figure 25b. Once more, the preconditioner with the loosest $\mathcal{H}_\epsilon$, i.e. the lowest numerical rank, is the preconditioner of choice to optimize for both memory and performance.

(a) Largest rank in factorization. Factorizations with smaller ranks lead to more iterations, but less time to solution and memory footprint.

(b) Memory requirements while refining the preconditioner accuracy $\mathcal{H}_\epsilon$. The most economical preconditioner regarding memory footprint is delivered with the largest $\mathcal{H}_\epsilon$.

Figure 25: Effect on the preconditioner accuracy $\mathcal{H}_\epsilon$ for the high-frequency Helmholtz equation in a heterogeneous medium discretized with $N = 128^3$ degrees of freedom and 12 points per wavelength.
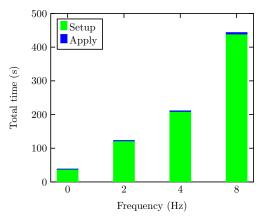
8.2. Low to high frequency Helmholtz regimes

Consider a sequence of Helmholtz problems, as described in Equation 5, at increasing frequency. If the frequency is set to $f = 0$ Hz, the zeroth-order term vanishes, and we are left with a constant-coefficient Poisson problem. At the other end of the spectrum, a frequency of $f = 8$ Hz corresponds to a moderately high-frequency Helmholtz problem at 12 points per wavelength, as the problem featured in the previous section. Table 5 shows the preconditioner accuracy chosen to require a maximum of 20 GMRES iterations to reach convergence.
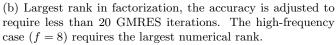
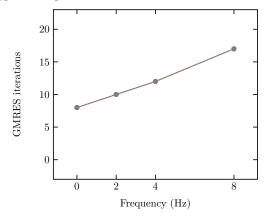| f	Points per wavelength	$\mathcal{H}_\epsilon$
0	-	1e-1
2	48	1e-3
4	24	1e-4
8	12	1e-6

Table 5: Tuning of the preconditioner to require at most 20 GMRES iterations for a sequence of Helmholtz problems at increasing frequencies. The problem with $f = 0$ represents a constant-coefficient Poisson problem, while $f = 8$ represents a moderately high-frequency Helmholtz problem.
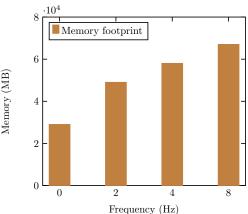
Figure 26a shows an apparent increase in both the setup and application phases of the preconditioner as a function of the frequency f. The growth in the setup is mainly due to the higher numerical ranks required to meet the upper limit of 20 iterations, as shown in Figure 26b. The increase in the application phase is due to both an increase in ranks, and an increase in the indefiniteness of the problem due to a higher frequency; as is evident from the growth in the number of required iterations depicted in Figure 26c. Finally as illustrated in figure 26d, the memory footprint also increases with the frequency as a consequence of higher ranks.

(a) Time requirements as a function of frequency. The high-frequency regime ($f = 8$) requires the most time in both setup and application phases.

(b) Largest rank in factorization, the accuracy is adjusted to require less than 20 GMRES iterations. The high-frequency case ($f = 8$) requires the largest numerical rank.

(c) Number of iterations as a function of frequency. The high-frequency regime ($f = 8$) requires the largest number of iterations.

(d) Memory requirements as a function of frequency. The high-frequency regime ($f = 8$) exhibits the largest memory footprint.

Figure 26: Preconditioner performance for the Helmholtz equation in a heterogeneous medium discretized with $N = 128^3$ degrees of freedom at increasing frequencies. The problem with $f = 0$ Hz represents a constant-coefficient Poisson problem, while $f = 8$ Hz represents a moderately high-frequency Helmholtz problem.

To give a comparison with traditional techniques, Table 6 shows the number of iterations that GMRES without preconditioner, the incomplete LU factorization, and algebraic multigrid as preconditioner require. For this problem

type, the ACR preconditioner was the only method that was able to solve all the problems under consideration.

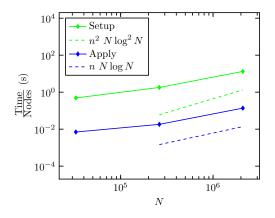| Frequency	GMRES + No Prec.		GMRES + ILU(30)		GMRES + AMG		GMRES + ACR	
	Iterations	Solve	Iterations	Solve	Iterations	Solve	Iterations	Solve
0	582	0.30	207	201.97	7	0.86	8	0.93
2	100,000+	-	100,000+	-	80	7.22	10	1.55
4	100,000+	-	100,000+	-	100,000+	-	12	2.13
8	100,000+	-	100,000+	-	100,000+	-	16	3.70

Table 6: Number of iterations and solve time for the solution of a sequence of increasingly indefinite Helmholtz problems $N = 128^3$ with a variable coefficient. Methods under consideration include the incomplete LU factorization (ILU), algebraic multigrid (AMG), and accelerated cyclic reduction (ACR).
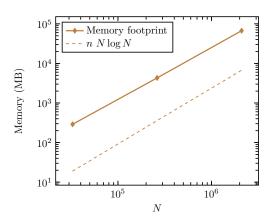
8.3. Operation count and memory footprint

As the previous experiments show, the high-frequency Helmholtz regime is where the highest numerical ranks are required. Therefore, it is of interest to show how the computations behave asymptotically as the problem size increases considering the estimate $k \sim O(n)$ [8]. Figure 27a shows a comparison of the preconditioner setup with the $O(n^2 \, N \log^2 N)$ estimate, and the preconditioner application with respect to the $O(n \, N \log N)$ estimate. Figure 27b shows the memory footprint of the preconditioner with respect to the estimate $O(n \, N \log N)$. Table 7 shows a fair agreement to the Chandrasekaran et al. estimate on the largest rank growth of the factorization, however, the average rank on the low-rank blocks of the ACR preconditioner grows slower than $k \sim O(n)$, which is reflected by a slightly lower than predicted memory consumption and setup time. The ACR preconditioner does not use the HSS format or a weak admissibility condition which results in off-diagonal blocks with large rank, but rather a standard admissibility condition that allows a more refined structure of the $\mathcal{H}$-matrix blocks, as discussed in Section 4.3, and shown in Figure 6.

N	**Largest rank**	**Average rank**
32^3	25	16
64^3	59	32
128^3	118	36

Table 7: Rank growth statistics for a sequence high-frequency Helmholtz problems in heterogeneous medium, discretized at 12 points per wavelength.

(a) Comparison of the preconditioner setup and application with their respective theoretical estimates.

(b) Comparison of the preconditioner memory footprint with its theoretical estimate.

Figure 27: Measured performance and memory footprint for the solution of a sequence high-frequency Helmholtz problems in heterogeneous medium, discretized at 12 points per wavelength. On average, the rank of the low-rank blocks of the ACR preconditioner grows slower than $O(n)$.

9. Concluding remarks

We presented a robust and scalable preconditioner based on the cyclic reduction method and hierarchical matrices in a distributed memory environment. The preconditioner relies on a block tridiagonal structure that commonly arise from the discretization of elliptic operators with variable-coefficient.

The preconditioner setup is based on a red-black ordering for which, if a 3D grid is considered, the ordering divides the grid into planes. These planes represent block rows of the original linear system, which are represented with a hierarchical matrix, in the $\mathcal{H}$ format, and its structure is defined using a binary spatial partitioning of the planar grid sections, employing a standard admissibility criterion that controls the rank of individual low-rank blocks.

The concurrency features of ACR constitute one of its strengths. The regularity of the decomposition allows a predictable load balance. The parallel features are demonstrated via the companion implementation in a distributed memory environment with numerical experiments that study the strong and weak scalability of the method. In our current implementation, concurrency at node level involves task-based parallelism of the hierarchical matrix arithmetic operations involved in the computation of the Schur complement and its evaluation. In future work, we plan on developing a set of distributed-memory hierarchical matrix operations that can exploit a larger set of processors to accelerate the setup phase of the preconditioner.

We demonstrated over a range of problem sizes and parameters that the preconditioner can tackle a broad class of problems that lack definiteness, such as the indefinite high-frequency Helmholtz equation in heterogeneous media, or lack symmetry, such as the convection-diffusion equation with a recirculating flow.

Since the accuracy of the $\mathcal{H}$-matrix approximations and their arithmetic operations can be tuned, it was demonstrated that the preconditioner could control the number of Krylov iterations. Furthermore, we discuss how these parameters can be used to optimize performance and memory consumption via comparisons with Krylov methods with established preconditioners such as algebraic multigrid and the incomplete LU factorization, and with direct solvers that perform a complete LU factorization.

As expected from all hierarchical low-rank approximations methods, the key to performance, and memory economy, is largely based on achieving an approximation with *low* rank; i.e. an efficient compression into a data-sparse format where k (the rank) is much less than n (the size of the block to be approximated). Numerical examples demonstrate that the required ranks agree with theoretical estimates and that for problems larger than a dozen of millions of unknowns the strong admissibility condition required less memory than the alternative (weak) admissibility condition.

10. Acknowledgements

References

[1] R. W. Hockney, A fast direct solution of Poisson's equation using Fourier analysis, Journal of the ACM 12 (1) (1965) 95–113. `doi:10.1145/321250.321259`.

[2] G. Chávez, G. Turkiyyah, D. Keyes, A direct elliptic solver based on hierarchically low-rank Schur complements, in: Proceedings of the 23rd International Conference on Domain Decomposition Methods in Science and Engineering XXIII (C.-O. Lee et al., eds.) Lecture Notes in Computational Science and Engineering, Springer International Publishing, 2017, pp. 135–143. `doi:10.1007/978-3-319-52389-7_12`.

[3] G. Chávez, G. Turkiyyah, S. Zampini, H. Ltaief, D. Keyes, Accelerated cyclic reduction: a distributed memory fast solver for structured linear systems, arXiv preprint arXiv:1701.00182.

[4] J. A. Meijerink, H. A. van der Vorst, 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.

[5] Y. Saad, Iterative methods for sparse linear systems, SIAM, 2003.

[6] R. Kriemann, Parallel $\mathcal{H}$-matrix arithmetics on shared memory systems, Computing 74 (3) (2005) 273–297. `doi:10.1007/s00607-004-0102-2`.

[7] R. Kriemann, $\mathcal{H}$-LU factorization on many-core systems, Computing and Visualization in Science 16 (3) (2013) 105–117. `doi:10.1007/s00791-014-0226-7`.

[8] 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 (5) (2010) 2261–2290.

[9] I. S. Duff, J. K. Reid, The multifrontal solution of indefinite sparse symmetric linear equations, ACM Transactions on Mathematical Software 9 (3) (1983) 302–325. `doi:10.1145/356044.356047`.

[10] R. Vandebril, M. Barel, G. Golub, N. Mastronardi, A bibliography on semiseparable matrices, Calcolo 42 (3-4) (2005) 249–270. `doi:10.1007/s10092-005-0107-z`.

[11] S. Chandrasekaran, M. Gu, T. Pals, A fast ULV decomposition solver for hierarchically semiseparable representations, SIAM Journal on Matrix Analysis and Applications 28 (3) (2006) 603–622. `doi:10.1137/S0895479803436652`.

[12] J. Xia, S. Chandrasekaran, M. Gu, X. Li, Superfast multifrontal method for large structured linear systems of equations, SIAM Journal on Matrix Analysis and Applications 31 (3) (2010) 1382–1411. `arXiv:http://dx.doi.org/10.1137/09074543X`, `doi:10.1137/09074543X`.

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

[14] J. Xia, Randomized sparse direct solvers, SIAM Journal on Matrix Analysis and Applications 34 (1) (2013) 197–227.

[15] P. Ghysels, X. S. Li, F.-H. Rouet, S. Williams, A. Napov, An efficient multicore implementation of a novel HSS-structured multifrontal solver using randomized sampling, SIAM Journal on Scientific Computing 38 (5) (2016) S358–S384.

[16] S. Wang, X. S. Li, F.-H. Rouet, J. Xia, M. V. De Hoop, A parallel geometric multifrontal solver using hierarchically semiseparable structure, ACM Transactions on Mathematical Software 42 (3) (2016) 21:1–21:21.

[17] S. Ambikasaran, E. Darve, An $\mathcal{O}(N \log N)$ fast direct solver for partial hierarchically semi-separable matrices, Journal on Scientific Computing 57 (3) (2013) 477–501. `doi:10.1007/s10915-013-9714-z`.

[18] A. Aminfar, E. Darve, A fast, memory efficient and robust sparse preconditioner based on a multifrontal approach with applications to finite-element matrices, International Journal for Numerical Methods in Engineering 107 (6) (2016) 520–540, nme.5196. `doi:10.1002/nme.5196`.

[19] C. Weisbecker, Improving multifrontal solvers by means of algebraic block low-rank representations, Ph.D. thesis, Institut National Polytechnique de Toulouse-INPT (2013).

[20] 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 (3) (2015) A1451–A1474.

[21] G. Chávez, Robust and scalable hierarchical matrix-based fast direct solver and preconditioner for the numerical solution of elliptic partial differential equations, Ph.D. thesis, King Abdullah University of Science and Technology (2017).

[22] I. Ibragimov, S. Rjasanow, K. Straube, Hierarchical Cholesky decomposition of sparse matrices arising from curl-curl-equation, Journal of Numerical Mathematics 15 (1) (2007) 31–57. `doi:10.1515/jnma.2007.031`.

[23] L. Grasedyck, R. Kriemann, S. Le Borne, Parallel black box $\mathcal{H}$-LU preconditioning for elliptic boundary value problems, Computing and Visualization in Science 11 (4-6) (2008) 273–291. `doi:10.1007/s00791-008-0098-9`.

[24] L. Grasedyck, R. Kriemann, S. Le Borne, Domain decomposition based $\mathcal{H}$-LU preconditioning, Numerische Mathematik 112 (4) (2009) 565–600. `doi:10.1007/s00211-009-0218-6`.

[25] J. Xia, M. Gu, Robust approximate Cholesky factorization of rank-structured symmetric positive definite matrices, SIAM Journal on Matrix Analysis and Applications 31 (5) (2010) 2899–2920. `doi:10.1137/090750500`.

[26] H. Pouransari, P. Coulier, E. Darve, Fast hierarchical solvers for sparse matrices using extended sparsification and low-rank approximation, arXiv preprint arXiv:1510.07363.

[27] W. Hackbusch, A sparse matrix arithmetic based on $\mathcal{H}$-matrices. Part I: Introduction to $\mathcal{H}$-matrices, Computing 62 (2) (1999) 89–108. `doi:10.1007/s006070050015`.

[28] M. Bebendorf, Hierarchical matrices: A means to efficiently solve elliptic boundary value problems, Vol. 63, Springer, 2008, Lecture Notes in Computational Science and Engineering.

[29] W. Hackbusch, Hierarchical matrices: Algorithms and analysis, Vol. 49, Springer Series in Computational Mathematics, 2015.

[30] B. L. Buzbee, G. H. Golub, C. W. Nielson, On direct methods for solving Poisson equations, SIAM Journal on Numerical Analysis 7 (4) (1970) pp. 627–656.

[31] R. Guivarch, L. Giraud, J. Stein, Parallel distributed fast 3D Poisson solver for meso-scale atmospheric simulations, International Journal of High Performance Computing Applications 15 (1) (2001) 36–46.

[32] B. Engquist, L. Ying, Sweeping preconditioner for the Helmholtz equation: hierarchical matrix representation, Communications on Pure and Applied Mathematics 64 (5) (2011) 697–735.

[33] J. Poulson, B. Engquist, S. Li, L. Ying, A parallel sweeping preconditioner for heterogeneous 3D Helmholtz equations, SIAM Journal on Scientific Computing 35 (3) (2013) C194–C212.

[34] G. Rodrigue, D. Wolitzer, Preconditioning by incomplete block cyclic reduction, Mathematics of Computation 42 (166) (1984) 549–565.

[35] A. Reusken, On the approximate cyclic reduction preconditioner, SIAM Journal on Scientific Computing 21 (2000) 565–590.

[36] P. Amodio, I. Gladwell, G. Romanazzi, An algorithm for the solution of bordered ABD linear systems arising from boundary value problems, in: Multibody Dynamics 2007, ECCOMAS, Milano (Italy), 2007.

[37] P. Amodio, M. Paprzycki, A cyclic reduction approach to the numerical solution of boundary value ODEs, SIAM Journal on Scientific Computing 18 (1) (1997) 56–68.

[38] W.-Y. Lin, C.-L. Chen, A parallel algorithm for solving tridiagonal linear systems on distributed memory multiprocessors, International Journal of High Speed Computing 6 (03) (1994) 375–386.

[39] G. Saghi, H. J. Siegel, J. L. Gray, Predicting performance and selecting modes of parallelism: a case study using cyclic reduction on three parallel machines, Journal of Parallel and Distributed Computing 19 (3) (1993) 219–233.

[40] R. A. Sweet, A parallel and vector variant of the cyclic reduction algorithm, SIAM Journal on Scientific and Statistical Computing 9 (4) (1988) 761–765.

[41] D. Goddeke, R. Strzodka, Cyclic reduction tridiagonal solvers on GPUs applied to mixed-precision multigrid, IEEE Transactions on Parallel and Distributed Systems 22 (1) (2011) 22–32.

[42] P. Quesada-Barriuso, J. Lamas-Rodríguez, D. B. Heras, M. Bóo, F. Argüello, Selecting the best tridiagonal system solver projected on multi-core CPU and GPU platforms, in: International Conference on Parallel and Distributed Processing Techniques and Applications, 2011.

[43] Y. Zhang, J. Cohen, J. D. Owens, Fast tridiagonal solvers on the GPU, ACM Special Interest Group on Programming Languages notices 45 (5) (2010) 127–136.

[44] C. Pheatt, Intel® threading building blocks, Journal of Computing Sciences in Colleges 23 (4) (2008) 298–298.

[45] M. R. Hestenes, E. Stiefel, Methods of conjugate gradients for solving linear systems, Journal of Research of the National Bureau of Standards 49 (1) (1952) 409–436.

[46] S. Y. Kadioglu, R. R. Nourgaliev, V. A. Mousseau, A comparative study of the harmonic and arithmetic averaging of diffusion coefficients for non-linear heat conduction problems, Tech. rep., INL/EXT-08-13999, Idaho National Laboratory, Idaho Falls, Idaho 83415 (2008).

[47] J. Mohring, R. Milk, A. Ngo, O. Klein, O. Iliev, M. Ohlberger, P. Bastian, Uncertainty quantification for porous media flow using multilevel Monte Carlo, in: Lirkov I., Margenov S., Waniewski J. (eds) International Conference on Large-Scale Scientific Computing. Lecture Notes in Computer Science, vol 9374. Springer, Cham, 2015, pp. 145–152.

[48] P. Bastian, M. Blatt, A. Dedner, C. Engwer, R. Klöfkorn, M. Ohlberger, O. Sander, A generic grid interface for parallel and adaptive scientific computing. Part I: Abstract framework, Computing 82 (2-3) (2008) 103–119.

[49] W. Briggs, V. Henson, S. McCormick, A Multigrid Tutorial, Second Edition, 2nd Edition, SIAM, 2000. `doi: https://doi.org/10.1137/1.9780898719505`.

[50] R. D. Falgout, U. M. Yang, hypre: a library of high-performance preconditioners, Springer Berlin Heidelberg, Berlin, Heidelberg, 2002, pp. 632–641.

[51] X. S. Li, J. W. Demmel, SuperLU DIST: A scalable distributed-memory sparse direct solver for unsymmetric linear systems, ACM Transactions on Mathematical Software 29 (2) (2003) 110–140.

[52] M. M. Gupta, J. Zhang, High accuracy multigrid solution of the 3D convection–diffusion equation, Applied Mathematics and Computation 113 (2) (2000) 249–274.

[53] O. G. Ernst, M. J. Gander, Why it is difficult to solve Helmholtz problems with classical iterative methods, in: Graham I., Hou T., Lakkis O., Scheichl R. (eds) Numerical Analysis of Multiscale Problems. Lecture Notes in Computational Science and Engineering, Vol. 83, Springer, 2012, pp. 325–363.

[54] L. Dalcin, N. Collier, P. Vignal, A. Côrtes, V. M. Calo, PetIGA: High-performance isogeometric analysis, Computer Methods in Applied Mechanics and Engineering 308 (2016) 151–181.

BibTeX entry of this article:

```
@article{Chavez2017,
author = "Gustavo Ch{\'a}vez and George Turkiyyah and Stefano Zampini and David Keyes",
title = "Parallel accelerated cyclic reduction preconditioner for three-dimensional
elliptic \{PDEs\} with variable coefficients ",
journal = "Journal of Computational and Applied Mathematics",
year = "2017",
issn = "0377-0427",
doi = "https://doi.org/10.1016/j.cam.2017.11.035",
url = "https://www.sciencedirect.com/science/article/pii/S0377042717305952",
}
```