International Journal of HIGH PERFORMANCE COMPUTING APPLICATIONS

Research Paper

The International Journal of High Performance Computing Applications
2022, Vol. 0(0) 1–18

DOI: 10.1177/10943420221115140
journals.sagepub.com/home/hpc


Compressed basis GMRES on high-performance graphics processing units

José I Aliaga[1], Hartwig Anzt[2,3], Thomas Grützmacher[2], Enrique S Quintana-Ortí[4] and Andrés E Tomás[4]

Abstract

Krylov methods provide a fast and highly parallel numerical tool for the iterative solution of many large-scale sparse linear systems. To a large extent, the performance of practical realizations of these methods is constrained by the communication bandwidth in current computer architectures, motivating the investigation of sophisticated techniques to avoid, reduce, and/or hide the message-passing costs (in distributed platforms) and the memory accesses (in all architectures). This article leverages Ginkgo's memory accessor in order to integrate a communication-reduction strategy into the (Krylov) GMRES solver that decouples the storage format (i.e., the data representation in memory) of the orthogonal basis from the arithmetic precision that is employed during the operations with that basis. Given that the execution time of the GMRES solver is largely determined by the memory accesses, the cost of the datatype transforms can be mostly hidden, resulting in the acceleration of the iterative step via a decrease in the volume of bits being retrieved from memory. Together with the special properties of the orthonormal basis (whose elements are all bounded by 1), this paves the road toward the aggressive customization of the storage format, which includes some floating-point as well as fixed-point formats with mild impact on the convergence of the iterative process. We develop a high-performance implementation of the "compressed basis GMRES" solver in the Ginkgo sparse linear algebra library using a large set of test problems from the SuiteSparse Matrix Collection. We demonstrate robustness and performance advantages on a modern NVIDIA V100 graphics processing unit (GPU) of up to 50% over the standard GMRES solver that stores all data in IEEE double-precision.

Keywords

Sparse linear systems, mixed precision, Krylov solvers, compressed basis GMRES, graphics processing units

Introduction

Krylov solvers enhanced with some type of sophisticated preconditioning technique nowadays compound a sound approach for the iterative solution of large and sparse linear systems (Saad, 2003). In particular, preconditioned Krylov solvers are often preferred over their direct counterparts for the solution of discretized high-dimensional problems (e.g., 3D problems), where a factorization-based direct solver would incur significant fill-in; see, for example, Davies (2006) and Saad (2003). Krylov solvers are also widely appealing for massively parallel architectures (such as graphics processing units or GPUs) due to their superior scalability.

At a high level, Krylov methods construct a basis that spans a Krylov subspace. At each iteration, this basis is expanded by adding another Krylov basis via the multiplication of the sparse coefficient matrix with the Krylov basis vector generated in the previous iteration and a sequence of vector operations and vector scalings to obtain an orthonormal set of Krylov basis vectors (Saad, 2003). Thus, each iteration of a Krylov solver comprises the multiplication of the sparse coefficient matrix with a vector (SPMV) plus a sequence of vector operations and reductions. All the numerical operations (kernels) appearing in Krylov methods are well-suited for parallelization. Unfortunately, most of these kernels, including SPMV, are memory-bound on virtually all modern processor architectures (Horowitz, 2014). As a result, many generic as well as hardware-specific

[1]Universitat Jaume I, Spain
[2]Karlsruhe Institute of Technology, Karlsruhe, Germany
[3]Innovative Computing Laboratory, University of Tennessee at Knoxville, TN, Knoxville, USA
[4]Universitat Politècnica de València, Valencia, Spain

Corresponding author:
Thomas Grützmacher, Karlsruher Institut fur Technologie, Hermann-von-Helmholtz-Platz 1, Eggenstein-Leopoldshafen 76344, Germany.
Email: thomas.gruetzmacher@kit.edu

optimization efforts for Krylov methods have focused on avoiding, reducing, or hiding (i.e., overlapping with computation) the communication/memory accesses of the algorithm. Some optimization techniques targeting the communication overhead include the following:

- The design of specialized (i.e., application-specific) sparse matrix data layouts that reduce the indexing information (overhead) and/or improve data locality when accessing the contents of the sparse coefficient matrix (Saad, 2003).
- The reorganization of the operations inside the body of the Krylov solver trading off reduced communication for an increase of computation per iteration, possibly also at the cost of introducing numerical instabilities that may result in slower convergence of the iteration; see, for example, Hoemmen (2010) and Cools (2019) and the references therein.
- The reformulation of the solver as an iterative refinement scheme combined with the use of mixed-precision for the storage of and arithmetic operations with the sparse coefficient matrix (Higham, 2002).
- The utilization of adaptive-precision schemes for memory-bound preconditioners (Anzt et al., 2019a).

In this article, we also address the communication costs of Krylov methods, focusing on the generalized minimal residual (GMRES) algorithm, a Krylov solver for general linear systems that explicitly maintains the complete set of Krylov basis vectors instead of relying on short recurrences (as many other Krylov solvers do) (Saad, 2003). Orthogonally to all previous communication optimization efforts, our optimized variant of the GMRES algorithm reduces communication in the access to the Krylov basis during the iteration loop body. In more detail, our GMRES algorithm leverages Ginkgo's memory accessor, introduced in Anzt et al. (2021) and Grützmacher et al. (2021), to decouple the memory storage format from the arithmetic precision so as to maintain the Krylov basis vectors in a compact "reduced precision" format. This radically diminishes the memory access volume during the orthogonalization, while not affecting the convergence rate of the solver, yielding notable performance improvements. Concretely, we make the following contributions in our article:

- We follow the ideas in Anzt et al. (2021) and Grützmacher et al. (2021) and use the therein presented "memory accessor" in order to decouple the memory storage format from the arithmetic precision, specifically applying this strategy to maintain the Krylov basis in reduced precision in memory while performing all arithmetic operations using full, hardware-supported IEEE 64-bit double-precision (DP).
- We analyze the benefits that result from casting the Krylov basis into different compact storage formats, including the natural IEEE 32-bit single-precision (SP) and 16-bit half-precision (HP) as well as some other non-IEEE fixed point-based alternatives enhanced with vector-wise normalization.
- We integrate the mixed-precision GMRES algorithm into the Ginkgo sparse linear algebra library (https://ginkgo-project.github.io).
- We provide strong practical evidence of the advantage of our approach by developing a high-performance realization of the solver for modern NVIDIA's V100 GPUs and testing it on a considerable number of large-scale problems from the SuiteSparse Matrix Collection (Davis and Hu, 2011) (https://sparse.tamu.edu/).

The rest of the article is organized as follows. First, we review a list of related works in the direction of mixed-precision Krylov solvers, and then, we briefly recall the GMRES algorithm before motivating the *compressed basis GMRES (CB-GMRES)* storing the orthonormal basis in reduced precision. Next, we provide details about how we decouple the memory precision from the arithmetic precision and how we realize the implementation of CB-GMRES in Ginkgo. The experimental evaluation of the CB-GMRES implementation follows, assessing the accuracy, convergence, performance, and flexibility of the developed algorithm. We conclude with a summary of the findings and ideas for future research.

Related work

The potential of using lower precision in different components of a Krylov solver has been previously investigated for both Lanczos-based (short-term recurrence) and Arnoldi-based (long-term recurrence) algorithms and the associated methods for linear systems of equations.

From the theoretical point of view, most of those works are based on rounding error analysis for Krylov solvers running in finite precision. In a relevant result, Paige (1980) derived distinct relations between the loss of orthogonality and other important quantities in finite precision Lanczos. Greenbaum (1989) extended these results to analyze backward stability for the CG method in finite precision. She also derived theoretical bounds for the maximum attainable accuracy in finite precision for CG, BiCG, BiCGSTAB, and other Lanczos-based methods (Greenbaum, 1997). Carson (2015) expanded these results to s-step Lanczos/CG variants, deducing that an s-step Lanczos in finite precision behaves like a classical Lanczos run in lower "effective" precision, where this "effective" precision depends on the conditioning of the polynomials

used to generate the s-step bases. Additional bounds for Lanczos-based Krylov solvers running in finite precision can be found in Meurant and Strakoš (2006).

Simoncini and Szyld (2003) and van Den Eshof and Sleijpen (2004) established a theory on "inexact Krylov subspace methods" that applies the basis-generating SPMV as a perturbed operator carrying some error, which may reflect situations where reducing the cost of the operator application is essential or where the operator is only available as an approximation. Theoretical results prove that inexact Krylov methods can achieve the same solution accuracy as high precision Krylov solvers if the error in the perturbed operator is controlled and adapted to the residual (Simoncini and Szyld, 2003 and van Den Eshof and Sleijpen, 2004).

Concerning long-recurrence strategies, Gratton et al. (2020) combined the previous findings from Björck (1967), Paige (1980), and Paige et al. (2006) to derive theoretical norms for a mixed-precision GMRES algorithm based on modified Gram–Schmidt. In this algorithm, they consider using inexact (e.g., single-precision) inner products in the orthogonalization process, which results in a loss of double-precision (DP) orthogonality of the Krylov basis vectors. This makes the work by Gratton et al. (2020) very similar to our approach. However, our solution is different in several aspects:

- In our CB-GMRES solver, we decouple the arithmetic precision from the memory storage format to maintain the basis vectors in lower precision while using high precision in all arithmetic;
- we consider not only IEEE single-precision as the reference compact storage format but also IEEE half-precision (HP) and fixed-point formats based on 32-bit and 16-bit integers;
- we realize a production-ready and sustainable implementation of CB-GMRES for high-performance GPU architectures based on classical Gram–Schmidt with re-orthogonalization; and
- we provide comprehensive experimental results analyzing accuracy, convergence, and performance of our CB-GMRES solver.

The GMRES algorithm

Consider the linear system

$$Ax = b \tag{1}$$

where the coefficient matrix $A \in \mathbb{R}^{n \times n}$ is sparse, with n_z nonzero entries, $b \in \mathbb{R}^n$ represents the right-hand side vector, and $x \in \mathbb{R}^n$ contains the sought-after solution vector. Figure 1 displays a mathematical formulation of the restarted GMRES algorithm for the iterative solution of (1). There we assume that $M \in \mathbb{R}^{n \times n}$ defines an appropriate preconditioner, x_0 is an initial approximation to the actual solution, e_1 stands for the first column of the square identity matrix of order $m + 1$, and the scalars m and η, respectively, define the dimension of the orthogonal basis and the threshold for the re-orthogonalization. The orthogonalization mechanism in the algorithm relies on the classical Gram–Schmidt (CGS) method, but a version that employs the modified Gram–Schmidt (MGS) variant is simple to derive from that (see Golub and Loan, 1996). Unless explicitly stated, we prefer CGS with re-orthogonalization over MGS as it is competitive in terms of accuracy while enabling higher performance on modern hardware architectures via the use of BLAS-2 routines. The stopping criterion for the iterative algorithm can be based, for example, on the residual $\|r_m\|_2 = \|b - Ax_m\|_2$ being smaller than a certain relative threshold $\tau \cdot \|b\|_2$. The GMRES algorithm can internally keep track of the residual by iteratively updating the residual vector in every iteration (for brevity, we omit this feature in Figure 1). However, rounding effects can cause the iterative residual to differ from the explicit residual, and therefore, every restart explicitly computes the residual to re-align the iteratively computed residual.

From the computational point of view, the main kernels appearing in the GMRES algorithm correspond to the application of the preconditioner M and the SPMV operation with the coefficient matrix A (both in Line 3), the orthogonalization of vector w with respect to the vectors in the basis V_j (Lines 5 and 8), the solution of the linear least squares (LLS) problem (Line 17), the assembly of the next iterate, which requires the application of the orthogonal basis followed by the preconditioner (Line 17), and a few minor vector operations such as AXPYS and vector scaling.

The LLS problem in the GMRES algorithm can be solved via the QR factorization (Golub and Loan, 1996), where this decomposition can be cheaply obtained using an updating technique as the Hessenberg matrices for two consecutive iterations that differ only in one column. Therefore, the cost associated with the solution of this problem is minor in comparison with that of the global algorithm. In addition, the operations that are necessary to update the new estimate to the solution x_m (Line 17) also contribute a minor cost to the overall procedure, as they are m times less frequent in comparison with the kernel calls in Lines 3, 5, and 8.

CB-GMRES storing the orthonormal basis in reduced precision

For simplicity, consider that the GMRES algorithm integrates a simple preconditioner, such as a Jacobi scheme (or a

1. Compute $r_0 := b - Ax_0$, $\beta := \|r_0\|_2$, and $v := r_0/\beta$. Set $V_1 = [v]$
2. **for** $j := 1, 2, \ldots, m$
3. Compute $w := A(M^{-1}v)$
4. $\omega := \|w\|_2$
5. Orthogonalize $h_{1:j,j} := V_j^T w$, $w := w - V_j h_{1:j,j}$
6. $h_{j+1,j} := \|w\|_2$
7. **if** $(h_{j+1,j} < \eta\,\omega)$ **then**
8. Re-orthogonalize $u := V_j^T w$, $w := w - V_j u$
9. $h_{1:j,j} := h_{1:j,j} + u$
10. $h_{j+1,j} := \|w\|_2$
11. **endif**
12. **if** $(h_{j+1,j} = 0)$ **or** $(h_{j+1,j} < \eta\,\omega)$ **then** set $m := j$ and **go to step 17**, **endif**
13. $v := w/h_{j+1,j}$
14. Set $V_{j+1} := [V_j,\ v]$
15. **endfor**
16. Define the $(m+1) \times m$ Hessenberg matrix $\bar{H}_m = \left(h_{ij}\right)_{1 \le i \le m+1, 1 \le j \le m}$
17. Compute y_m the minimizer of $\|\beta e_1 - \bar{H}_m y\|_2$ and $x_m := x_0 + M^{-1}(V_m y_m)$
18. **if** satisfied **then Stop**, **else** set $x_0 := x_m$ and **go to step 1**, **endif**

Figure 1. Algorithmic formulation of the restarted GMRES algorithm for the solution of sparse linear systems.

block-Jacobi variant with a small block size) (Saad, 2003). The performance of the algorithm is then strongly determined by the costs of the SPMV kernel and the general matrix–vector products (GEMV), with V_j^T and V_j. These are memory-bound kernels, with their execution times largely dictated by the number of memory accesses (memory operations or memops hereafter). The optimization we propose thus aims to reduce the cost of the GEMV operations by storing the Krylov basis V_j in a more compact reduced precision format.

In order to analyze the theoretical memop count of the SPMV and the GEMV kernels, for simplicity, let us assume the following:

1. The right-hand side vectors for both types of matrix–vector products reside in cache. In general, this is not true but, for the following theoretical derivation, the memory layer where the vectors reside is not important.
2. The sparse coefficient matrix is stored using the compressed sparse row (CSR) format. This is a general and flexible data layout that employs one integer per nonzero value to represent its column index, plus $n + 1$ integers for the row pointers (Saad, 2003).
3. The re-orthogonalization mechanism included in the GMRES algorithm (Lines 7–11 in Figure 1) is not needed.

Then, the ratio between the contributions of the two GEMV and the CSR SPMV to the memop count, due to the accesses to the corresponding matrices, is given by

$$\frac{\text{Memops GEMV}}{\text{Memops SPMV}} = \frac{2nm'}{n_z(1+f)+(n+1)f} \approx \frac{2nm'}{ns(1+f)+nf} = \frac{2m'}{s(1+f)+f} \tag{2}$$

where $s = n_z/n$ is the average number of nonzero entries per row of the sparse matrix; $m' = j - 1$ is the size of the already-computed Krylov basis, that is, the number of vectors the new basis vector is orthogonalized against; and f represents a factor for the indexing overhead into the sparse data structures, for example, when using 32-bit integers to represent the indices and 64-bit for the data values, $f = 32/64 = 1/2$.

For a non-restarted version of GMRES, the size of the Krylov subspace m' steadily grows with the iteration count ($m' = j - 1$ at iteration j), which hints that the memory operations related to the orthogonalization can quickly dominate the cost. In practical implementations though, the GMRES solver is usually enhanced with a restart mechanism, as in the formulation of the algorithm in the previous section, to keep both the memory requirements and the orthogonalization cost at reasonable levels. Depending on the problem size and the available resources, the typical values for the restart parameter vary between $m = 30$ and 200. At the same time, the nonzero-per-row ratio s is in general relatively small and often significantly smaller than the restart parameter m. Therefore, considering the memory operations in that restart cycle, equation (2) then becomes

$$\frac{\text{Memops GEMV}}{\text{Memops SPMV}} = \frac{\sum_{j=1}^{m-1} 2nj}{m(n_z(1+f)+(n+1)f)} \approx \frac{m}{s(1+f)+f}. \tag{3}$$

With typical parameters $f = 1/2$ (CSR format) and a restart parameter $m = 100$, the memory access count due to the orthogonalization theoretically thus exceeds the memory access overhead for the SPMV kernel for matrices with ratio $s = n_z/n < 67$.

CB-GMRES. In order to reduce the memory access volume in the orthogonalization step of GMRES, we propose to store the vectors of the Krylov basis V_j in a compact reduced precision format, retrieve the data from memory in that format, and transform the values into IEEE 64-bit DP prior to the orthogonalization computations they are involved in (Lines 5, 8, and 17). This advocates for decoupling the memory storage format from the arithmetic precision while preserving IEEE 64-bit precision in the arithmetic operations (Anzt et al., 2019b).

The decoupling strategy provides full flexibility in terms of choosing a memory representation format, enabling the usage of the natural IEEE 16-bit or 32-bit formats as well as other, "more flexible" alternatives (with no hardware support for the arithmetic). In particular, the property that the entries of the orthonormal vectors forming the Krylov basis are all bounded by 1 pushed us to explore the efficiency of more aggressive customized formats. For example, it is possible to reduce the number of bits employed for the exponent in the floating-point format by normalizing it with respect to a baseline factor. In our investigation, we take this approach to the extreme, resulting in the evaluation of fixed-point formats for the storage of the orthogonal basis. For this purpose 1) we normalize each vector of the basis by scaling its entries with (the inverse of) its largest vector entry (in absolute value); and 2) we then store only the fractional part of each value of the result as an integer number, plus the normalization factor for each vector.

For convenience, we refer to the resulting algorithm as *compressed basis GMRES (CB-GMRES)* in the remainder of the article. We again emphasize that we still use DP in all arithmetic operations and only use lower precision formats for the memory operations.

Discussion. Storing the orthogonal basis of a Krylov method in a reduced precision format will typically introduce rounding errors that may affect the numerical properties of the method, potentially impacting the convergence and numerical stability of the iterative solver. As the solution approximation is optimal in the generated Krylov subspace, the perturbed Krylov basis vectors may result in a loss in the DP orthogonality of the basis vectors and a different (Krylov) subspace and in which the solution approximation is computed. However, the solution approximation process in the LLS solver accounts for the perturbed basis vectors, and as long as the generated subspace allows for a good approximation of the solution in the sense that there exists a good solution approximation in the space spanned by the Krylov basis vectors, this approximation will be found in the optimization process. Hence, on the one hand, as long as the basis vectors are "relatively" close to the optimal basis vectors, we can expect that the convergence will be only mildly affected. On the other hand, we may assume that the need for increasing the Krylov subspace dimension (equivalent to additional iterations) can be compensated by the faster execution of each iteration. The numerical experiments in this work generally confirm this assumption.

To close this section, we emphasize that:

- Our approach is complementary to other techniques which aim to reduce the memory access overhead, in the sense that it can be combined with these other optimizations. For example, CB-GMRES can be complemented by customizing the sparse matrix data layout to the application, operating with an iterative refinement scheme, or exploiting customized precision in the preconditioner, among others.
- The arithmetic precision is decoupled from the representation format so that we can actually store the data for the Krylov basis in any format while relying on the data types with hardware support for the arithmetic operations.

Implementation of CB-GMRES

The Ginkgo sparse linear algebra library

For convenience and ease of use, we have realized the CB-GMRES algorithm in the Ginkgo ecosystem (Anzt et al., 2020a,b): Ginkgo is a sparse linear algebra library implemented in modern C++ that embraces two principal design concepts: The first principle, aiming at future technology readiness, is to consequently separate the numerical algorithms from the hardware-specific kernel implementation to ensure correctness (via comparison with sequential reference kernels), performance portability (by applying hardware-specific kernel optimizations), and extensibility (via kernel backends for other hardware architectures). The second design principle—pursuing user-friendliness—is the convention to express functionality in terms of linear operators: every solver, preconditioner, factorization, matrix–vector product, and matrix reordering are expressed as linear operators (or composition thereof). This allows to easily combine the CB-GMRES with any preconditioner available in Ginkgo and to select the realization of the SPMV kernel that is most appropriate for the characteristics of a specific problem (Anzt et al., 2020c).

Ginkgo relies on an "executor" concept to favor platform portability. The executor encapsulates all information about memory location and execution domain of linear algebra objects and automatically orchestrates memory allocation, memory transfers, and kernel selection. For the CB-GMRES implementation with the orthonormal Krylov basis stored in reduced precision, the executor concept is

extended with a "memory accessor," described next, that handles the data conversion transparently to the user.

Memory accessor

At a high level, the idea of the CB-GMRES solver is to compress the orthogonal matrix/vector before and after the memory operations using one of the reduced/customized storage formats but still use the working precision (i.e., IEEE754 DP) for the arithmetic operations (Anzt et al., 2021 and Grützmacher et al., 2021). Retrieving the orthonormal basis in reduced precision from memory thus requires reading the basis contents and converting them into DP. When these values are stored in SP, the conversion is easy to perform via a datatype casting operator. For fixed-point representations, though, the conversion requires some additional manipulations plus the scaling with a normalization factor.

To decouple the memory access and conversion from the code development effort, we use a memory accessor that converts the data between DP and the memory storage/communication format on-the-fly (see Figure 2). The efficient implementation of the accessor aims to hide the cost of these data conversions by overlapping them with the memory accesses, in principle introducing a minor or even negligible overhead. In addition, the introduction of this technique can accelerate the execution as accessing the data in lower precision significantly reduces the volume of memory accesses per iteration.

Considering the realization of the CB-GMRES algorithm, after the new basis vector $v_j = v$ is formed at iteration j, the memory accessor is activated in order to compress the DP contents of this vector before storing them into memory; see Lines 13 and 14 in the algorithm in Figure 1. The memory accessor is also active when retrieving the full orthogonal basis V_j from memory; see Lines 5 and 8 of the algorithm.

On the technical side, the accessor leverages static polymorphism (via C++ templates) for both the arithmetic precision (in our work, fixed to IEEE DP) and the memory format. While this flexible design can accommodate any memory format, we currently only support fp64, fp32, and fp16 (for ieee DP, SP, and HP, respectively) and int32 and int16 (for 32-bit and 16-bit fixed-point formats) in Ginkgo. The versions based on integer formats rely on a fixed-point representation in order to maintain the orthonormal basis vectors. This representation only requires a fractional part because the vectors are normalized, making each vector entry smaller than one. However, this is not efficient for large vectors because the largest absolute value will likely be significantly smaller than one, therewith wasting representation range (and precision). To optimize the accuracy, a different scaling factor is used for each vector

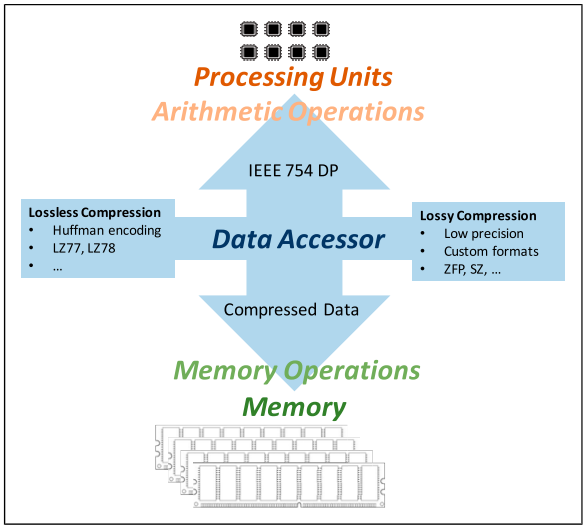

Figure 2. Accessor separating the memory format from the arithmetic format and realizing on-the-fly data conversion in each memory access.

$$\sigma_j = \frac{\|v_j\|_\infty / \|v_j\|_2}{\text{max\_intbb}}$$

where v_j is the vector computed at iteration j before normalization and max_intbb is the maximum positive value of the integer representation using bb $\in \{16, 32\}$ bits. Both norms can be computed simultaneously so that the extra overhead due to the memory accesses to obtain the infinity norm remains small. The vector v_j is then stored in V_{j+1} as

$$V_{j+1} = \left[V_j, v_j/\sigma_j\right],$$

and any subsequent access to the contents of V_{j+1} implies an intrinsic post-multiplication by a diagonal matrix $\Sigma_{j+1} = \text{diag}(\sigma_1, \sigma_2, \ldots, \sigma_j, \sigma_{j+1})$ that contains the scaling factors on the diagonal. This scaling adds one multiplication per element to the computational cost of any operation involving the orthogonal basis and storing the scaling factor in memory. However, as the whole algorithm is heavily bandwidth-bound, we expect that this arithmetic overhead remains small.

Experimental evaluation of the compressed basis GMRES

In this section, we analyze several properties of the CB-GMRES algorithm in order to assess the benefits of this solver as part of a production code. Concretely, we investigate the following questions:

1. Can we achieve high accuracy in the solution approximations?
2. How significant is the convergence delay introduced by moving away from the "full" precision Krylov

basis vectors and utilizing instead basis vectors that are low precision approximations of these orthonormal vectors?
3. What are the performance advantages of the CB-GMRES over the standard (DP) GMRES?
4. Which specific storage format should we use for the memory operations?

Setup

To answer these questions, we select a subset of 49 large-scale test matrices from the SuiteSparse Matrix Collection (Davis and Hu, 2011) that we adopt as benchmark problems to explore the accuracy, convergence, and performance of the CB realizations of the GMRES algorithm. The selected test matrices (listed along with some key properties in Table 1) correspond to the largest regular test cases from the SuiteSparse Matrix Collection for which a preconditioned GMRES running in double-precision converges within a reasonable iteration count. All solver implementations are part of the Ginkgo library and utilize building blocks from the Ginkgo environment such as preconditioners, utility functions, and benchmarking environments. The SPMV kernel integrated into all variants of GMRES to generate the Krylov basis vectors is Ginkgo's CSR-based SPMV routine; this particular realization of SPMV maintains the coefficient matrix in compressed sparse row (CSR) format and accounts for the nonzero distribution in the target matrices by automatically selecting the CSR kernel that provides the best performance (Anzt et al., 2020c). The orthogonalization kernel inside the GMRES solvers is based on CGS with optional reorthogonalization. For the majority of the test cases, this strategy has shown to be superior over the MGS orthogonalization in terms of accuracy and runtime. For convenience, we offer the convergence results for an MGS-based GMRES in the supplementary material. The GMRES and CB-GMRES solvers are algorithmically identical except that the CB-GMRES algorithm can store the basis in reduced precision. We use the notation GMRES<arith, mem>, where arith refers to the precision format used in the arithmetic operations and mem refers to the precision format used to store the Krylov basis. Consequently, GMRES<fp64, fp64> and GMRES<fp32, fp32> denote the standard GMRES solver operating in ieee 754 DP and ieee 754 SP, respectively. In contrast, GMRES<fp64, fp32>, GMRES<fp64, fp16>, GMRES<fp64, int32>, and GMRES<fp64, int16> are CB-GMRES versions differing in the memory precision format. In Table 2, we list the distinct GMRES variants we use in the remainder of the article along with details and markers we use in the performance graphs.

Table 1. Test matrices.

Matrix	Size	Non-zero	Non-zeros per row
af_0_k101	503,625	17,550,675	34.8
af_1_k101	503,625	17,550,675	34.8
af_2_k101	503,625	17,550,675	34.8
af_3_k101	503,625	17,550,675	34.8
af_4_k101	503,625	17,550,675	34.8
af_5_k101	503,625	17,550,675	34.8
af_shell1	504,855	17,562,051	34.8
af_shell10	1,508,065	52,259,885	34.7
af_shell2	504,855	17,562,051	34.8
af_shell3	504,855	17,562,051	34.8
af_shell4	504,855	17,562,051	34.8
af_shell5	504,855	17,579,155	34.8
af_shell6	504,855	17,579,155	34.8
af_shell7	504,855	17,579,155	34.8
af_shell8	504,855	17,579,155	34.8
af_shell9	504,855	17,588,845	34.8
apache2	715,176	4,817,870	6.7
Atmosmodd	1,270,432	8,814,880	6.9
Atmosmodj	1,270,432	8,814,880	6.9
Atmosmodl	1,489,752	10,319,760	6.9
Atmosmodm	1,489,752	10,319,760	6.9
audikw_1	943,695	77,651,847	82.3
bone010	986,703	47,851,783	48.5
boneS10	914,898	40,878,708	44.7
Bump_2911	2,911,419	127,729,899	43.9
circuit5M_dc	3,523,317	14,865,409	4.2
Cube_Coup_dt6	2,164,760	124,406,070	57.5
CurlCurl_2	806,529	8,921,789	11.1
CurlCurl_3	1,219,574	13,544,618	11.1
CurlCurl_4	2,380,515	26,515,867	11.1
ecology1	1,000,000	4,996,000	5.0
ecology2	999,999	4,995,991	5.0
Fault_639	638,802	27,245,944	42.7
Flan_1565	1,564,794	114,165,372	73.0
G3_circuit	1,585,478	7,660,826	4.8
Geo_1438	1,437,960	60,236,322	41.9
Hook_1498	1,498,023	59,374,451	39.6
inline_1	503,712	36,816,170	73.1
ldoor	952,203	42,493,817	44.6
mc2depi	525,825	2,100,225	4.0
ML_Geer	1,504,002	110,686,677	73.6
parabolic_fem	525,825	3,674,625	7.0
Serena	1,391,349	64,131,971	46.1
Ss	1,652,680	34,753,577	21.0
t2em	921,632	4,590,832	5.0
thermal2	1,228,045	8,580,313	7.0
tmt_sym	726,713	5,080,961	7.0
tmt_unsym	917,825	4,584,801	5.0
Transport	1,602,111	23,487,281	14.7

Table 2. List of solvers we consider along with the markers we use in the remainder of the article.

✳	MGS-GMRES<fp64,fp64>	DP GMRES based on MGS orthogonalization
☆	GMRES<fp64,fp64>	DP GMRES (baseline in CB-GMRES evaluation)
○	MGS-GMRES<fp32,fp32>	SP GMRES based on MGS orthogonalization
▽	GMRES<fp64,fp32>	CB-GMRES using fp32 for storing the Krylov basis
△	GMRES<fp64,fp16>	CB-GMRES using fp16 for storing the Krylov basis
◊	GMRES<fp64,int32>	CB-GMRES using int32 for storing the Krylov basis
–	GMRES<fp64,int16>	CB-GMRES using int16 for storing the Krylov basis

In the performance tests, we utilize Ginkgo's CUDA executor, which is heavily optimized for NVIDIA GPUs. We run all experiments on an NVIDIA V100 GPU with support for compute capability 7.0 NVIDIA Corp (2017). The V100 accelerator board is equipped with 16 GB of main memory, 128 KB L1 cache, and 6 MB of L2 cache. A bandwidth test reported 897 GB/s for main memory access in this particular device (Tsai et al., 2020). The theoretical peak performance for the V100 GPU is 7.83 DP TFLOPS (i.e., $7.83 \cdot 10^{12}$ floating-point operations per second). Ginkgo's CUDA backend was compiled using CUDA version 9.2.

Impact on basis orthogonality

As a first step in the experimental analysis, we investigate how rounding the values in the Krylov basis vectors to a lower precision format affects the quality of the Krylov basis. In particular, we are interested in the orthogonality loss. For that, we choose the GMRES<fp64,fp32> configuration where all arithmetic operations use IEEE 754 DP while IEEE 754 SP is used for storing the basis vectors. As a metric to assess the orthogonality at iteration k, we consider the basis $V = [v_1, v_2 \ldots v_{k+1}]$ and evaluate $\|V^T \cdot V - I\|_\infty = \max_{1 \le i \le m} \sum_{j=1}^{n} |(V^T \cdot V - I)_{ij}|$. Figure 3 visualizes this metric before and after the re-orthogonalization step for the CB-GMRES solver and compares the results of this orthogonality metric against those attained from a standard DP GMRES and a standard SP GMRES. The analysis reveals that the orthogonality of the Krylov basis vectors degrades when the basis vectors are stored in lower precision, yet the quality remains superior to that observed for SP GMRES. We next investigate how storing the Krylov basis in a compressed form impacts the attainable accuracy.

Accuracy of CB-GMRES

To investigate whether the CB-GMRES can match the accuracy levels attained by DP GMRES, we consider 49 linear systems of the form $Ax = b$, with the coefficient matrix defined from the test matrices in Table 1. For each problem, we set the i-th entry of the exact solution x to $x[i] = \sin(i)$, $i = 1, 2, \ldots, n$; then scale x to a unit norm ($x := x/\|x\|_2$); and finally generate b as $b := A \cdot x$. The GMRES algorithm is started with an initial guess $x_0 = 0$, uses a restart parameter $m = 100$, and is stopped when the solution approximation x^* yields a residual $\|Ax^* - b\|_2 \le 10^{-12}\|b\|_2$. In this experiment, we consider both the non-preconditioned solvers and a setting where all methods (both GMRES and CB-GMRES) are left-preconditioned with a scalar Jacobi preconditioner—which is equivalent to scaling the linear problems to a unit diagonal prior to the iterative solution process. The motivation is that a Jacobi-preconditioned GMRES is more realistic for production use while the non-preconditioned GMRES may enhance numerical instabilities. The Jacobi preconditioner is always stored in the arithmetic precision of the solver in order to isolate the convergence difference to the change in the storage format.

To avoid expensive explicit residual computations, the GMRES algorithm internally updates a recurrence residual that is used to check convergence. However, when using finite precision and due to the accumulation of rounding error, this iteratively computed residual can diverge from the real residual, and the GMRES algorithm may stop "too early" even though the real residual did not fall below the selected threshold. Using the compressed basis formats to store the orthonormal basis may enlarge this effect. To tackle this problem, we modify all the implementations to compute the explicit residual once convergence is indicated by the recurrence residual but continue iterating with the updated residual in case the actual accuracy threshold is not fulfilled.

To assess the solution accuracy, in Figure 4, we report the normalized explicitly computed residual $\|Ax^* - b\|_2/\|b\|_2$ for the solution approximations generated with the distinct CB-GMRES versions either executed as a plain algorithm (top) or enhanced with a scalar Jacobi preconditioner (bottom). For comprehensiveness, we report the final residual norm also for those cases where the accuracy target cannot be reached. In these initial results, we observe that the standard DP GMRES based on CGS with re-orthogonalization (GMRES<fp64,fp64>), the standard DP GMRES based on modified Gram–Schmidt (MGS-GMRES<fp64,fp64>), and the CB-GMRES variants storing the basis vectors in 32-bit floating-point precision

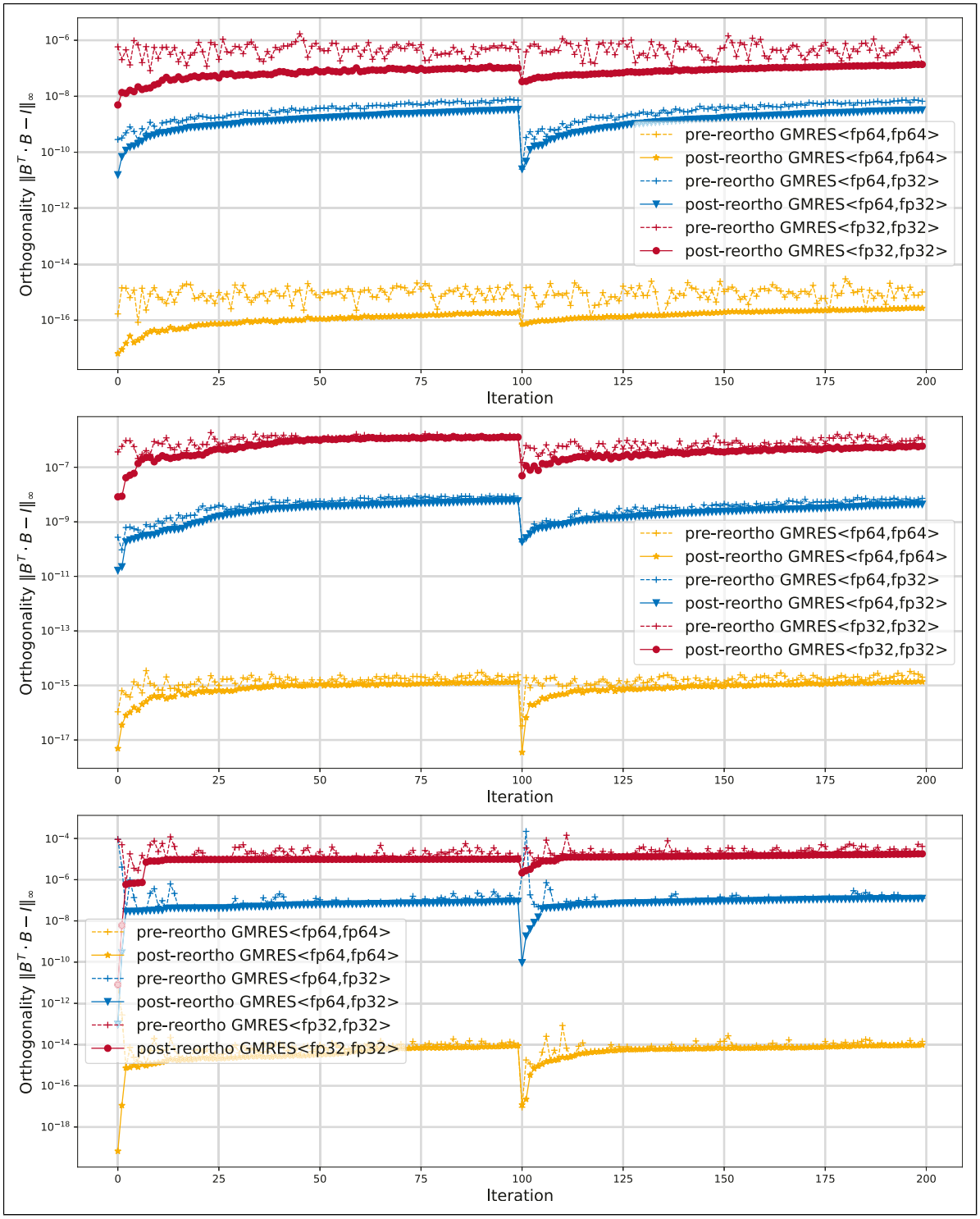

Figure 3. Orthogonality of the Krylov basis vectors before and after re-orthogonalization for the standard DP GMRES and SP GMRES and the CB-GMRES GMRES<fp64,fp32>. The test matrices are parabolic_fem (top), ss (center), and circuit5M_dc (bottom).

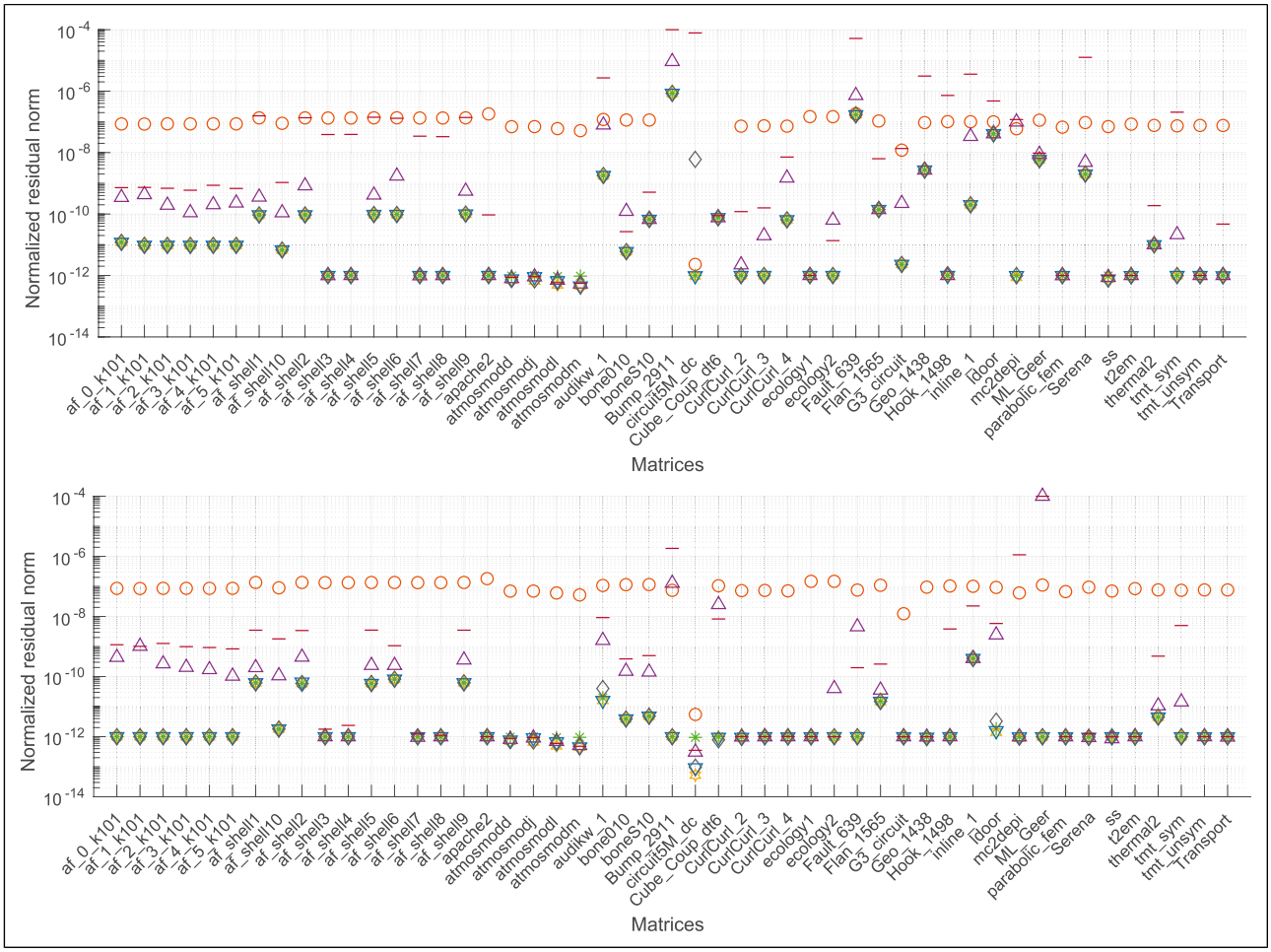

Figure 4. Normalized residual of plain GMRES (top) and Jacobi-preconditioned GMRES (bottom) using different precision for arithmetic and memory operations. Detailed results are listed in the supplementary material.

(GMRES<fp64,fp32>) or 32-bit fixed-point precision (GMRES<fp64,int32>) generally achieve the same residual accuracy for both the non-preconditioned application and the Jacobi-preconditioned case. We furthermore observe that an SP GMRES (MGS-GMRES<fp32,fp32>) fails to provide solution approximations of the same accuracy level and may therefore be disregarded as a valid option when aiming for high accuracy. By storing the Krylov basis in fp16 or int16, the CB-GMRES algorithm converges to a solution of lower accuracy, however, often still achieving a residual accuracy better than an SP GMRES (MGS-GMRES<fp32,fp32>).

Convergence of CB-GMRES

Next, we investigate the impact of storing the basis vectors in a compressed format on the convergence of GMRES. To expose the effects, in Figure 5, we visualize the convergence behavior of the Jacobi-preconditioned CB-GMRES variants for the circuit5M_dc and Serena problems. While all CB-GMRES variants ultimately reach the relative residual stopping criterion, the storage format selected for the Krylov basis impacts the convergence rate and, in consequence, the iteration count. The spikes in the residual curves occur in the iterations where the recurrently computed residual is updated by an explicit residual. This occurs when the Krylov basis size reaches the restart parameter or the recurrence residual indicates convergence. For GMRES<fp64,int16> in particular, we notice significant corrections of the recurrence residual. Overall, we observe that using a compressed format to store the Krylov basis can delay convergence and require additional basis vectors. In order to quantify this effect for a broad range of problems, in Figure 6, we display the iteration count of the CB-GMRES variants relative to the DP GMRES iteration count if they reach the same residual accuracy (even if that accuracy does not fulfill the residual norm stopping criteria). Again, we include results for the non-preconditioned solvers (top) and

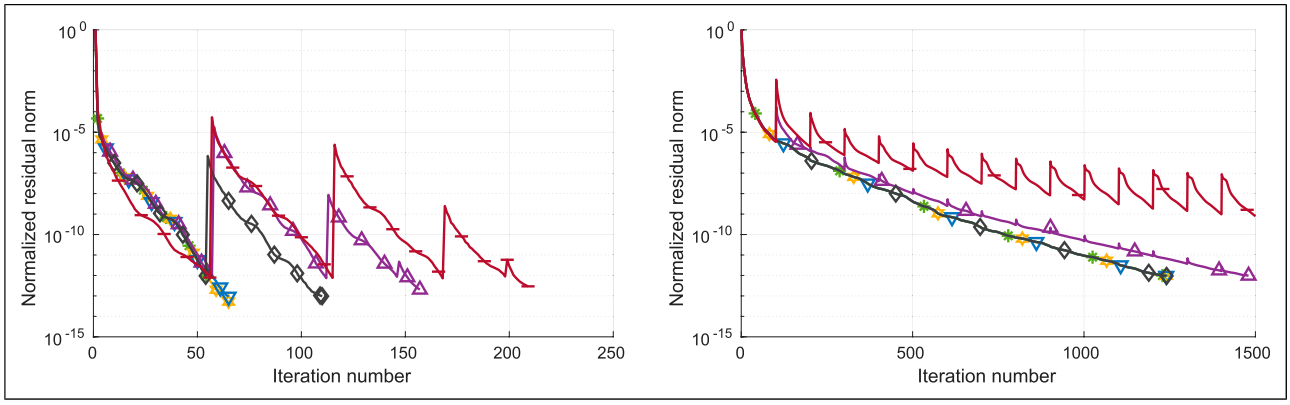

Figure 5. Detailed convergence analysis of the Jacobi-preconditioned CB-GMRES variants for the circuit5M_dc and Serena problems.

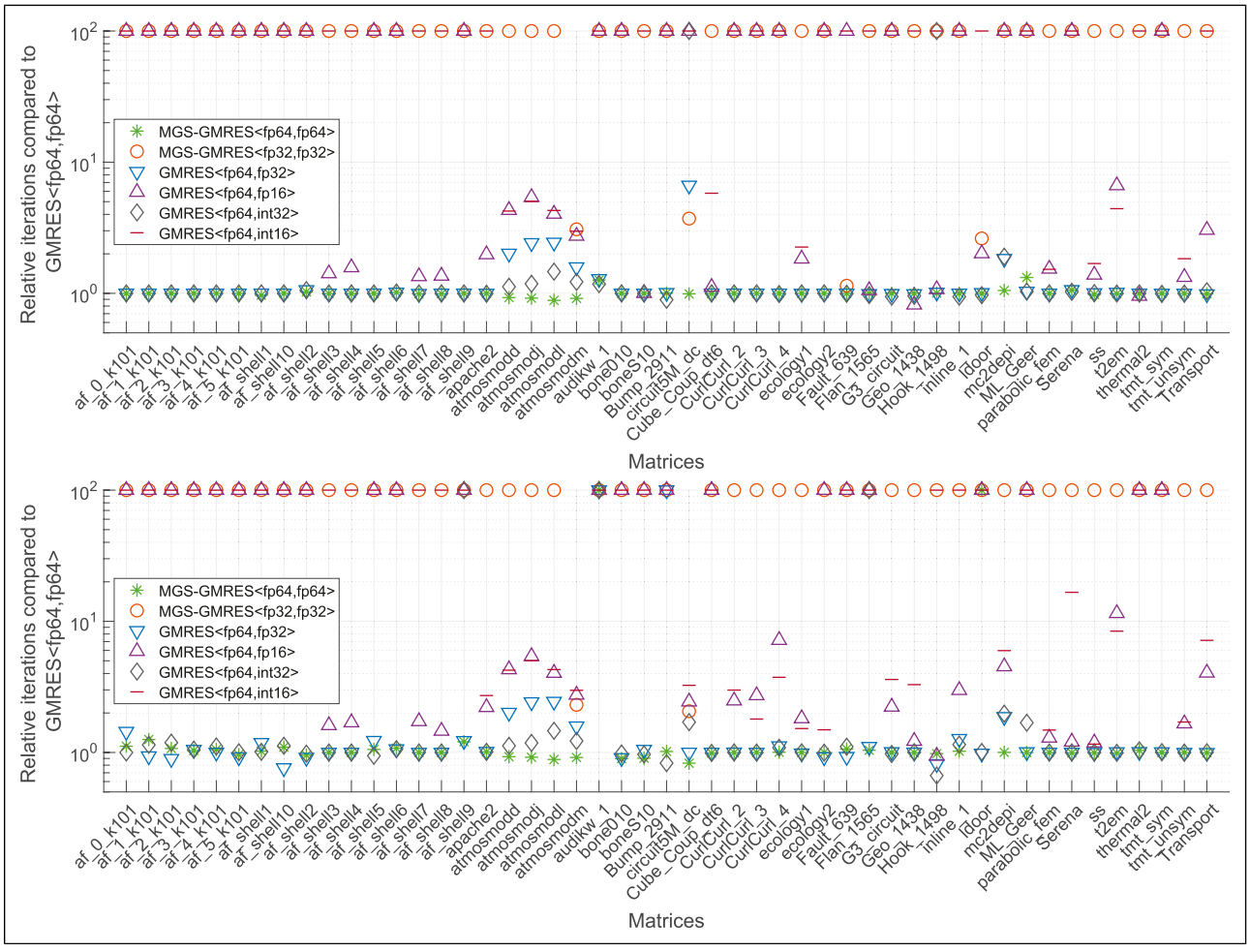

Figure 6. Iteration overhead of the CB-GMRES variants relative to the DP GMRES iteration count. The upper graph represents the results for the plain solver runs and the lower graph for the Jacobi-preconditioned solver runs.

the Jacobi-preconditioned solvers (bottom). If a solver does not succeed in reaching the same residual accuracy, we set the iteration overhead marker to "100" to clearly indicate that this solver is not a valid option.

This experiment shows that the CB-GMRES realizations GMRES<fp64,fp32> and GMRES<fp64,int32> generally match the iteration count of DP GMRES. Exceptions are the ATMOSMOD problems where the

CB-GMRES variants using 32-bit memory precision need a few additional iterations. In contrast, when the orthogonal basis is stored using the 16-bit formats, even if the residual accuracy can be achieved, the overhead often increases dramatically. We recall that the orthogonalization inside GMRES is based on a classical Gram–Schmidt equipped with re-orthogonalization (CGS-reortho). For convenience, we also include the iteration counts for a DP GMRES using modified Gram–Schmidt (MGS) orthogonalization. In contrast to CGS-reortho, MGS requires the use of less efficient BLAS-1 routines and therefore results in lower performance on highly parallel architectures. At the same time, the iteration counts reveal that the use of MGS rarely improves the convergence of the GMRES solver (see cases with iteration overhead smaller 1).

In the left-hand side plot in Figure 7, we report a few key statistics obtained from the experimental evaluation with the 49 test problems using either the non-preconditioned GMRES or the Jacobi-preconditioned GMRES. While storing the Krylov basis in fp32 or int32 generally incurs no iteration overhead, when using fp16 storage, we obtain a median iteration overhead of 1.5× with the 50%-quantiles reaching up to 2.5× and the 90%-quantiles reaching up to 5×. For int16, we obtain a median iteration overhead of 2.5× with the 50%-quantiles reaching up to 4× and the 90%-quantiles reaching up to 8×.

Performance of CB-GMRES

Even though we have now experimentally demonstrated that the CB-GMRES variants can compensate for the perturbations in the subspace via additional iterations (which is equivalent to extending the subspace by additional basis vectors), the resulting algorithms will only be useful in production if the associated iteration overhead is compensated by a runtime reduction coming from the decreased memory access volume. In the right-hand side plot in Figure 7, we show statistics on the performance improvements that CB-GMRES renders over DP GMRES when using different storage formats for the Krylov basis, again considering both a plain GMRES and its Jacobi-preconditioned variant. As could be expected from the large iteration overheads, storing the Krylov basis in int16 or fp16 usually results in a slow-down of the global solution process. Conversely, storing the Krylov basis in int32 or fp32 yields attractive performance improvements, with slight advantages for the GMRES<fp64,fp32> variant. As listed in Table 3, the median speed-up for GMRES<fp64,fp32> is 1.4×, with the 50%-quantiles reaching up to 1.5× and outliers reaching up to 1.75×. We note a few outliers that are likely related to rounding effects enabling faster or slower convergence than the DP GMRES.

In Figure 8, we provide a detailed performance evaluation by visualizing the speed-up for the distinct test problems, distinguishing between the non-preconditioned GMRES (top graph) and the Jacobi-preconditioned GMRES (bottom graph). Here, we notice a rather uniform picture concerning the speed-ups for GMRES<fp64,fp32> and GMRES<fp64,int32>, with the exception of the atmosmod problems where the iteration overhead cannot be compensated with faster memory access. Overall, GMRES<fp64,fp32> is slightly superior over GMRES<fp64,int32>, which is likely due to the overhead of the scaling process and the additional scaling factors needed when storing the basis vectors in int32.

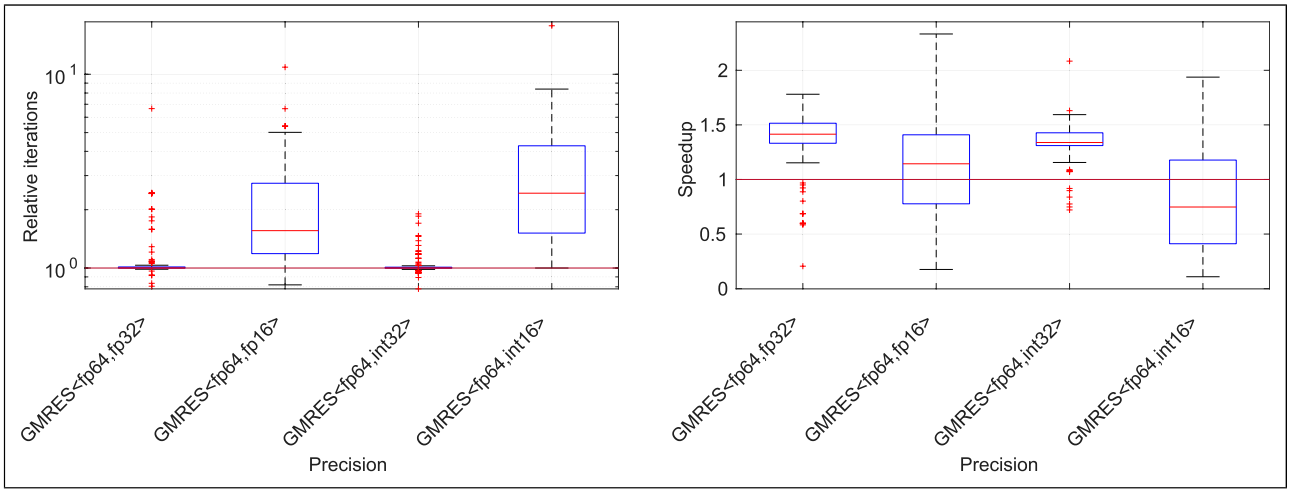

Figure 7. Statistics obtained from running the CB-GMRES algorithms on the 49 test problems. Left: iteration overhead (relative to DP GMRES) and right: speed-up relative to DP GMRES. The data reflects both the non-preconditioned runs and the Jacobi-preconditioned settings.

Table 3. Statistics for the CB-GMRES speed-up relative to the GMRES<fp64,fp64> baseline implementation on the test matrices listed in Table 1.

Solver	Arithmetic mean	Arithmetic median	Variance
GMRES<fp64,fp64>	1	1	0
GMRES<fp64,fp32>	1.36	1.41	0.09
GMRES<fp64,fp16>	1.04	0.98	0.26
GMRES<fp64,int32>	1.32	1.34	0.04
GMRES<fp64,int16>	0.67	0.55	0.16

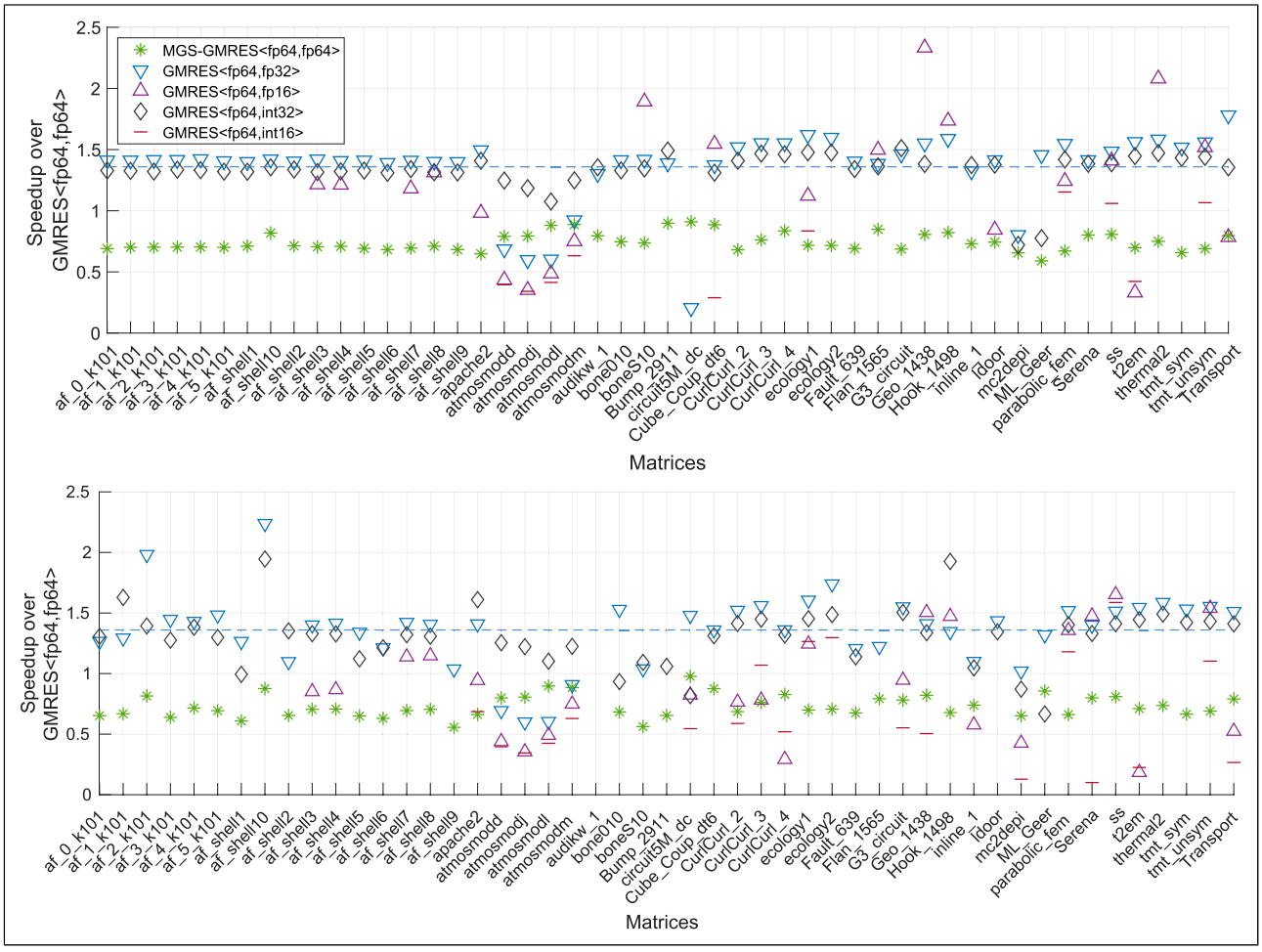

Figure 8. Speed-up of the non-preconditioned CB-GMRES (top) and Jacobi-preconditioned CB-GMRES (bottom) over the respective DP GMRES variants.

From this experiment, we conclude that the GMRES<fp64,fp32> is an appropriate choice for a wide range of problems. We also note that GMRES based on modified Gram–Schmidt orthogonalization (MGS-GMRES<fp64,fp64>) is consequently slower than GMRES based on classical Gram–Schmidt orthogonalization (GMRES<fp64,fp64>). This may be expected as MGS requires the use of less efficient BLAS-1 operations.

Varying the Krylov subspace dimension

When motivating the use of a more compact storage format to maintain the orthonormal vectors in an earlier section, we argued that the memory savings against DP GMRES grow with the dimension of the Krylov subspace m. In more detail, when ignoring numerical effects, we can expect that the speed-up asymptotically reaches the ratio between the storage format complexities: 4× when

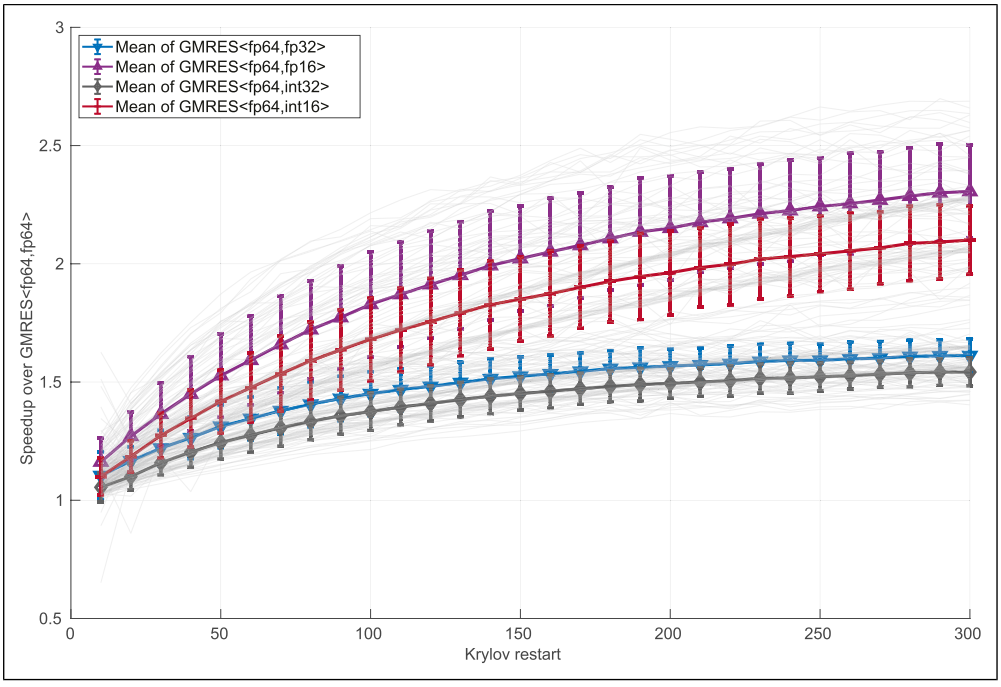

Figure 9. Speed-up for different CB-GMRES variants (GMRES<arith,mem>) over DP GMRES (GMRES<fp64,fp64>) for increasing restart values.

using GMRES<fp64,fp16> or GMRES<fp64,int16> and 2× when using GMRES<fp64,fp32> or GMRES<fp64,int32>. In Figure 9, we quantify those speed-ups experimentally, considering restart parameters in the range 10–300. We note that larger restart values are rarely employed as they come with high computational complexity and memory requirements typically exceeding the hardware capabilities. We emphasize that in this experiment, we ignore any numerical effects but only focus on the runtime needed to execute 10 restart cycles with different restart sizes. Also, even though we already identified the GMRES<fp64,fp32> as being superior in terms of convergence and performance, we include all CB variants in this analysis. In Figure 9, we employ gray lines to indicate the speed-up behavior for the distinct matrices and use boxplots to illustrate the statistics for the CB-GMRES variants. The results indicate that the average speed-ups for GMRES<fp64,fp32> or GMRES<fp64,int32> asymptotically approach a value below 2×, with the speed-ups being constantly higher for the former (which requires no scaling). The speed-up is smaller than 2× because the cost savings are limited to those obtained from the compressed storage of the orthogonal basis but not in other parts of the algorithm , for example, the SPMV kernel (see Amdahl's law). For GMRES<fp64,fp16> or GMRES<fp64,int16>, the speed-up values are larger, though below the 4× theoretical bound. Again, the scaling process and memory overhead make the GMRES<fp64,int16> speed-ups inferior to the GMRES<fp64,fp16> speed-ups.

Acknowledging that Figure 9 does not reflect the numerical effects that occur when using larger restart parameters, in Figure 10, we quantify the actual iteration overheads (left) and speed-ups (right) for the non-preconditioned and Jacobi-preconditioned CB-GMRES over the DP GMRES variants when increasing the restart value to $m = 300$. Compared to Figure 7, we notice a moderate growth in the iteration overhead and a more substantial increase of speed-up. The median speed-ups increase to 1.6× for GMRES<fp64,fp32> and 1.5× for GMRES<fp64,int32> over the DP GMRES baseline, respectively. The fact that these speed-ups match those expected from Figure 9 indicates that using 32-bit formats for storing the Krylov basis in CB-GMRES has only a negligible impact on the convergence behavior. Conversely, the speed-ups for GMRES<fp64,fp16> and GMRES<fp64,int16> do not fulfill the expectations, confirming that 16-bit formats are insufficient for storing the Krylov basis in CB-GMRES.

Combining CB-GMRES with incomplete factorization preconditioning

In the previous sections, we have analyzed the convergence and performance of CB-GMRES when being used as a stand-alone solver or a preconditioned method equipped

with a lightweight scalar Jacobi scheme. Some situations, however, require the use of a more sophisticated preconditioner to aggressively reduce the iteration count. One of the most popular and general preconditioners is the ILU(0) preconditioner, belonging to the class of incomplete LU factorization (ILU) preconditioners (Saad, 2003). The ILU preconditioners approximate the factorization of the system matrix, ignoring some of the fill-in that would arise in an exact factorization, and applying the preconditioner by solving the triangular systems coming from the incomplete factors (Saad, 2003). The most popular ILU preconditioner is ILU(0), which actually ignores all fill-in, therewith preserving the sparsity pattern of the system matrix in the incomplete factors. As both the ILU generation via a modified Gaussian elimination and the ILU application in terms of sparse triangular solves are inherently sequential, significant efforts are being spent on developing algorithms that are capable of leveraging the parallelism of GPU architectures (Chow et al., 2015 and Anzt et al., 2015). To acknowledge the goal of evaluating a high-performance realization of an ILU-preconditioned GMRES solver, we next consider the application of the (left) ILU preconditioner in terms of matrix–vector multiplications with inverse approximations of the incomplete factors. Specifically, in this experiment, the ILU preconditioner generation is handled via NVIDIA's cuSPARSE ILU, followed by the approximate inversion and subsequent application by the Incomplete Sparse Approximate Inverse (ISAI) algorithm

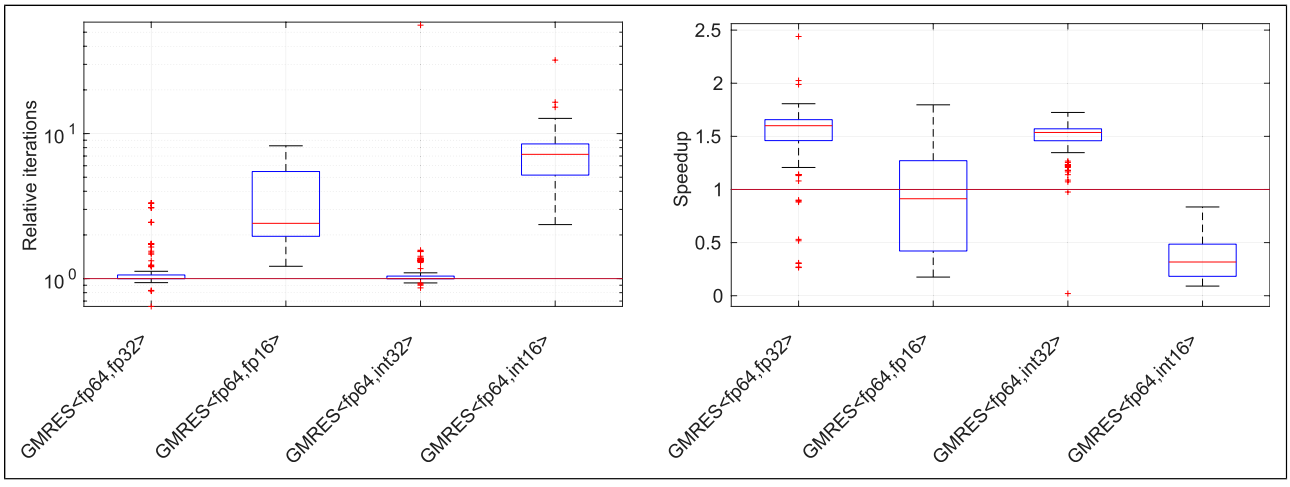

Figure 10. Statistics obtained from running the CB-GMRES algorithms on the 49 test problems. Left: iteration overhead (relative to DP GMRES) and right: speed-up relative to DP GMRES. The data reflects both the non-preconditioned runs and the Jacobi-preconditioned settings, all using the restart parameter $m = 300$.

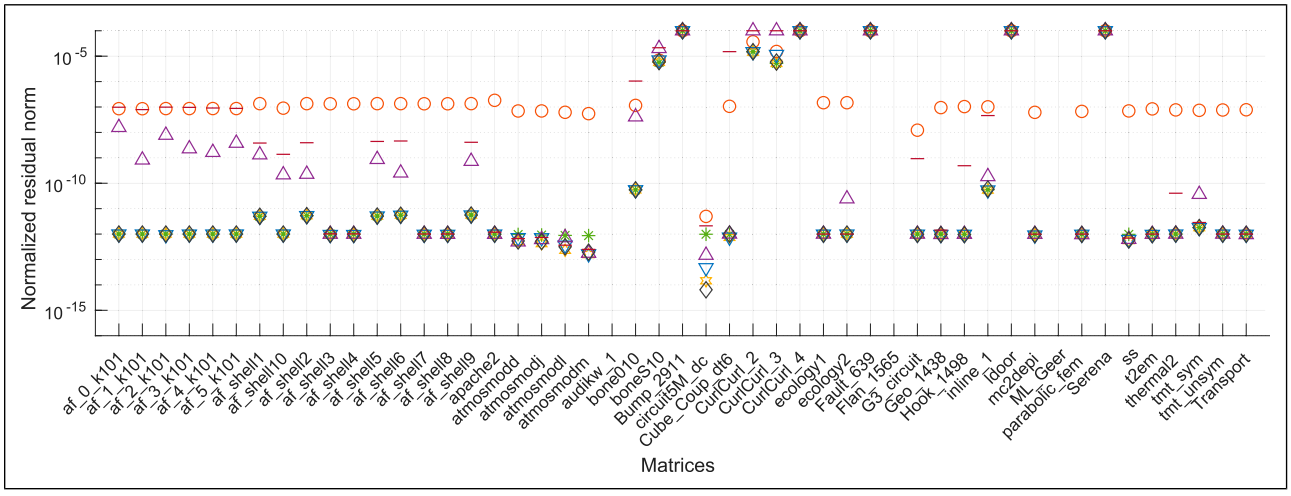

Figure 11. Normalized residual of ILU-preconditioned GMRES and ILU-preconditioned CB-GMRES using different precision for arithmetic and memory operations.

(Anzt et al., 2018), both available in the Ginkgo library. Similar to Jacobi, all sections of the ILU preconditioner compute and store the values in the arithmetic precision of the corresponding solver in order to isolate the Krylov basis storage precision as the only difference. We ignore test matrices where the generation of an ILU preconditioner or the ISAI triangular solver fails. Using this setup, in Figure 11, we report the attainable relative residual accuracy of the ILU-preconditioned CB-GMRES variants using a restart parameter $m = 100$ and a relative residual stopping criterion of 10^{-12}. As in the previous experiments, the ILU-preconditioned SP GMRES (MGS-GMRES<fp32,fp32>) is unable to provide the same accuracy as the DP GMRES (ILU-GMRES<fp64,fp64>). Conversely, the CB-GMRES using low precision only for storing the Krylov basis generally succeeds in providing the same accuracy if 32-bit storage is used (ILU-GMRES<fp64,fp32> and ILU-GMRES<fp64,int32>). Using 16-bit storage usually decreases the approximation accuracy, while often still providing higher accuracy than the SP GMRES. In Figure 12, we quantify the iteration overhead (top) and speed-up (bottom) that the ILU-preconditioned CB-GMRES achieves over the DP counterpart. Again, markers indicating a 100× iteration overhead represent solvers that were unable to achieve the same approximation accuracy. We highlight that the ILU-preconditioned ILU-GMRES<fp64,fp32> mostly exhibits a negligible iteration overhead over the ILU-preconditioned DP GMRES (top graph in Figure 12), resulting again in attractive speed-ups over the DP GMRES (bottom graph in Figure 12). Compared with the un-preconditioned and Jacobi-preconditioned solver configurations reported in Figure 8, the advantages of the ILU-GMRES<fp64,fp32> over the standard DP GMRES decrease when using an ILU preconditioner. This is expected as the ILU preconditioner accounts for a significant portion of the solver execution time, giving the CB-GMRES less room for improving the overall runtime.

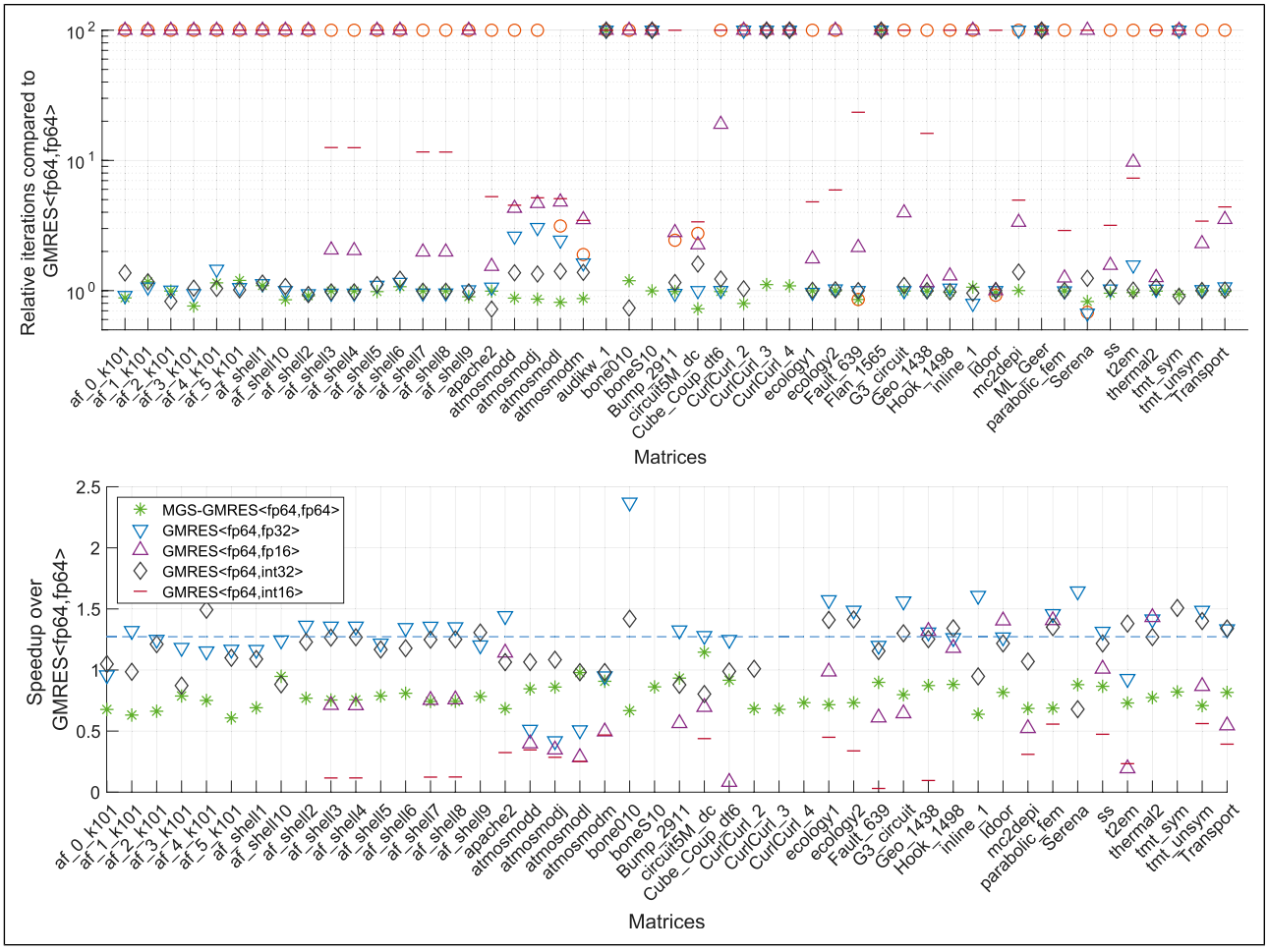

Figure 12. Iteration overhead (top) and speed-up (bottom) of the ILU-preconditioned CB-GMRES over the ILU-preconditioned DP GMRES.

Summary and outlook

We have introduced and evaluated a GMRES algorithm that maintains the Krylov basis in a compressed (compact) form, in order to reduce the traffic between memory and the processor arithmetic units, while performing all arithmetic in double-precision, to preserve the hardware-supported high precision arithmetic in all computations. The key to this approach lies in decoupling the memory storage format of the orthogonal basis from the arithmetic precision when operating with that basis. This general strategy was proposed and integrated into Ginkgo's library, in the form of a memory accessor, which is leveraged and applied in our work to the specific case of the GMRES algorithm.

We have integrated a high-performance realization of the CB-GMRES operating with 16-bit and 32-bit Krylov basis storage into the Ginkgo open source library. The performance evaluation of this solver on an NVIDIA V100 GPU demonstrates the practical advantages of the communication-reduction technique, which are aligned with the acceleration that could be expected from Amdahl's law. Using 32-bit storage for the Krylov basis, the algorithm achieves an average 1.6× speed-up over a standard double-precision GMRES. On the other hand, the 16-bit formats further reduce the communication volume, but they regularly fail to preserve the convergence characteristics of the GMRES solver. Overall, we believe that the proposed technique is useful as it tackles the memory wall problem that dominates the performance of many current processors-applications. Furthermore, its benefits are orthogonal and, therefore accumulative, to those that can be attained with other communication-reduction techniques applied, for example, to the preconditioner, the realization of SPMV, or the GMRES algorithm itself.

In future work, we will investigate whether compression techniques that are traditionally used to compress large datasets can pose an alternative to the use of 32-bit and 16-bit fixed- and floating-point formats for storing the compressed basis vectors.

Declaration of conflicting interests

The author(s) declared no potential conflicts of interest with respect to the research, authorship, and/or publication of this article

Funding

The author(s) disclosed receipt of the following financial support for the research, authorship, and/or publication of this article: José I. Aliaga and Andrés E. Tomás were supported by Project PID2020-113656RB-C21 funded by MCIN/AEI/ 10.13039/501100011033, whereas Enrique S. Quintana-Ortí was supported by Project PID2020-113656RB-C22 funded by MCIN/AEI/ 10.13039/501100011033. Hartwig Anzt and Thomas Grützmacher were supported by the "Impuls und Vernetzungsfond" of the Helmholtz Association under grant VH-NG-1241 and the US Exascale Computing Project (17-SC-20-SC), a collaborative effort of the U.S. Department of Energy Office of Science and the National Nuclear Security Administration.

ORCID iDs

Jose I. Aliaga https://orcid.org/0000-0001-8469-764X
Thomas Grutzmacher https://orcid.org/0000-0001-9346-2981
Andrés E Tomás https://orcid.org/0000-0003-3969-2174

Supplemental Material

Supplemental Material for this article is available online.

References

Anzt H, Chow E and Dongarra J (2015) Iterative sparse triangular solves for preconditioning. In: European conference on parallel processing, Berlin, Heidelberg, 2015, Springer, pp. 650–661.

Anzt H, Cojean T, Chen YC, et al. (2020a) Ginkgo: A high performance numerical linear algebra library. *Journal of Open Source Software* 5(52): 2260. DOI: 10.21105/joss.02260.

Anzt H, Cojean T, Flegar G, et al. (2020b) *Ginkgo: A Modern Linear Operator Algebra Framework for High Performance Computing*.

Anzt H, Cojean T and Grützmacher T (2021) *Technical Report: Design of the Accessor*. LLNL Report LLNL-SR-818775. DOI: 10.2172/1773264.

Anzt H, Cojean T, Yen-Chen C, et al. (2020c) Load-balancing sparse matrix vector product kernels on gpus. *ACM Trans. Parallel Comput* 7(1)–26. URL. DOI: 10.1145/3380930.

Anzt H, Dongarra J, Flegar G, et al. (2019a) Adaptive precision in block-Jacobi preconditioning for iterative sparse linear system solvers. *Concurrency and Computation: Practice and Experience* 31(6): e4460.

Anzt H, Flegar G, Grützmacher T, et al. (2019b) Toward a modular precision ecosystem for high-performance computing. *The International Journal of High Performance Computing Applications* 33(6): 1069–1078. URL. DOI: 10.1177/1094342019846547 https://doi.org/10.1177/1094342019846547.

Anzt H, Huckle TK, Bräckle J, et al. (2018) Incomplete sparse approximate inverses for parallel preconditioning. *Parallel Computing* 71: 1–22.

Björck Å (1967) Solving linear least squares problems by Gram-Schmidt orthogonalization. *BIT Numerical Mathematics* 7(1): 1–21.

Carson EC (2015) *Communication-avoiding Krylov Subspace Methods in Theory and Practice*. PhD Thesis. Berkeley: University of California.

Chow E, Anzt H and Dongarra J (2015) Asynchronous iterative algorithm for computing incomplete factorizations on gpus. In: *International Conference on High Performance Computing*, Cham: Springer, pp. 1–16.

Cools S (2019) Analyzing and improving maximal attainable accuracy in the communication hiding pipelined BiCGStab method. *Parallel Computing* 86: 16–35.

Davies T (2006) *Direct Methods for Sparse Linear Systems*. Philadelphia, PA, USA: Society for Industrial and Applied Mathematics.

Davis TA and Hu Y (2011) The university of florida sparse matrix collection. *ACM Transactions on Mathematical Software* 38: 1–25. DOI: 10.1145/2049662.2049663.

Golub GH and Loan CFV (1996) *Matrix Computations*. 3rd edition. Baltimore: The Johns Hopkins University Press.

Gratton S, Simon E, Titley-Peloquin D, et al. (2020) Exploiting variable precision in GMRES. *SIAM J. Sci. Comput. (to appear)*.

Greenbaum A (1989) Behavior of slightly perturbed Lanczos and conjugate-gradient recurrences. *Lin. Alg. Appl* 113: 7–63.

Greenbaum A (1997) Estimating the attainable accuracy of recursively computed residual methods. *SIAM J. Matrix Anal. Appl* 18(3): 535–551.

Grützmacher T, Anzt H and Quintana-Ortí ES (2021) Using ginkgo's memory accessor for improving the accuracy of memory-bound low precision blas. In: *Software - Practice and Experience*, pp. 1–18. DOI: 10.1002/spe.3041.

Higham NJ (2002) *Accuracy and Stability of Numerical Algorithms*. Second edition. Philadelphia, PA, USA: Society for Industrial and Applied Mathematics. ISBN 0-89871-521-0.

Hoemmen M (2010) *Communication-Avoiding Krylov Subspace Methods*. PhD Thesis. USA, p. AAI3413388.

Horowitz M (2014) Computing's energy problem (and what we can do about it). In: IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC), San Francisco, CA, USA, 09–13 February 2014, pp. 10–14. DOI: 10.1109/ISSCC.2014.6757323.

Meurant G and Strakoš Z (2006) The lanczos and conjugate gradient algorithms in finite precision arithmetic. *Acta Numerica* 15: 471–542. DOI: 10.1017/S096249290626001X.

NVIDIA Corp (2017) *Whitepaper: NVIDIA TESLA V100 GPU ARCHITECTURE*.

Paige CC (1980) Accuracy and effectiveness of the Lanczos algorithm for the symmetric eigenproblem. *Lin. Alg. Appl* 34: 235–258.

Paige CC, Rozložník M, Strakoš Z, et al. (2006) Modified Gram-Schmidt MGS, least squares, and backward stability of MGS-GMRES. *SIAM J. Matrix Anal. Appl* 28(1): 264–284.

Saad Y (2003) *Iterative Methods for Sparse Linear Systems*. 2nd edition. Philadelphia, PA, USA: Society for Industrial and Applied Mathematics.

Simoncini V and Szyld DB (2003) Theory of inexact Krylov subspace methods and applications to scientific computing. *SIAM J. Sci. Comput* 25(2): 454–477.

Tsai YM, Cojean T and Anzt H (2020) Sparse linear algebra on AMD and NVIDIA GPUs – the race is on. In: Sadayappan P, Chamberlain BL, Juckeland G, et al (eds), *High Performance Computing*. Cham: Springer International Publishing, pp. 309–327. ISBN 978-3-030-50743-5.

van den Eshof J and Sleijpen GL (2004) Inexact Krylov subspace methods for linear systems. *SIAM J. Matrix Anal. Appl* 26(1): 125–153.