Random Sketching to Enhance the Numerical Stability of Block Orthogonalization Algorithms for s-step GMRES

Ichitaro Yamazaki, Andrew J. Higgins, Erik G. Boman,
and Daniel B. Szyld

Report 25-04-25
April 2025

Department of Mathematics
Temple University
Philadelphia, PA 19122

This report is available in the World Wide Web at
`http://faculty.cst.temple.edu/~szyld`
and also at `arXiv:2503.16717`

Random Sketching to Enhance the Numerical Stability of Block Orthogonalization Algorithms for *s*-step GMRES

Ichitaro Yamazaki[1], Andrew J. Higgins[1], Erik G. Boman[1], Daniel B. Szyld[2]

Abstract

We integrate random sketching techniques into block orthogonalization schemes needed for *s*-step GMRES. The resulting, one-stage and two-stage, block orthogonalization schemes generate the basis vectors whose overall orthogonality error is bounded by machine precision as long as each of the corresponding block vectors are numerically full rank. We implement these randomized block orthogonalization schemes using standard distributed-memory linear algebra kernels for *s*-step GMRES available in the Trilinos software packages. Our performance results on the Perlmutter supercomputer (with four NVIDIA A100 GPUs per node) demonstrate that these randomized techniques can enhance the numerical stability of the orthogonalization and overall solver, without a significant increase in the execution time.

1. Introduction

Generalized Minimum Residual (GMRES) [1] is a popular subspace projection method for iteratively solving a large linear system of equations as it computes the approximate solution that minimizes the residual norm in the generated Krylov subspace. To compute the approxmate solution, GMRES generates the orthonormal basis vectors of its projection subspace based on two main computational kernels: 1) Sparse-Matrix Vector multiply (SpMV), typically combined with a preconditioner, and 2) orthogonalization.

Though GMRES is a robust iterative method for solving general linear systems, the performance of these two kernels can be limited by communication costs (e.g., the cost of moving data through the local memory hierarchy and between the MPI processes). For instance, on a distributed-memory computer, to orthogonalize a new basis vector at each iteration, GMRES requires global reduces among all the MPI processes and performs its local computation based on either BLAS-1 or BLAS-2 operations. Hence, though the breakdown of the iteration time depends on the target hardware architecture and problem properties (e.g., the sparsity structure of the matrix and the preconditioner being used), orthogonalization can become a significant part of the iteration time, especially when scalable implementations of SpMV and preconditioner are available.

To improve the performance of the orthogonalization and of GMRES, communication-avoiding (CA) variants of GMRES [2, 3], based on *s*-step methods [4, 5], have been proposed. These variants generate a set of *s* basis vectors at a time, utilizing two computational kernels: 1) the Matrix Powers Kernel (MPK) to generate the set of *s* Krylov vectors by applying SpMV and preconditioner *s* times, followed by 2) the Block Orthogonalization Kernel that orthogonalizes the set of $s+1$ basis vectors at once. This provides the potential to reduce the communication cost of generating the *s* basis vectors by a factor of *s* (requiring the global reduce only at every *s* step and using BLAS-3 for performing most of the local computation). This is a very attractive feature, especially on currently available GPU clusters, where communication can be significantly more expensive compared to computation.

Since the potential speedup from the block orthogonalization is limited by the small step size *s* required to maintain the numerical stability of MPK, a two-stage variant of block orthogonalization was proposed [6]. In order to maintain the well-conditioning of the basis vectors, the first stage of this orthogonalization scheme pre-processes the block of $s+1$ basis, while the full orthogonalization is delayed until enough number of basis vectors, $\widehat{s}+1$, are generated to obtain the higher performance. This improves the performance of the block orthogonalization process while using the small step size *s*.

Block orthogonalization consists of 1) *inter* block orthogonalization to orthogonalize a new block of vectors against the already-orthogonalized blocks of vectors and 2) *intra* block orthogonalization to orthogonalize among the vectors within the new block. For the inter-block orthogonalization, Block Classical Gram-Schmidt with re-orthogonalization (BCGS twice, or BCGS2) obtains good performance on current hardware architectures because it is based entirely on BLAS-3. For robustness and performance of the overall block orthogonalization, the critical component is the algorithm used for the first intra-block orthogonalization [7]. In this paper, we consider the use of CholQR [8] twice (CholQR2) as our intra-block orthogonalization, which is based mainly on BLAS-3. Unfortunately, though the above combinations of the algorithms performs well on current hardware, *s*-step basis vectors can be ill-conditioned, and CholQR2 can fail when the condition number of $s+1$ vectors is greater than the reciprocal of the square-root of machine epsilon.

To enhance the numerical stability of the above block or-

[1]Sandia National Laboratories, Albuquerque, New Mexico, U.S.A
[2]Temple University, Philadelphia, Pennsylvania, U.S.A

thogonalization schemes and of the overall s-step GMRES solver, we integrate random sketching techniques. Theoretical studies of such randomized schemes for the intra-block orthogonalization have been established in two recent papers [9, 10]. We extend these studies to develop randomized BCGS2 schemes that generate the blocks of basis vectors whose overall orthogonality errors are bounded by machine epsilon. We have initially presented our preliminary results of the current paper at the SIAM Conference on Parallel Processing for Scientific Computing (SIAM PP), 2024 [11].

Our main contributions are:

- We integrate random sketching techniques into BCGS2 such that overall orthogonalization error is on the order of machine precision in both one-stage and two-stage frameworks as long as each of the corresponding block of $s+1$ and $\widehat{s}+1$ vectors are numerically full-rank, respectively.
- We present numerical results to demonstrate the improved numerical stability using random sketching techniques (to pre-process the basis vectors) compared to the state-of-the-art deterministic algorithms (BCGS2 with CholQR2).
- We implement Gaussian and Count sketching, and its combination, Count-Gauss sketching [12], for the s-step GMRES in Trilinos [13], which is a collection of open-source software packages for developing large-scale scientific and engineering simulation codes. Trilinos software stack allows the solvers, like s-step GMRES, to be portable to different computer architectures, using a single code base. In particular, our implementation of the random-sketching is based solely on standard distributed-memory linear algebra kernels (GEMM and SpMM), which are readily available in vendor-optimized libraries.
- We study the performance of the block orthogonalization and s-step GMRES on the Perlmutter supercomputer at National Energy Research Scientific Computing (NERSC) center. Our performance results on up to 64 NVIDIA A100 GPUs show that random sketching has virtually no overhead to enhance the numerical stability of the one-stage BCGS2. Although it has a higher overhead due to the larger sketch size required for the two-stage algorithm, the overhead became less significant as we increased the number of MPI processes. For example, the overhead was about 1.49× on 1 node, while it was about 1.19× on 16 nodes.

Table 1 lists the notation used in this paper. In addition, we use $Q_{\ell:t}$ to denote the blocks column vectors of Q with the block column indexes ℓ to t, while $q_{k:s}$ is the set of vectors with the column indexes k to s. We then use the bold small letter $\mathbf{v}_j$ to denote the sketched version of the block vector V_j, e.g., $\mathbf{v}_j = \Theta^T V_j$. Finally, $[Q, V]$ is the column concatenation of Q and V. For our numerical analysis, we use $c_k(\epsilon, n, s)$ to represent a scalar constant that is in the order of the machine epsilon ϵ but also depends on the matrix dimensions n and s.

notation	description
n	problem size
m	subspace dimension
s	step size (for the first stage)
$\widehat{s}$	second step size (for the second stage and $s \le \widehat{s} \le m$)
$v_k^{(j)}$	kth basis vector within the j-th s basis vectors
V_j	jth s-step basis vectors including the starting vector, i.e., a set of $s+1$ vectors generated by MPK $V_j = [v_{s(j-1)+1}, v_{s(j-1)+2}, \ldots, v_{sj+1}]$ and $V_0 = [v_0]$ to simplify the notation
$\underline{V}_j$	same as V_j except excluding the last vector, which is the first vector of V_{j+1}, i.e., a set of s vectors $\underline{V}_j = [v_{s(j-1)+1}, v_{s(j-1)+2}, \ldots, v_{sj}]$
$\widehat{V}_j$	V_j after the first inter-block orthogonalization
$\widehat{Q}_j$	V_j after the pre-processing stage
Q_j	orthogonal basis vectors of V_j
Θ	sketch matrix
$\mathbf{v}_j$	sketched version of the block vector V_j (i.e., $\mathbf{v}_j := \Theta^T V_j$)
ϵ	machine epsilon
$\kappa(V_j)$	condition number of V_j

Table 1: Notation used in the paper.

2. Related Work and Our Motivations

In recent years, random sketching has been used to improve the performance of Krylov solvers [14, 9, 15, 16]. In contrast to the previous works that have focused on "pseudo-optimal" GMRES, generating the basis vectors that are orthonormal with respect to the sketched inner-product, we use the random sketching to generate the well-conditioned basis vectors, but then explicitly generate the ℓ_2-orthonormal basis vectors.

- One reason for this is that except for one special case (i.e., two-stage with $\widehat{s} = m$, where m is the restart length), we sketch only a part of the Krylov subspace (e.g., each panel or big panel of $s+1$ or $\widehat{s}+1$ basis vectors, respectively), requiring the ℓ_2-orthogonality to ensure the overall consistency of the basis vectors over the restart loop.
- In addition, though the sketched norm is expected to be close to the original norm, they could deviate from the ℓ_2-norm in practice [16]. Though generating the ℓ_2-orthonormal basis vectors requires additional cost (both in term of computation and communication), we expect the convergence of our implementation of sketched s-step GMRES to be the same as the original s-step GMRES, which is useful in practice.

Communication-avoiding (CA) variants of the tall-skinny Householder QR algorithm have been proposed [17] and its superior performance over the standard algorithm has been demonstrated [18]. In this paper, we focus on the performance comparison of the randomized algorithm against CholQR-like algorithms. Although with a careful implementation, CA Householder may obtain the performance close to CholQR, we believe CholQR, which is mostly based on standard BLAS-3 operation, as the baseline performance is beneficial. In addition, for s-step GMRES, it is convenient to explicitly generate the orthogonal basis vectors, where the CA variants require non-negligible performance overhead.

The GPU performance of random sketching has been previously studied [19, 20]. This paper differs from these previous works since we focus on the algorithmic development of numerically stable block orthogonalization schemes. We then integrate the resulting randomized block orthogonalization schemes into s-step GMRES, and study their performance impact on a GPU cluster. We focus on the practical implementations of random sketching, using readily-available standard linear algebra kernels, though specialized kernels may improve the performance.

Some of the sparse sketching techniques have the potential to reduce the computational complexity of the orthogonalization process. Unfortunately, we have not seen this performance benefit in our performance experiments, using the standard linear algebra kernels. Nervelessness, our main focus is to enhance the numerical stability of the solver by integrating the random sketching techniques. Specifically, we develop randomized block orthogonalization algorithms that obtain $O(\epsilon)$ orthogonality errors, given the corresponding block vectors are numerically full-rank.

Since random sketching enhances the stability, it may be possible to use a larger step size for some matrices. However, it is often not feasible to tune the step size for each problem on a specific hardware (the largest step size to maintain the stability of MPK). Hence, we focus on improving the stability of the solver with the current default setup (i.e., allowing us to solve the problem, where the original algorithm failed), and study the required performance overhead. However, we will also discuss the computational and communication complexities of each algorithm in Section 8.

3. Background

By $\Theta \in \mathbb{R}^{n\times\widehat{m}}$, we denote a *random sketch matrix*, with $\widehat{m} \ll n$. The general concept of "random sketching" is to apply this random sketch matrix Θ (generated using specific probability distributions) to a large matrix V to obtain a "sketched" matrix $\mathbf{v} = \Theta^T V$ that greatly reduces the row dimension of V while preserving its fundamental properties, such as its norm and singular values, as much as possible. In particular, the sketch matrix Θ is typically chosen to be a *subspace embedding*, or a linear map to a lower dimensional space that preserves the ℓ_2-inner product of all vectors within the subspace up to a factor of $\sqrt{1 \pm \mu}$ for some $\mu \in (0, 1)$ [21, 22]. Such embeddings also preserve ℓ_2-norms in a similar way [9].

Definition 3.1 (μ-subspace embedding)**.** *Given $\mu \in (0, 1)$, the sketch matrix $\Theta \in \mathbb{R}^{n\times\widehat{m}}$ is a* μ-subspace embedding *for the subspace $\mathcal{V} \subset \mathbb{R}^n$ if $\forall x, y \in \mathcal{V}$,*

$$|\langle x, y\rangle - \langle \Theta^T x, \Theta^T y\rangle| \leq \mu \|x\|_2 \|y\|_2. \tag{1}$$

Equation (1) provides a straightforward relation between the sketching matrix and the preservation of the ℓ_2-norm.

Corollary 3.1.1. *If the sketch matrix $\Theta \in \mathbb{R}^{n\times\widehat{m}}$ is a* μ-subspace embedding *for the subspace $\mathcal{V} \subset \mathbb{R}^n$, then $\forall x \in \mathcal{V}$,*

$$\sqrt{1-\mu}\, \|x\|_2 \leq \|\Theta^T x\|_2 \leq \sqrt{1+\mu}\, \|x\|_2. \tag{2}$$

Corollary 3.1.1 implies that we can also bound the singular values of a matrix V by those of the sketched matrix $\Theta^T V$. Hence if $\Theta^T V$ is well conditioned, then so is V.

Corollary 3.1.2. *If the sketch matrix $\Theta \in \mathbb{R}^{n\times\widehat{m}}$ is a* μ-subspace embedding *for the subspace $\mathcal{V} \subset \mathbb{R}^n$, and V is a matrix whose columns form a basis of $\mathcal{V}$, then*

$$\begin{aligned}(1+\mu)^{-1/2}\, \sigma_{min}(\Theta^T V) \leq \sigma_{min}(V) \leq \sigma_{max}(V) \\ \leq (1-\mu)^{-1/2}\, \sigma_{max}(\Theta^T V).\end{aligned} \tag{3}$$

Thus,

$$\kappa(V) \leq \sqrt{\frac{1-\mu}{1+\mu}}\, \kappa(\Theta^T V). \tag{4}$$

Proofs for Corollary 3.1.1 and 3.1.2 can be found in [9].

The limitation of μ-subspace embedding presented in Definition 3.1 is that to ensure that the sketch matrix approximately preserves norms and inner products, one needs to know the subspace $\mathcal{V} \subset \mathbb{R}^n$ a priori. In contrast, to use sketching techniques in Krylov subspace methods efficiently, we need a sketch matrix that does not require complete prior knowledge of the subspace, since Krylov subspaces are generated as the algorithm iterates. This can be accomplished by using $(\mu, \delta, \widehat{s})$ *oblivious ℓ_2-subspace embeddings* [9].

Definition 3.2 ($(\mu, \delta, \widehat{s})$ oblivious ℓ_2-subspace embedding)**.** *The sketch matrix $\Theta \in \mathbb{R}^{n\times\widehat{m}}$ is a* $(\mu, \delta, \widehat{s})$ oblivious ℓ_2-subspace embedding *if it is a μ-subspace embedding for any fixed $\widehat{s}$-dimensional subspace $\mathcal{V} \subset \mathbb{R}^n$ with probability at least $1 - \delta$.*

One concrete example of a $(\mu, \delta, \widehat{s})$ oblivious ℓ_2-subspace embedding is $\Theta = \frac{1}{\sqrt{\widehat{m}}} G$ where $G \in \mathbb{R}^{n\times\widehat{m}}$ is a Gaussian matrix and the sketch size is given by $\widehat{m} = \Omega(\mu^{-2}\widehat{s})$ [23]. In practice, the sketch size may be chosen as $\widehat{m} \approx \widehat{s}/\mu^2$ [21]. This relation allows a simple correspondence between the sketch size (or equivalenty the embedding dimension) $\widehat{m}$ and the subspace dimension $\widehat{s}$ for a given μ. For instance, to achieve $\mu = 1/\sqrt{2}$, the sketch size of $\widehat{m} \approx 2\widehat{s}$ is sufficient, though in principle, one could choose a different value of μ to construct a different sketch size. Other $(\mu, \delta, \widehat{s})$ oblivious ℓ_2-subspace embeddings exist that can be stored in a sparse format, including sub-sampled randomized Hadamard and Fourier transforms (SRHT and SRFT, respectively), and "sparse dimension reduction maps" [9, 21].

4. Block Orthogonalization for s-step GMRES

Figure 1 shows the pseudocode of s-step GMRES for solving a linear system $Ax = b$ with a preconditioner M^{-1}, which has been also implemented in Trilinos software framework [24, 13]. Though we focus on monomial basis vectors in this paper, Trilinos also has an option to generate Newton basis [25] to improve the numerical stability of the basis vectors V_j generated by the "matrix-powers kernel" (MPK).

Compared to the standard GMRES, s-step GMRES has the potential to reduce the communication cost of generating the s

Input: coefficient matrix A, right-hand-side vector b, initial vector x, and appropriately-chosen "change-of-basis-matrix" T (see [3, Section 3.2.3] for details)
Output: approximate solution x

1: $r = b - Ax$,
2: $\gamma = \|r\|_2$
3: **while** not converged **do**
4: $q_1 = r/\gamma$, and $h_{1,1} = 0$
5: **for** $j = 1 : m/s$ **do**
6: *// Matrix Powers Kernel to generate new s vectors*
7: $v_1^{(j)} := q_{s+1}^{(j-1)}$
8: **for** $k = 1 : s$ **do**
9: $v_{k+1}^{(j)} = AM^{-1}v_k^{(j)}$
10: **end for**
11: *// Block orthogonalization of $s+1$ basis vectors*
12: $[Q_j, R_j] := \text{BlkOrth}(Q_{1:(j-1)}, V_j)$
13: **end for**
14: *// Generate the Hessenberg matrix such that $AQ = QH$*
15: $H_{1:m+1,1:m} = R_{1:m+1,1:m+1}TR^{-1}_{1:m,1:m}$
16: *// Compute approximate solution with minimum residual*
17: $\hat{y} = \arg\min_{y \in Q_{1:m+1}} \|\gamma e_1 - H_{1:m+1,1:m}y\|_2$
18: $x = x + V_m\hat{y}$
19: $r = b - Ax$
20: $\gamma = \|r\|_2$
21: **end while**

Figure 1: Pseudocode of s-step GMRES where $[Q_j, R_j] = \text{qr}(Q, V_j)$ extends the QR factorization such that $QR = V$ with $Q^TQ = I$ and upper-triangular R with non-negative diagonals

basis vectors by a factor of s, where standard GMRES is essentially s-step GMRES with the step size of one. For instance, to apply SpMV s times (Lines 7 to 9 of the pseudocode), several CA variants of MPK exist [26]. On a distributed-memory computer, CA variants may reduce the communication latency cost of SpMV, associated with the point-to-point neighborhood Halo exchange of the input vector, by a factor of s (though it requires additional memory and local computation, and it may also increase the total communication volume). However, while in practice, SpMV is typically combined with a preconditioner to accelerate the convergence rate of GMRES, only a few CA preconditioners of specific types have been proposed [27, 28]. To support a wide-range of preconditioners used by applications, instead of CA MPK, Trilinos s-step GMRES uses a standard MPK (applying each SpMV with neighborhood communication in sequence), and focuses on improving the performance of block orthogonalization. Also, avoiding the global communication may lead to a greater gain on the orthogonalization performance than CA MPK does on SpMV performance.

There have been significant advances in the theoretical understanding of s-step Krylov methods [2]. However, though the orthogonality error bound to obtain the backward stability of GMRES has been established [29], to the authors knowledge, there are no known theoretical bounds on the orthogonality errors, which are required to obtain the maximum attainable accuracy of s-step GMRES. Hence, in this paper, we focus on block orthogonalization schemes that can maintain the overall $O(\epsilon)$ orthogonality error of the generated orthonormal block basis vectors $Q_{1:j}$, where ϵ is the machine precision:

$$\|I - Q_{1:j}^TQ_{1:j}\| \quad = \quad O(\epsilon). \tag{5}$$

Input: $Q_{1:j-1}$ and V_j
Output: Q_j and $R_{1:j-1,j}$

1: *// Orthogonalize V_j against $Q_{1:j-1}$*
2: $R_{1:j-1,j} := Q_{1:j-1}^TV_j$
3: $Q_j := V_j - Q_{1:j-1}R_{1:j-1,j}$

(a) BCGS Inter-block orthogonalization.

Input: V_j
Output: Q_j, $R_{j,j}$

1: *// Form Gram matrix*
2: $G = V_j^TV_j$
3: *// Compute its Cholesky factorization to generate R*
4: $R_{j,j} = \text{chol}(G)$
5: *// Generate orthonormal Q*
6: $Q_j := V_jR_{j,j}^{-1}$

(b) CholQR Intra-block orthogonalization.

Input: $Q_{1:j-1}$ and V_j
Output: Q_j and $R_{1:j-1,j}$

1: *// BCGS orthogonalization*
2: $[\widehat{V}_j, \widehat{R}_{1:j-1,j}] := \text{BCGS}(Q_{1:j-1}, V_j)$
3: $[\widehat{Q}_j, \widehat{R}_{j,j}] := \text{IntraBlk}(\widehat{V}_j)$
4: **if** $j == 1$ **then**
5: $Q_j := \widehat{Q}_j$
6: **else**
7: *// BCGS re-orthogonalization*
8: $[Q_j, T_{1:j-1,j}] := \text{BCGS}(Q_{1:j-1}, \widehat{Q}_j)$
9: $[Q_j, T_{j,j}] := \text{CholQR}(Q_j)$
10: *// Update upper-triangular matrix*
11: $R_{1:j-1,j} := R_{1:j-1,j} + T_{1:j-1,j}\widehat{R}_{j,j}$
12: $R_{j,j} := T_{j,j}\widehat{R}_{j,j}$
13: **end if**

(c) BCGS twice (BCGS2).

Figure 2: Block Classical Gram-Schmidt (BCGS) to orthogonalize V_j against the orthonormal vectors $Q_{1:j-1}$, where "chol(G)" returns the upper-triangular Cholesky factor of G.

The block orthogonalization algorithm consists of two steps: the inter- and intra-block orthogonalization to orthogonalize the new set of $s+1$ basis vectors against the previous vectors and among themselves, respectively. To maintain orthogonality, in practice, both steps are applied with re-orthogonalization.

There are several combinations of the inter- and intra-block orthogonalization algorithms [30], but in this paper, we focus on the block-orthogonalization process that uses Block Classical Gram-Schmidt (BCGS) both for the first inter-block orthogonalization and for the re-orthogonalization, and uses Cholesky QR (CholQR) factorization [8] for the intra-block re-orthogonalization. To ensure the stability and performance of the overall block orthgonalization, the remaining critical component is the first intra-block orthogonalization scheme [7], which is the focus of this paper. Beside this first intra-block orthogonalization, such a block orthogonalization can be implemented using mostly BLAS-3 operations and needs only three global reduces. As a result, it performs well on current hardware architectures. The pseudocode of this block orthogonalization process is shown in Figure 2c.

In [7], it has been shown that in order to ensure the $O(\epsilon)$ orthogonality error of all the basis vectors $Q_{1:j}$, the first intra-orthogonalization algorithm (Line 3 of Figure 2c) needs to gen-

erate $\widehat{Q}_j$ such that

$$\|I - \widehat{Q}_j^T \widehat{Q}_j\| \quad = \quad O(\epsilon). \tag{6}$$

In the next section, we explore two algorithms, which performs well on the current hardware architectures and achieves (6), for the first intra-block orthogonalization. We discuss the condition on each block vector (i.e., the required condition number $\kappa(\widehat{V}_j)$ of the input basis vectors $\widehat{V}_j$) that is sufficient in order for each algorithm to guarantee (6).

5. Intra-Block Orthogonalization Algorithms

5.1. CholQR twice (CholQR2)

For the first intra-block orthogonalization, we first explore the use of CholQR [8] twice (CholQR2). As we can see in Figure 2b, CholQR can be implemented mostly based on BLAS-3 and requires only one synchronization.

In [6, 8, 31, Theorem IV.1], it has been shown that the orthogonality error of Q_j generated by CholQR is bounded as

$$\|I - Q_j^T Q_j\| = O(\epsilon)\kappa(V_j)^2, \tag{7}$$

when the following condition is satisfied:

$$c_1(\epsilon, n, s)\kappa(\widehat{V}_j)^2 < 1/2, \tag{8}$$

where $c_1(\epsilon, n, s)$ is a constant. Hence, BCGS2 with CholQR2 obtains the overall $O(\epsilon)$ orthogonality error as formalized in the following proposition.

Proposition 5.1. *When the condition* (8) *is satisfied, the orthgonality error of the basis vectors $\widehat{Q}_j$ computed by CholQR2, and hence of the basis vectors $Q_{1:j}$ generated by BCGS2 with CholQR2, is of the order of the machine precision (as the condition* (6) *ensures the condition* (5)*).*

Proof. The proof of Proposition 5.1 follows by combining the results from two facts: first, BCGS2 attains $O(\epsilon)$ orthogonality error overall provided each "IntraBlk" step on Line 3 of Figure 2c attains $O(\epsilon)$ orthogonality error [7], and CholQR2 attains $O(\epsilon)$ orthogonality error when (8) is satisfied [32]. □

The main drawback of CholQR is that it computes the Gram matrix of the input basis vectors V_j to be orthogonalized (Line 2 in Figure 2b), and the Gram matrix has the condition number which is the square of the input vectors' condition number. Hence, CholQR can fail when the condition number of the input vectors V_j is greater than the reciprocal of the square-root of the machine epsilon ϵ (i.e., $\kappa(V_j) > 1/O(\epsilon^{1/2})$). This can cause numerical issues, especially for the s-step method, because even when Newton or Chebyshev basis are generated, the s-step basis vectors can be ill-conditioned with a large condition number.

To alleviate this potential numerical instability, Trilinos implements a "recursive" variant of CholQR, as shown in Figure 3; when Cholesky factorization of the Gram matrix fails at the kth step due to a non-positive diagonal, it orthogonalizes just the first $k-1$ vectors by CholQR. It then orthogonalizes the remaining vectors against the first $k-1$ (roughly) orthonormal

Input: $\widehat{V} \in \mathbb{R}^{n\times s+1}$
Output: $Q \in \mathbb{R}^{n\times s+1}$, $R \in \mathbb{R}^{s+1\times s+1}$
1: *// Form Gram matrix*
2: $G = \widehat{V}^T \widehat{V}$
3: *// Compute its Cholesky factorization of G*
4: $R = \text{chol}(G)$
5: **if** Cholesky factorization failed at kth step **then**
6: **if** $k == 1$ **then**
7: Throw away the rest.
8: **else**
9: *// Orthogonalize $\widehat{V}_{1:k-1}$ and $V_{k:s+1}$ against $Q_{1:k-1}$*
10: $Q_{1:k-1} := \widehat{V}_{1:k-1} R^{-1}_{1:k-1,1:k-1}$
11: $\widehat{V}_{k:s+1} := \widehat{V}_{k:s+1} - Q_{1:k-1} R_{1:k-1,k:s+1}$
12: *// Recursively call CholQR on $V_{k:s+1}$*
13: $[Q_{k:s+1}, R_{k:s+1,k:s+1}] = \text{CholQR}(\widehat{V}_{k:s+1})$
14: **end if**
15: **else**
16: *// Orthogonalize V*
17: $Q := \widehat{V} R^{-1}$
18: **end if**

Figure 3: Recursive Cholesky QR (CholQR) to orthonormalize a set of vectors $V \in \mathbb{R}^{n\times s}$, where "chol($G$) returns the upper-triangular Cholesky factor of the Gram matrix G.

vectors by BCGS and recursively calls CholQR on the remaining vectors. This avoids the algorithmic breakdown of the orthogonalization process by adaptively adjusting the block size to orthogonalize the $s+1$ basis vectors. To orthogonalize the remaining vectors against the first $k-1$ vectors, it uses the partial Cholesky factors, and hence no additional overhead is needed. However, it needs to re-compute the dot-products of the remaining vectors, and may require multiple global reduces for ill-conditioned basis vectors. Furthermore, though it is often effective in recovering from the failures in combination with MPK, there is no bound on the orthogonality error with this recursive variant.

5.2. Randomized-Householder CholQR (RandCholQR)

Since MPK can generate ill-conditioned basis vectors, the requirement (8) can be too restrictive. To enhance the numerical robustness, we integrate the random-sketching techniques.

Instead of forming the Gram matrix of the basis vectors, which is the main cause of the numerical instability, Randomized CholQR (RandCholQR) first computes the random sketch $\widehat{\mathbf{v}}_j$ of the basis vectors $\widehat{V}_j$. It then generates their well-conditioned basis vectors $\widehat{Q}_j$ using the upper-triangular matrix computed by the stable QR factorization of the sketched vectors $\widehat{\mathbf{v}}_j$. When the input vectors $\widehat{V}_j$ are numerically full-rank and the sketch matrix Θ is a (μ, δ, s)-oblivious ℓ_2-subspace embedding (see Definition 3.2), it can be shown that the generated basis vectors $\widehat{Q}_j$ has the condition number of $O(1)$ [14, 33]. Hence, according to (7), as we call CholQR on $\widehat{Q}_j$, the resulting vector Q_j has an $O(\epsilon)$ orthogonality error. Figure 4b shows the pseudocode of the resulting RandCholQR algorithm for the intra-block orthogonalization.

The sketching typically requires one synchronization, and as a result, we expect RandCholQR to perform similarly to CholQR2. The actual performance of the algorithm depends

Input: V_j
Output: $\widehat{V}_j$, $R_{j,j}$
1: *// Sketch the new "panel"*
2: $\mathbf{v_j} := \Theta^T V_j$
3: *// Orthogonalize new* **sketched** *panel*
4: $[\mathbf{q}_j, R_{j,j}] := \text{HH}(\mathbf{v}_j)$
5: *// Generate well-conditioned basis*
6: $\widehat{V}_j := V_j R_{j,j}^{-1}$

(a) Randomized Householder QR (RandHH)

Input: V_j
Output: Q_j, $R_{j,j}$
1: *// RandQR on the new "panel"*
2: $[\widehat{V}_j, R_{j,j}] := \text{randHH}(V_j)$

3: *// Orthogonalize well-conditioned basis*
4: $[Q_j, T_{j,j}] := \text{CholQR}(\widehat{V}_j)$
5: $R_{j,j} := T_{j,j} R_{j,j}$

(b) Randomized Householder CholQR (RandCholQR)

Figure 4: Randomized QR algorithm, where HH(V) returns the orthogonal basis vectors Q and the upper-triangular matrix R based on the Householder QR algorithm such that $V = QR$.

on the type of the sketching being used. In this paper, we look at the following random-sketching techniques that can be implemented using standard linear algebra kernels. Regardless of the types of the sketching used, the last step of the randCholQR, to generate the well-conditioned basis vectors through forward substitutions, requires $O(ns^2)$ computation (Line 6 of Figure 4b). Hence, in the discussion below, we focus on the first two steps of the algorithm (Lines 2 and 4).

- Gaussian-Sketching can be implemented using a dense GEneral Matrix-Matrix multiply (GEMM). The nice feature of this approach is that it requires the sketch size of only $O(s)$ [23]. However, this is a "dense" sketch, and the dense sketch matrix Θ needs to be explicitly stored to call GEMM. Hence, the Gaussia Sketch has the storage overhead of $O(ns)$ and the computational complexity of $O(ns^2)$ to generate the sketch $\widehat{\mathbf{v}}_j$, and its overall computational complexity is $O(ns^2 + s^3)$.

  Though this complexity is the same as CholQR, the complexity of RandHH with the dense Gaussian-Sketch has a larger constant associated with the sketch size (i.e., ns^2 flops for CholQR to compute the Gram matrix, compared to $4ns^2$ floating-point operations (flops) for RandHH to generate the sketched vectors with the sketch size of $2s$).

  The terms associated with s^2 and s^3 in the complexity $O(ns^2 + s^3)$ could become significant, especially when we need to sketch a large number of basis vectors (e.g., for the two-stage approach discussed in Section 6).

- Count-Sketching can be implemented using Sparse-Matrix Matrix (dense vectors) multiply (SpMM) with a sparse sketch matrix Θ having one nonzero entry in each row (with numerical values of either 1 or -1). This is a "sparse" sketching, and compared to Gaussian-Sketching, it has a lower storage cost of $O(n)$ and a lower computational complexity of $O(ns)$.

  One drawback, however, is that it requires the larger sketch size of $O(s^2)$ [34]. This could be a significant performance overhead, or a sequential performance bottleneck, when we need to sketch a large number of basis vectors, s. In particular, on a distributed-memory computer, the basis vectors $\widehat{V}_j$ are distributed among the MPI processes in a 1D block row format (see Section 10 for more detailed discussion about our implementation). Hence to form sketched vectors $\widehat{\mathbf{v}}_j$, it requires the global all-reduce of $O(s^2)$-by-s dense sketched vectors, i.e.,

$$\widehat{\mathbf{v}}_j := \sum_{p=1}^{n_p} (\Theta^{(p)})^T \widehat{V}_j^{(p)},$$

  where $\Theta^{(p)}$ and $\widehat{V}_j^{(p)}$ are the parts of Θ and $\widehat{V}_j$, which is distributed on the p-th process, respectively, and $\sum_{p=1}^{n_p}$ is the global all-reduce. The communication volume $O(s^3)$ needed for the all-reduce can become significant, especially for a large s (e.g., for the two-stage algorithm).

  In addition, the Householder QR factorization of the sketched vectors is performed redundantly on a CPU by each MPI process (Line 4 of Figure 4b), leading to a potential parallel performance bottleneck (i.e., the $O(s^4)$ complexity for the local computation with the Count-Sketch, compared to the $O(s^3)$ complexity with the Gaussian-Sketch).

  The overall computational complexity of the Count Sketch is $O(ns + s^4)$, compared to $O(ns^2 + s^3)$ of the Gaussian sketch.

- To combine the advantage of the above two sketching approaches, Count-Gaussian Sketching uses a $O(s^2) \times n$ Count-Sketch followed by a $O(s) \times O(s^2)$ Gaussian Sketch. Hence, most of the computational and storage costs are due to Count-Sketch, and then the Gaussian Sketch is locally applied such that the size of the final sketched vector is $O(s)$.

  Compared to the Count-Sketch, the size of the global all-reduce is reduced from $O(s^2)$-by-s to $O(s)$-by-s, i.e.,

$$\widehat{\mathbf{v}}_j := \sum_{p=1}^{n_p} \Theta_g^T ((\Theta_c^{(p)})^T \widehat{V}_j^{(p)}),$$

  where Θ_c and Θ_g are the Count and Gaussian sketch matrices, respectively. The communication volume is reduced from the Count-Sketch because the Gaussian sketch is applied locally before the global all-reduce. Hence the communication volume for the all-reduce is $O(s^2)$, and is the same as the Gaussian-Sketch and is reduced from $O(s^3)$ needed for Count-Sketch.

  Though it is now requires only $O(s^3)$ computation to compute the QR factorization of the final sketch, the Gaussian sketch is applied redundantly on each MPI process, which has the computational complexity of $O(s^4)$ and is

the same as that needed for the Count-Sketch. Nevertheless, the local GEMM for the Count-Gauss may perform more efficiently than the local HH for Count-Sketching. Moreover, in our implementation, GEMM before the global-reduce is computed on a GPU, while HH afer the global reduce is computed on CPU.

Compared to the Gaussian-Sketch (or to the generation of the Gram matrix for CholQR), the Count-Gaussian reduces the computation complexity to apply the sketch from $O(ns^2)$ to $O(ns)$. Moreover, even if the Count-Gaussian did not lead to a performance improvement over the Gaussian-Sketch in practice (due to a more efficient dense matrix-matrix multiply implementation, compared to a sparse-matrix matrix multiply), the storage requirement is reduced from $O(ns)$ to $O(n)$.

The overall computational complexity of the Count-Gauss is $O(ns + s^4)$.

Proposition 5.2. *When RandCholQR is used as the first intra-block orthogonalization, the orthogonality error of the resulting basis vectors $Q_{1:j}$ is of the order of the machine precision when Θ^T forms an μ-subspace embedding over the range of $\widehat{V}_j$, and the following condition is satisfied:*

$$c_2(\epsilon, n, s)\kappa(\widehat{V}_j) < 1/2, \tag{9}$$

where $c_2(\epsilon, n, s)$ is a constant, or equivalently if MPK generates numerically full-rank basis vectors $\widehat{V}_j$.

Proof. The proof of Proposition 5.2 follows by combining the results from two facts: first, BCGS2 attains $O(\epsilon)$ orthogonality error overall provided each "IntraBlk" step in line 3 attains $O(\epsilon)$ orthogonality error [7], and RandCholQR attains $O(\epsilon)$ orthogonality error when (9) is satisfied [33]. □

Hence, compared to CholQR2, which requires (8), RandCholQR improves the numerical robustness of the overall block orthogonalization process, while we expect only a small overhead in term of its execution time (as shown in Section 11).

Input: $Q_{1:\ell}$, s, $\hat{s}$
Output: $Q_{\ell+1:t+1}$ and $R_{1:t+1,\ell+1:t+1}$
1: *// t is last block column ID of the next big panel*
2: $t := \ell + \hat{s}/s - 1$
3: **for** $j = \ell + 1, \ell + 2, \ldots, t + 1$ **do**
4: *// Matrix-Powers Kernel*
5: **for** $k = 1, 2, \ldots, s$ **do**
6: $v_{k+1}^{(j)} := A v_k^{(j)}$
7: **end for**
8: *// Orthogonalize new panel*
9: *// against previous big panels*
10: $[\widehat{V}_j, R_{1:\ell,j}] :=$ BCGS($Q_{1:\ell}$,V_j)
11: *// Intra-Big Panel PreProcess*
12: $[\widehat{Q}_j, R_{\ell+1:j,j}] :=$ PreProc($\widehat{Q}_{\ell+1:j-1}$, $\widehat{V}_j$)
13: **end for**
14: *// CholQR of big panel*
15: $[T_{\ell+1:t+1,\ell+1:t+1}, Q_{\ell+1:t+1}] :=$ CholQR($\widehat{Q}_{\ell+1:t+1}$)
16: $R_{\ell+1:t+1,\ell:t+1} := T_{\ell+1:t+1,\ell+1:t+1} R_{\ell+1:t+1,\ell+1:t+1}$
17: **if** $\ell > 0$ **then**
18: *// Orthogonalization*
19: $[Q_{\ell+1:t+1}, T_{1:t+1,\ell+1:t+1}] =$ BCGS+CholQR($Q_{1:\ell}$, $Q_{\ell+1:t+1}$)
20: $R_{1:\ell,\ell:t+1} := T_{1:\ell,\ell+1:t+1} R_{\ell:t+1,\ell+1:t+1} + R_{1:\ell,\ell+1:t+1}$
21: $R_{\ell+1:t+1,\ell+1:t+1} := T_{\ell+1:t+1,\ell+1:t+1} R_{\ell+1:t+1,\ell+1:t+1}$
22: **end if**

Figure 5: Two-stage Block Orthgonalization with MPK.

Input: $Q_{1:\ell}$ and $V_{\ell+1:t+1}$
Output: $Q_{\ell+1:t+1}$ and $R_{1:t+1,k:t+1}$
1: *// First inter big-panel orthogonalization*
2: $[\widehat{V}_{\ell+1:t+1}, R_{1:\ell,\ell+1:t+1}] :=$ BCGS($Q_{1:\ell}$,$V_{\ell+1:t+1}$)
3: *// First intra big-panel orthogonalization*
4: **for** $j = \ell + 1, \ell + 2, \ldots, t$ **do**
5: $[\widehat{Q}_j, R_{\ell+1:j,j}] :=$ PreProc($\widehat{Q}_{\ell+1:j-1}$, $\widehat{V}_j$)
6: **end for**
7: $[T_{\ell+1:t+1,\ell+1:t+1}, Q_{\ell+1:t+1}] :=$ CholQR($\widehat{Q}_{\ell+1:t+1}$)
8: $R_{\ell+1:t+1,\ell:t+1} := T_{\ell+1:t+1,\ell+1:t+1} R_{\ell+1:t+1,\ell+1:t+1}$
9: **if** $\ell > 0$ **then**
10: *// Re-orthogonalization of big panel*
11: $[Q_{\ell+1:t+1}, T_{1:t+1,\ell+1:t+1}] =$ BCGS+CholQR($Q_{1:\ell}$, $Q_{\ell+1:t+1}$)
12: $R_{1:\ell,\ell:t+1} := T_{1:\ell,\ell+1:t+1} R_{\ell:t+1,\ell+1:t+1} + R_{1:\ell,\ell+1:t+1}$
13: $R_{\ell+1:t+1,\ell+1:t+1} := T_{\ell+1:t+1,\ell+1:t+1} R_{\ell+1:t+1,\ell+1:t+1}$
14: **end if**

Figure 6: Two-stage Block Orthgonalization without MPK.

6. Two-stage Block Orthogonalization Framework

BCGS2 discussed in Section 5 can orthogonalize the set of $s + 1$ basis vectors with only five synchronizations and using BLAS-3 operations to performs most of its local computation. However, its performance may be still limited by the small step size s required to maintain the stability of MPK. In order to improve the performance of the block orthogonalization while using the small step size, "two-stage" block orthogonalization algorithms have been proposed [6]. Instead of fully-orthogonalizing the basis vectors at every s steps, the two-stage algorithm only "pre-processes" the s-step basis vectors V_j at every s steps. The objective of this first stage is to maintain the well-conditioning of the basis vectors at a cost that is lower than that required for the full orthogonalization or with the same cost as the initial orthogonalization (e.g., BCGS with CholQR). Then once a sufficient number of basis vectors, $\widehat{s}$, are generated to obtain higher performance, the second stage orthogonalizes the $\widehat{s}$ basis vectors at once. For the following discussion, we refer to the blocks of s and $\widehat{s}$ vectors as the "panels" and "big panels", respectively. Also, to distinguish from the two-stage algorithms, we refer to the BCGS algorithms discussed in Section 5 as "one-stage" algorithms.

For this paper, we focus on the specific variant of the two-stage algorithm shown in Figure 5. This variant first (roughly) orthogonalizes the new panel of basis vectors V_j against the previous big panels using BCGS (Line 10). It then pre-processes the resulting panel vectors $\widehat{V}_j$ within the current big panel (Line 12). Finally, the second stage fully orthogonalizes the big panel, first orthogonalizing the big panel using CholQR (Line 15), followed by BCGS with CholQR on the big panels (Line 19).

The main objective of the pre-processing stage is to keep the condition number of the basis vectors small at a low cost. Namely, for the two-stage algorithm to maintain the same level

of the stability as the one-stage algorithm, after the first inter-big panel BCGS (Line 10 of Figure 5), the condition number of the big panel is hoped to be the same order as that of each panel in the one-stage algorithm,

$$\kappa(\widehat{V}_{\ell+1:j}) \approx \kappa(\widehat{V}_{\ell+1}). \tag{10}$$

Without the pre-processing step (i.e., one-stage with $s = \widehat{s}$), the condition number of the basis vectors will increase exponentially with the step size (see numerical results in [6]).

As it can be seen as the generic block orthogonalization scheme, without MPK, in Figure 6, this two-stage algorithm can be considered as BCGS2 (shown in Figure 2c) that orthogonalizes the big panel as a set of block vectors, where the pre-processing step, followed by CholQR, is used as the first intra-block orthogonalization. Hence, if the condition number of the big panel $\widehat{Q}_{\ell+1:t+1}$, after the pre-processing step (Lines 4 to 6) is $O(1)$, then the orthogonality error of the big panel $Q_{\ell+1:t+1}$ after the first CholQR (Line 7) is $O(\epsilon)$, and thus, the overall two-stage block orthogonalization is stable with $O(\epsilon)$ orthogonality error of the resulting vectors, i.e.,

$$\|I - Q_{1:t+1}^T Q_{1:t+1}\| = O(\epsilon).$$

In the next section, we introduce two pre-processing schemes and discuss the condition on the input big panel $\kappa(\widehat{V}_{\ell+1:t+1})$ to ensure the overall stability of the two-stage algorithm.

7. Preprocessing Schemes for Two-stage Framework

We first formally establish the following proposition.

Proposition 7.1. *If a pre-processing scheme can maintain the condition number of the big panel to be bounded as*

$$\kappa\left(\widehat{Q}_{\ell+1:t+1}\right) = O(1), \tag{11}$$

then the overall stability of the two-stage scheme is ensured.

Proof. Using (7), condition (11) implies that $Q_{\ell+1:t+1}$ produced by CholQR of $\widehat{Q}_{\ell+1:t+1}$ at Line 7 of Figure 6 satisfies

$$\|I - Q_{\ell+1:t+1}^T Q_{\ell+1:t+1}\| = O(\epsilon).$$

Therefore, by [7], the overall stability of the two-stage scheme is ensured. □

7.1. BCGS with Pythagorean Inner Product (BCGS-PIP)

Our first pre-processing scheme for the two-stage algorithm is based on the single-reduce variant [30, 35] of BCGS with CholQR, shown in Figure 7. Instead of explicitly computing the Gram matrix, this variant uses the Pythagorean rule, hence requiring only one global-reduce for both the inter-block and intra-block orthogonalization.[3] In addition, it was shown [30, Theorem 3.4] that if the condition number of the input basis vectors is bounded as

$$c_3(\epsilon, n, \widehat{s})\kappa(\widehat{V}_{\ell+1:t+1})^2 < 1/2, \tag{12}$$

[3]For the first block (i.e., $j = \ell + 1$), BCGS-PIP is equivalent to CholQR.

Input: $Q_{1:j-1}$ and V_j
Output: Q_j and $R_{1:j,j}$
1: *// Dot-products*
2: $R_{1:j,j} := [Q_{1:j-1}, V_j]^T V_j$
3: *// Gram matrix generation by Pythagorean*
4: $G := R_{j,j} - R_{1:j-1,j}^T R_{1:j-1,j}$
5: *// CholQR intra-block orthogonalization*
6: $R_{j,j} := \mathrm{Chol}(G)$
7: *// Block orthogonalization*
8: $\widehat{V}_j := V_j - Q_{1:j-1} R_{1:j-1,j}$
9: $Q_j := \widehat{V}_j R_{j,j}^{-1}$

Figure 7: BCGS with Pythagorean Inner Product (BCGS-PIP).

where $c_3(\epsilon, n, s)$ is a constant and $t+1$ is the last block index of the big panel (i.e., $t := \ell + \widehat{s}/s - 1$), then the orthogonality error of the big panel computed by BCGS-PIP satisfies

$$\|I - \widehat{Q}_{\ell+1:t+1}^T \widehat{Q}_{\ell+1:t+1}\| \le c_3(\epsilon, n, \widehat{s})\kappa(\widehat{V}_{\ell+1:t+1})^2. \tag{13}$$

Hence, we have the following proposition:

Proposition 7.2. *If this pre-processing scheme can maintain the condition number of the big panel to satisfy the condition* (12)*, then the overall stability of the two-stage scheme is ensured.*

Proof. Because of the assumption (12) and the bound (13), we have

$$\|I - \widehat{Q}_{\ell+1:t+1}^T \widehat{Q}_{\ell+1:t+1}\| < 1/2.$$

Thus, by Weyl's inequality, we obtain

$$\begin{cases} \sigma_{\min}\left(\widehat{Q}_{\ell+1:t+1}^T \widehat{Q}_{\ell+1:t+1}\right) \ge 1 - \|I - \widehat{Q}_{\ell+1:t+1}^T \widehat{Q}_{\ell+1:t+1}\| > 1/2 \\ \sigma_{\max}\left(\widehat{Q}_{\ell+1:t+1}^T \widehat{Q}_{\ell+1:t+1}\right) \le 1 + \|I - \widehat{Q}_{\ell+1:t+1}^T \widehat{Q}_{\ell+1:t+1}\| < 3/2. \end{cases}$$

Therefore,

$$\kappa\left(\widehat{Q}_{\ell+1:t+1}\right) = \sqrt{\kappa\left(\widehat{Q}_{\ell+1:t+1}^T \widehat{Q}_{\ell+1:t+1}\right)} = O(1), \tag{14}$$

and the proof follows by Proposition 7.1. □

A similar two-stage scheme based on BCGS-PIP was studied in [6]. The variant studied in this paper is slightly different and has a more stable behavior. Namely, its robustness depends on the condition number of the big panel $\widehat{V}_j$ as shown in the condition (12), while the stability of the previous variant depended on the condition number of the accumulated big panels, i.e., $c(\epsilon)\kappa([Q_{1:\ell}, V_{\ell+1:t+1}])^2 < 1/2$.

Similarly to CholQR, BCGS-PIP can fail when the condition number of the big panel is greater than the reciprocal of the square-root of the machine epsilon ϵ. This can cause numerical issues (especially for the ill-conditioned basis vectors generated by MPK).

7.2. Randomized BCGS

To enhance the stability of the pre-processing scheme based on BCGS-PIP, we consider a randomized BCGS scheme shown in Figure 8 as our pre-processing algorithm. This algorithm sketches the big panel, but s basis vectors at a time. To ensure stability, we orthogonalize the sketched vectors $\widehat{\mathbf{v}}_{\ell+1:t+1}$ using

Input: V_j (new block),
$\widehat{Q}_{j_1:j-1}$ (previous approximately orthogonal blocks), and
$\mathbf{q}_{j_1:j-1}$ (previous sketch vectors)
Output: $\widehat{Q}_j$, $\mathbf{q}_j$, $R_{1:j,j}$
1: *// Sketch the new "panel"*
2: $\mathbf{v}_j := \Theta^T V_j$
3: *// Orthogonalize new* **sketched** *panel within big panel*
4: $[\mathbf{q}_j, R_{j_1:j,j}] :=$ BCGS2-HH($\mathbf{q}_{j_1:j-1}, \mathbf{v}_j$)
5: *// Generate well-conditioned basis*
6: $\widehat{V}_j := V_j - \widehat{Q}_{j_1:j-1} R_{j_1:j-1,j}$
7: $\widehat{Q}_j := \widehat{V}_j R_{j,j}^{-1}$

(a) The jth step (for PreProc for Figure 6)

Input: $V_{1:t+1}$
Output: $\widehat{Q}_{1:t+1}$, $R_{1:t+1,1:t+1}$
1: *// Sketch the big panel*
2: $\mathbf{v}_{1:t+1} := \Theta^T V_{1:t+1}$
3: *// Orthogonalize the sketch of the big panel*
4: **for** $j = 1, 2, \ldots, t+1$ **do**
5: $[\mathbf{q}_j, R_{1:j,j}] :=$ BCGS2-HH($\mathbf{q}_{1:j-1}, \mathbf{v}_j$)
6: **end for**
7: *// Generate well-conditioned basis by forward-substitution*
8: **for** $j = 1, 2, \ldots, t+1$ **do**
9: $\widehat{V}_j := V_j - \widehat{Q}_{1:j-1} R_{1:j-1,j}$
10: $\widehat{Q}_j := \widehat{V}_j R_{j,j}^{-1}$
11: **end for**

(b) Accumulated steps.

Figure 8: Randomized BCGS2 (RandBCGS2).

BCGS2 with Householder intra-block orthogonalization. Similar randomized block orthogonalization algorithms were discussed in [14, 15, 33]. We used this randomized algorithm as the pre-processing scheme for the two-satage BCGS2, in order to maintain the well-conditioning of the big panel $\widehat{Q}_{\ell+1:t+1}$ and obtain the overall $O(\epsilon)$ orthogonality error of $Q_{1:t+1}$.

Proposition 7.3. *The overall stability of two-stage algorithm is ensured when randomized BCGS pre-processing is used with Θ^T that forms an μ-subspace embedding over the range of $\widehat{V}_{\ell+1:t+1}$, and the following condition is satisfied:*

$$c_4(\epsilon, n, \widehat{s})\kappa(\widehat{V}_{\ell+1:t+1}) < 1/2. \tag{15}$$

where $c_4(\epsilon, n, s)$ is a constant, or equivalently when MPK generates numerically full-rank basis vectors for each big panel $\widehat{V}_{\ell+1:t+1}$.

Proof. Observe that RandBCGS2 (in Figure 8b) is identical to RandHH (in Figure 4a) except that BCGS2-HH is used, instead of HH factorization, on the sketched vectors to generate the upper-triangular matrix. In [33, Corollary 5.1], it was proven that RandHH results in $\kappa\left(\widehat{Q}_{\ell+1:t+1}\right) = O(1)$.

We prove in Appendix that the backward error of the QR factorization via BCGS2-HH only differs from the Householder QR backward error by a constant factor. Hence, the backward error analysis in [33, Section 5.2.4] only differs by a constant factor when BCGS2+HH, instead of HH, was used on the sketched vectors. Therefore, by [33, Corollary 5.1], RandBCGS2 will generate the basis vectors $\widehat{Q}_{\ell+1:t+1}$ whose condition number is bounded by $\kappa\left(\widehat{Q}_{\ell+1:t+1}\right) = O(1)$, and the proof follows by Proposition 7.1. □

Assuming that the condition number of the big panel is in the same order as that of each panel in the one-stage algorithm (as in the condition (10)), the condition (15) on the big panel for the two-stage algorithm is equivalent to the condition (9) on the panel for the one-stage algorithm, and hence the two-stage algorithm with RandBCGS2 is as stable as the one-stage algorithm with RandHH.

The roundoff error analysis of a randomized BCGS, that is similar to the one in Figure 8, has been presented in [14], where their framework allows generating the sketches of the block for the inter and intra block orthogonalization, separately, while in this paper, we focus on the one in Figure 8 that uses a single sketch of the big panel (similar to randCholQR, but by generating the sketch of each block at a time).

BCGS-PIP and RandBCGS2, shown in Figures 7 and 8, respectively, have a similar algorithmic structure. In particular, both algorithms would require one global-reduce, followed by a small local computation (either Cholesky factorization of the Gram matrix or BCGS2 orthogonalization of the sketched vectors) and forward-substitution to generate the basis vectors $\widehat{Q}_j$. Though RandBCGS2 has the larger complexity for the local computation, the main factor that impacts their performance difference is the dot-products required for BCGS-PIP and the random-sketching required for RandGCGS.

7.3. Remarks on Complexity

We discuss the overall complexity of the various block orthogonalization algorithms in Section 8, but we provide a few remarks on the complexity, comparing the one-stage and two-stage algorithms, here.

Compared to the one-stage algorithm with RandHH, the two-stage algorithm with RandBCGS2 reduces the communication cost by delaying the orthogonalization until $\widehat{s}+1$ basis vectors are generated. However, this two-stage algorithm needs to sketch the big panel with the total of $\widehat{s}+1$ basis vectors, hence requiring a larger sketch size than the one-stage algorithm. For instance, with the Gaussian sketch, its sketch size for the two-stage algorithm is proportional to the number of columns in the big panel, $\widehat{s}+1$, while the one-stage algorithm sketches each panel of $s+1$ basis vectors, and its sketch size is proportional to the panel size, $s+1$. Hence, in the randomized two-stage algorithm, there is a trade-off between the reduced communication cost for the orthogonalization and the increased computational cost for sketching, which we discuss in the next section.[4]

When $\widehat{s} < m$, the two-stage algorithm with the two preprocessing schemes in Section 7 requires about the same computational cost as the one-stage algorithm. On the other hand, when $\widehat{s} = m$, the two-stage algorithm has a lower computational cost than the one-stage algorithm (see the asymptotic complexity discussion in Section 8). This is because if the preprocesing step can keep the well-conditioning of the basis vectors over the whole restart-cycle of s-step GMRES, then the reorthogonalization (Line 19) is not needed, reducing the total computational

[4] As Count-Gaussian sketching has $O(n)$ complexity, it has the potential to remove this overhead of the randomized two-stage algorithm.

	Flop Count					
	Projection		Normalization		Total	
One-stage:						
standard	$2nm(m+1/2)$	$2nm(m+1/2)$	$3nm$	$3nm$	$2nm^2$	$2nm^2$
s-step	$2nm(s+1)(m-s+1/2)/s$	$2nm(s+1)(m-s+1/2)/s$	$4nm(s+1)(s+3/2)/s$	$2nm(s+1)(s+3/2)/s$	$2nm^2(s+1)/s$	$2nm^2(s+1)/s$
sketch($\hat{s}=s$)	$2nm(s+1)(m-s+1/2)/s$	$2nm(s+1)(m-s+1/2)/s$	$4nm(s+1)(7s/4+2)/s$	$2nm(s+1)(s+3/2)/s$	$2nm^2(s+1)/s$	$2nm^2(s+1)/s$
Two-stage:						
pip	$2nm(s+1)(m-s+1/2)/s$	$2nm(s+1)(m-s+1/2)/s$	$4nm(s+1)(s+3/2)/s$	$2nm(s+1)(s+3/2)/s$	$2nm^2(s+1)/s$	$2nm^2(s+1)/s$
sketch($s<\hat{s}<m$)	$2nm(s+1)(m-\hat{s}+1/2)/s$	$2nm(\hat{s}+1)(m-\hat{s}-1/2)/\hat{s}$	$2nm(s+1)(5\hat{s}/2-s/2+5/2)/s$ $+2nm(\hat{s}+1)(\hat{s}+3/2)/\hat{s}$	$2nm(\hat{s}+1)(\hat{s}+3/2)/\hat{s}$	$2nm^2(s+1)/s$	$2nm^2(\hat{s}+1)/\hat{s}$
sketch($\hat{s}=m$)	$2nm(s+1)(5m/2-s/2+5/2)/s$		$nm(s+1)^2/s$	$2n(m+1)(m+3/2)$	$5nm^2(s+1)/s$	$2nm^2$

Table 2: Asymptotic computational complexity of orthogonalization scheme over one GMRES restart cycle, where for a given starting unit vector, the "standard" and "s-step" are based on CGS2 and BCGS2 with CholQR2, respectively. For the complexity with the random sketch, we use Gaussian Sketch with the sketch size of $\widehat{m} = 2(\widehat{s}+1)$. For each components, there are two columns for each component, "Projection", "Normalization", and "Total", where for the standard and s-step algorithms, the first and second columns are for orthogonalization and then for reorthogonalization, while for the remaining algorithms, they are for the preprocessing based on Gaussian random-sketching and then for orthogonalization. For two-stage, "Projection" and "Normalization" are the inter-block and intra-block orthogonalization of big panels. "Total" only shows the leading terms of the complexity.

cost of the orthogonalization. In this case, it is also possible to skip the ℓ_2-orthogonalization (i.e., CholQR on Line 15), and solve the least-square problem in the sketched space. Nevertheless, in this paper, we will focus on generating the ℓ_2-orthogonal basis vectors, but will show the breakdown of the orthogonalization time, including the CholQR time, in Section 11.

8. Asymptotic Complexity

Tables 2 and 3 compare the computational, storage, and communication complexities of the different block orthogonalization schemes, respectively. These complexities are total costs using one MPI process, and not for distributed-memory.

- The s-step GMRES includes the starting vector to the set of the vectors to be orthogonalize. For instance, with $s = 1$, s-step orthogonalizes two vectors at each step. This leads to about $2\times$ more flops for "Projection" for s-step GMRES, compared to the standard GMRES. In addition, CholQR of two vectors requires about $(10/3)\times$ more flops than computing a dot-product and scaling of a single vector.
- The randomized block orthogonalization algorithm performs the preprocessing based on random-sketching to generate the well-conditioned basis vectors, followed by the ℓ_2-orthogonalization of the basis vectors. Hence, the randomized algorithm replaces the first orthogonalization process of the standard algorithm with the randomized transformation of the basis vectors, while the orthogonalization process is identical and performs the same number of flops as the re-orthogonalization process of the standard algorithm.
- With $\widehat{s} = s$ (one-stage), compared to CholQR2, Rand-CholQR has higher computation complexity and larger communication volume, but the relative overhead is small (about the order of s/m) in the total complexity. This leads to insignificant increase in the orthogonalization time (for improving the numerical stability) in our performance tests.

	Storage	Communication	
		Latency	Volume
standard	nm	$4m$	$nm(2m+4)$
s-step	nm	$4\frac{m}{s}$	$nm\frac{2m+4+4s}{s}$
sketch($\widehat{s}=s$)	$n(m+\widehat{m})$	$4\frac{m}{s}$	$nm\frac{2m+4+4s+\widehat{m}}{s}$
sketch($s<\widehat{s}<m$)	$n(m+\widehat{m})$	$\frac{m}{s}+3\frac{m}{\widehat{s}}$	$nm(\frac{m+2+2s+\widehat{m}}{s}+\frac{m+2+4\widehat{s}}{\widehat{s}})$
sketch($\widehat{s}=m$)	$n(m+\widehat{m})$	$\frac{m}{s}+1$	$nm\frac{m/2+\widehat{m}+2+2s}{s}+2nm$

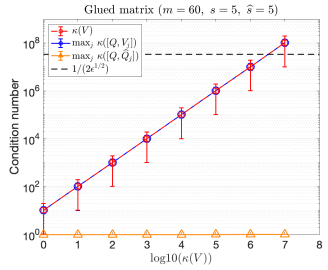
Table 3: Asymptotic storage and communication costs of orthogonalization scheme over one GMRES restart cycle, where "standard" and "s-step" are based on CGS2 and block CGS2 followed by CholQR2, respectively.

- One-stage algorithm with CholQR2 and two-stage algorithm with BCGS-PIP have about the same computational complexity.
- With $\widehat{s} = m$, the two-stage framework has the best communication latency cost, but random-sketching has the highest computational overhead, where the total computational cost of the orthogonalization is increased by a factor of $1.75\times$. Nevertheless, when the performance is limited by the latency, this computational overhead may not be significant.

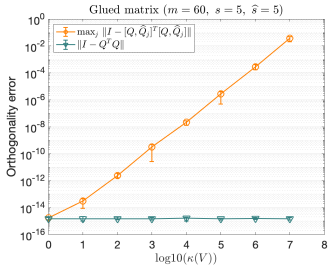
  Though the storage cost is increased by a factor of three, which could limit the use of the dense Gaussian sketch, the overhead can be reduced by using sparse Count sketch.
- These storage and computational overheads may be reduced using a smaller sketch size ($s < \widehat{s} < m$). This reduces the overhead to be $O(\widehat{s}/m)$, though the reduction in the latency is also reduced. Nevertheless, in our experiments, the two-stage algorithm obtained the best performance using $\widehat{s} = m$.

9. Numerical Experiments

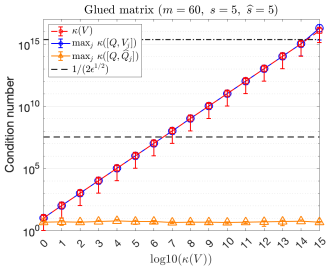
We compared the orthogonality errors of the proposed block orthogonalization schemes using the default double precision in MATLAB. For these studies, instead of studying the numerical properties of the proposed methods within the s-step GMRES, we treat them as general block orthogonalization schemes and use synthetic matrices as the input vectors. This allows us to



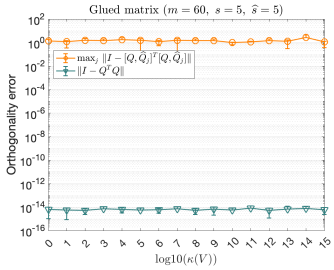
(a) Condition number with CholQR2.



(b) Orthogonality errors with CholQR2.



(c) Condition number with RandCholQR.



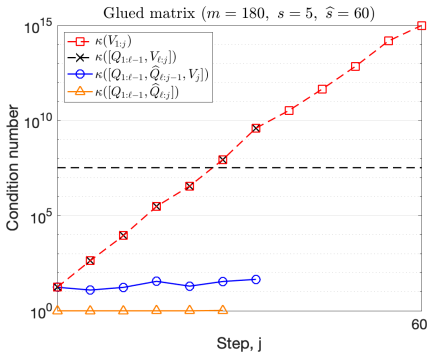
(d) Orthogonality errors with RandCholQR.

Figure 9: Condition number and orthogonality error with one-step BCGS2 with CholQR2 or RandCholQR first intra-block orthogonalization on `glued` matrix.

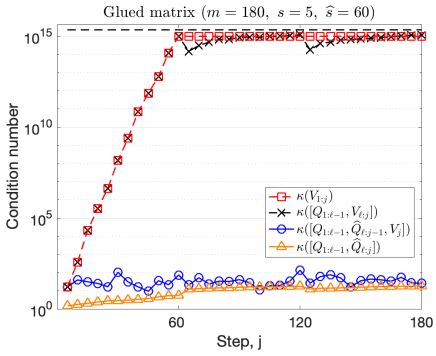
control the condition number of the matrix easily. Numerical results showing how the condition numbers of the Krylov vectors could grow, can be found in [6].

We first study how the orthogonality errors grow with the condition numbers of the input vectors for the one-stage algorithms. Figure 9 shows the orthogonality error when CholQR2 or RandCholQR are used for the first intra block orthogonalization of the one-stage BCGS2. Our test matrix is the `glued` matrix [36] that has the same specified order of the condition number for each panel V_j and for the overall matrix $V_{1:12}$. As expected, the orthogonality error of the basis vectors $\widehat{Q}$ was $O(\epsilon)$ using CholQR2 when the condition number of the input matrix is smaller than $O(\epsilon)^{-1/2}$. With RandCholQR, the same $O(\epsilon)$ orthogonality errors were obtained as long as the input matrix is numerically full-rank with the condition number of $O(\epsilon)^{-1}$, and hence demonstrating superior stability compared to CholQR2.

Next, we show that the two-stage approach obtains $O(\epsilon)$ orthogonality error when the condition (12) or (15) is satisfied, using BCGS-PIP or RandBCGS2, respectively. Figure 10 shows the condition number of basis vectors using the two-stage approach with BCGS-PIP or RandBCGS2 as the preprocessing schemes, while Figure 11 shows the orthogonality errors using RandBCGS2. The test matrix is the `glued` matrix, where each big panel has the condition number $O(10^{15})$. For this synthetic matrix, BCGS-PIP failed when the condition number of the accumulated panels increased more than $O(\epsilon)^{-1/2}$. In contrast, RandBCGS2 managed to keep the $O(1)$ condition number of the big panel $[Q_{1:\ell-1}, \widehat{Q}_{\ell:t}]$, and the overall orthogonality error of Q was $O(\epsilon)$.



(a) BCGS-PIP.



(b) RandBCGS2

Figure 10: Condition number (marker at every s steps) using two-stage approach with BCGS-PIP and RandBCGS2 on `glued` matrix with $(n, m, \hat{s}, s) = (100000, 180, 60, 5)$.

10. Implementation

To study the performance of the block orthogonalization algorithms for s-step GMRES running on a GPU cluster, we have implemented these algorithms within the Trilinos software framework [24, 13]. Trilinos is a collection of open-source software libraries, called packages, for solving linear, non-linear, optimization, and uncertainty quantification problems. It is intended to be used as building blocks for developing large-scale scientific or engineering applications. Hence, any improvement in the solver performance could have direct impacts to the application performance. In addition, Trilinos software stack provides portable performance of the solver on different hardware architectures, with a single code base. In particular, our implementation is based on Tpetra [37, 38] for distributed matrix and vector operations and Kokkos-Kernels [39] for the on-node portable matrix and vector operations (which also provides the interfaces for the vendor-optimized kernels like NVIDIA cuBLAS, cuSparse, and cuSolver).

On a GPU cluster, our GMRES implementation uses GPUs to generate the orthonormal basis vectors. The coefficient matrix A and Krylov vectors V are distributed among MPI processes in 1D block row format (e.g., using a graph partitoner like ParMETIS), where A is locally stored as the Compressed Sparse Row (CSR) format, while V is stored in the column-major format. The operations with the small projected matrices, including solving a small least-squares problem, is redundantly done on CPU by each MPI process.

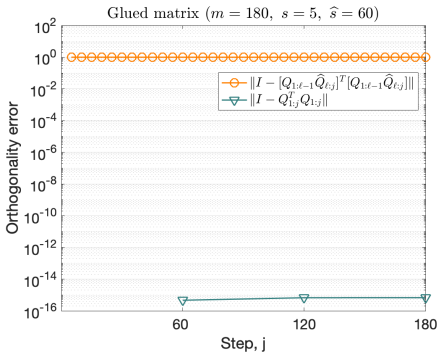


Figure 11: Orthogonality error (orange circle marker at every s steps, while green triangle marker at every $\widehat{s}$ steps) using two-stage approach with Rand-BCGS2 on `glued` matrix with $(n, m, \hat{s}, s) = (100000, 180, 60, 5)$.

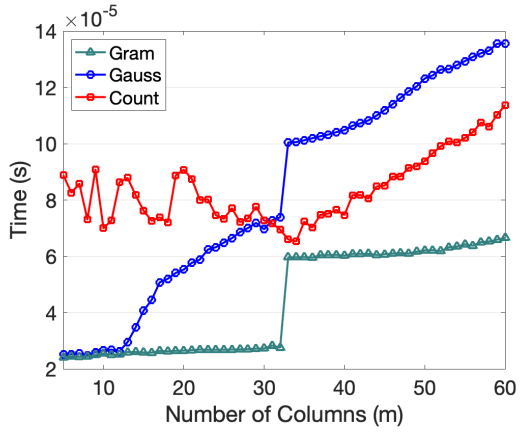
Our focus is on the block orthogonalization of the vectors, which are distributed in 1D block row format among the MPI processes. The orthogonalization process mainly consists of dot-products, vector updates, and vector scaling (e.g., $R_{1:j-1,j} := Q_{1:j-1}^T V_j$ and $\widehat{V}_j := V_j - Q_{1:j-1}R_{1:j-1,j}$ of BCGS in Figure 2a, and $Q_j := \widehat{V}_j R_{j,j}^{-1}$ of CholQR in Fig. 2b, respectively). The dot-products $Q_{1:j-1}^T V_j$ requires a global reduce among all the MPI processes, and the resulting matrix $R_{1:j-1,j}$ is stored redundantly on the CPU by all the MPI processes. Given the upper-triangular matrix, the vectors can be updated and scaled locally without any additional communication. All the local computations are performed by the computational kernels through Kokkos Kernels, either on a CPU or on a GPU.

We have implemented the random-sketching using standard linear algebra kernels as discussed in Section 5.2. These distributed-memory or on-node kernels are readily available through Tpetra or Kokkos-Kernels, given that the random-sketching matrix $\boldsymbol{\Theta}$ is explicitly generated and stored in memory.

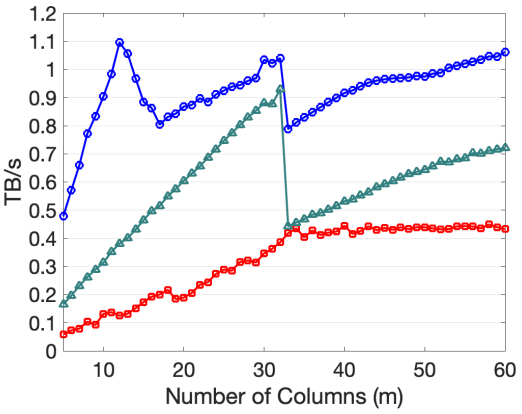
- For Gaussian sketch, the dense sketching matrix Θ is distributed among the MPI processes in the 1D block row format, and each MPI process stores the local matrix in the column major order.
- For Count sketch, the sparse sketching matrix Θ is also distributed in the 1D block row format, where each local sparse matrix is stored in the CSR format.
- For Count-Gaussian sketching, the sparse Count-sketching matrix Θ_c is distributed in the 1D block row format, but the dense Gaussian-sketching matrix Θ_g is duplicated on all the MPI processes such that it can be applied before performing the global-reduce of the final sketched basis vectors.

11. Performance Experiments

We conducted our performance tests on the Perlmutter supercomputer at National Energy Research Scientific Computing (NERSC) Center. Each compute node of Perlmutter has one 64-core AMD EPYC 7763 CPUs and four NVIDIA A100 GPUs. On each node, we launched 4 MPI processes (one MPI per GPU) and assigned 16 CPU cores to each MPI process. For all the performance studies with one-stage or two-stage algorithm, we used the sketch size of $2s$ or $2\hat{s}$ for the Gaussian Sketch, while the sketch size of $2s^2$ or $2\hat{s}^2$ is used for the Count Sketch, respectively.



(a) Time (s).



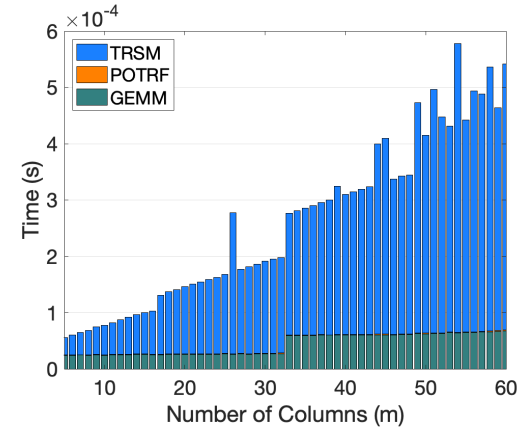
(b) Bandwidth.

Figure 12: Sketch performance on one NVIDIA A100 GPU.

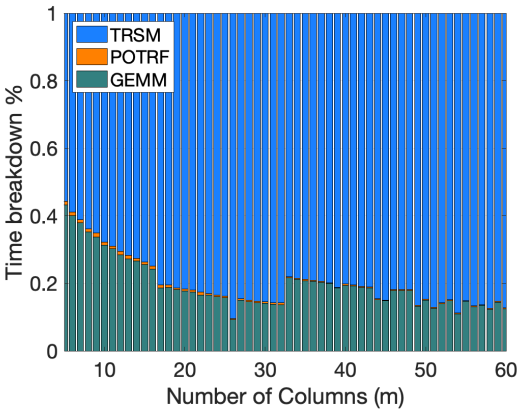
The code was compiled using Cray's compiler wrapper with Cray LibSci version 23.2, CUDA version 11.5 and Cray MPICH version 8.1. The GPU-aware MPI was not available on Perlmutter, and hence, all the MPI communications are performed through the CPU. We configured Trilinos such that all the local dense and sparse matrix operations are performed using CUBLAS and CuSparse, respectively.

11.1. Single-GPU Sketching Performance (Gaussian vs. Count)

Figure 12a compares the time required for Gaussian and Count sketch on a single NVIDIA A100 GPU, with the time required for computing the Gram matrix of the block vectors (e.g., needed for CholQR), where the number of rows is fixed at 10^5, while the number of columns is increased from 5 to 60. The Count Sketch was slower than the Gaussian Sketch for a small number of columns (e.g., $s = 5$ for the one-stage block orthogonalization). However, the time required for the Count Sketch stayed relatively constant with the increasing number of

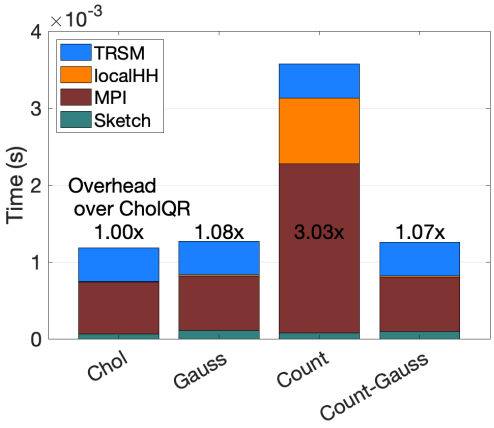


(a) Time (s).

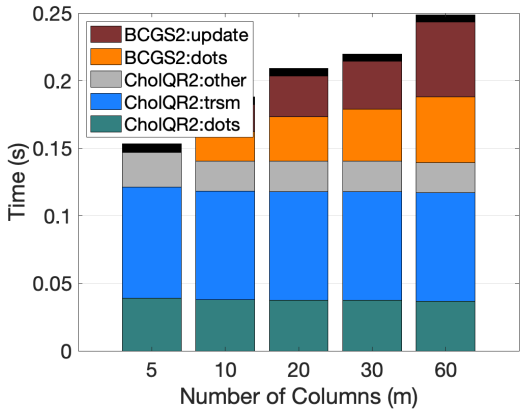


(b) Ratio.

Figure 13: Time breakdown for CholQR on one NVIDIA A100 GPU.



(a) Breakdown of Intra-block CholQR2 time ($m = 40$).



(b) BCGS2 with CholQR2.

Figure 14: Breakdown of orthogonalization time on four NVIDIA A100 GPUs.

columns (because its computational complexity $O(n)$ is independent of m), and it became faster than the Gaussian Sketch for a large enough number of columns (e.g., $\widehat{s} = 60$ for the two-stage orthogonalization, though we sketch s columns at a time).

Figure 12b shows the observed bandwidth for the three algorithms with 4 bytes and 8 bytes to store the column index for the sparse CSR format and the double-precision numerical value for both the dense vectors and sparse matrix, respectively. We let the amount of the data required to move to be 8 bytes $\cdot\, mn$ for computing the Gram matrix (reading V once), and 8 bytes$\cdot(3mn)$ or 8 bytes$\cdot mn+(8+4)$ bytes$\cdot m$ for the Gaussian Sketch or Count Sketch (reading V and dense or sparse Θ once), respectively. As we expected, due to the irregular data access, the Count Sketch obtained the lower bandwidth than the Gaussian Sketch (which obtained close to the NVIDIA A100 memory bandwidth, 1.5TBytes per second). The bandwidth obtained for computing the Gram matrix was about the half of that observed for the Gaussian Sketch. This could be because Tpetra uses non-symmetric dense matrix-matrix multiply (GEMM) for computing the Gram matrix, potentially doubling the amount of the required data traffic.

11.2. Breakdown of Orthogonalization Time on one and multiple GPUs (Gaussian, Count, vs. CountGauss)

We now study the performance of the block orthogonalization kernels. Firgure 13 show the breakdown of time needed for CholQR on a single NVIDIA A100 GPU. With our experiment setups, the dense triangular solves (TRSM) required for generating the orthogonal basis vectors took longer than the dense matrix-matrix vector multiply (GEMM) needed to generate the Gram matrix, even though their computational complexity costs are about the same. The time needed to compute the Cholesky factorization of the Gram matrix on the CPU was negligible.

Figure 14a shows the breakdown of the intra-block orthogonalization time on the four NVIDIA A100 GPUs available on a single Perlmutter compute node. Compared to the Gaussian Sketch, the "sketch time" (i.e., the time to apply the sketch) of Count-Sketch was slightly faster, but due to its larger sketch size, it required more time for the global-reduce and local Householder QR. Overall, Count-Gaussian sketch obtained the best performance, but the performance on Perlmutter was largely dominated by the global-reduce and TRSM (and not by the sketching time), and Gaussian and Count-Gaussian sketch obtained similar performance.

Figure 14b then shows the breakdown of the BCGS, with CholQR2, orthogonalization time on the four NVIDIA A100

# nodes	GMRES + ICGS (1659) SpMV	Ortho	Total	*s*-step + CholQR2 (1660) SpMV	Ortho	Total	*s*-step + RandQR (1660) SpMV	Ortho	Total	Two-stage + PIP (1700) SpMV	Ortho	Total	Two-stage + RandBCGS2 (1700) SpMV	Ortho	Total
1	7.96	35.00	40.81	7.38	9.14+8.42	23.95	7.42	9.11+8.55	24.08	8.14	9.87	16.92	9.10	14.69	21.58
					1.99×	1.70×		1.98×	1.69×		3.55×	2.41×		2.38×	1.89×
2	6.26	22.34	26.47	6.28	5.97+4.96	15.65	6.30	6.05+5.04	15.75	6.48	5.41	10.91	7.09	7.85	13.28
					2.04×	1.69×		2.01×	1.68×		4.13×	2.43×		2.85×	1.99×
4	5.13	16.56	19.85	5.08	4.10+3.54	11.45	5.06	4.14+3.66	11.54	5.24	3.59	7.87	5.74	4.92	9.25
					2.17×	1.73×		2.12×	1.72×		4.61×	2.52×		3.37×	2.15×
8	4.47	14.46	17.31	4.47	3.17+2.88	9.43	4.43	3.13+2.99	9.44	4.44	2.54	6.21	4.66	3.09	6.94
					2.39×	1.84×		2.36×	1.93×		5.69×	2.79×		4.68×	2.49×
16	4.41	13.40	15.78	4.06	2.69+2.43	8.24	4.06	2.70+2.51	8.32	4.15	2.24	5.50	4.26	2.66	5.89
					2.62×	1.92×		2.57×	1.90×		5.98×	2.87×		5.04×	2.68×

Table 4: Parallel Strong Scaling of time-to-solution with 7-points 3D Laplace, $n = 300^3$. On each node, we launched 4 MPI processes (one MPI per GPU). The table also shows the speedup gained using s-step ($s = 5$) and two-stage ($\widehat{s} = m$) over standard GMRES ($m = 100$) for orthogonalization and total solution time.

GPUs. It shows the average time required by the s-step GMRES over 600 iterations to orthogonalize the basis vectors with $s = 5$ for solving the 2D Laplace problem of dimension 700^2, and hence the number of columns refers to the GMRES' restart cycle. The inter-block orthogonalization became more significant as the restart-cycle length was increased. However, the TRSM time was still the most dominant part of the overall orthogonalization time.

11.3. s-step GMRES Strong-scaling Results

Although the breakdown of the iteration time depends on the matrix, in Table 4, we show the parallel strong-scaling performance of the s-step GMRES for solving a 3D Laplace problem, from which we can infer the performance for other problems (please see, for instance [6], for the performance using difference matrices). We used the restart length of 100 (i.e., $m = 100$), and considered GMRES to have converged when the relative residual norm is reduced by six orders of magnitude.

For the one-stage algorithm, compared to CholQR, the random sketching had virtually no overhead, while improving the stability of the orthogonalization as shown in Section 9. Compared to the one-stage algorithm, the random sketching had more significant overhead for the two-stage orthogonalization due to the larger sketch size. However, the overhead became less significant as we increased the number of MPI processes, and the latency cost became more significant, with 1.49, 1.45, 1.37, 1.22, and 1.19× overhead on 1, 2, 4, 8, and 16 nodes, respectively.

12. Conclusion

We integrated random sketching techniques into the block orthogonalization process, required for the s-step GMRES. The resulting algorithm ensures that the overall orthogonalization errors are bounded by the machine precision as long as each of the block vectors are numerically full-rank. Our performance results demonstrated that the numerical stability of the block orthogonalization process is improved with a relatively small performance overhead. Our implementation of the random sketching utilizes standard linear algebra kernels such that it is portable to different computer architectures. Though the vendor-optimized versions of these kernels are often available, they may not be optimized for the specific shapes or sparsity patterns of the sketching matrices, and we have observed the sparse sketch often obtains the suboptimal performance. Nevertheless, the sparse random sketching has the complexity of $O(n)$, and with a careful implementation, RandCholQR, using a spares sketch, might not only enhance the numerical stability but also be able to outperform CholQR2.

Appendix

Here we provide a sketch of the proof of Proposition 7.3.

Proof. Because of Proposition 7.1, it is sufficient to prove that the condition number of the big panel $\widehat{Q}_{\ell+1:t+1}$, before the CholQR (on Line 7 of Figure 6), is bounded as

$$\kappa\left(\widehat{Q}_{\ell+1:t+1}\right) = O(1). \tag{16}$$

Then with the bound (7), after the CholQR, the resulting big panel has the orthogonality error,

$$\|I - Q_{\ell+1:t+1}^T Q_{\ell+1:t+1}\| = O(\epsilon),$$

and by an argument identical to [7, Theorem 6.1], the overall stability of the two-stage BCGS2 is ensured with the $O(\epsilon)$ orthogonality error among all the generated basis vectors,

$$\|I - Q_{1:t+1}^T Q_{1:t+1}\| = O(\epsilon),$$

up to a constant term $c(\epsilon, n, \widehat{s})$.

The general strategy for proving (16) is based on the observation that RandBCGS2 is identical to RandHH except that BCGS2-HH (Line 5 of Figure 8b) is used to orthogonalize the sketched vectors $\mathbf{v}_{1:t+1}$, instead of HH (Line 4 of Figure 4a). For instance, the Lines 8–11 of Figure 8b are equivalent to applying the forward-substitution to the big panel $\widehat{Q}_{1:t+1} = V_{1:t+1} R_{1:t+1,1:t+1}^{-1}$. Hence, if we show that the backward error incurred during BCGS2-HH differs from HH by a constant factor $c(\epsilon, n, \widehat{s})$, then we can use the error analysis of RandHH from [33] to prove (16).

More specifically, in [33, Corollary 5.1], it was proven that RandHH results in $\kappa\left(\widehat{Q}_{\ell+1:t+1}\right) = O(1)$. This proof relies on [33, Sections 5.2.1–5.2.8]. Now, the error analysis in the first three subsections [33, Sections 5.2.1–5.2.3] applies to RandBCGS2 because they do not depend on the results of the QR factorization. Furthermore, the analysis of the backward error with the QR factorization in [33, Section 5.2.4] only differs by a constant factor if we prove the backward error of the QR factorization via BCGS2-HH only differs from the Householder

QR backward error by a constant factor. Hence, the remaining error analysis in [33, Sections 5.2.5–5.2.8] applies to RandBCGS2, which uses BCGS2-HH in place of Householder QR, up to a constant factor. Therefore, by [33, Corollary 5.1], we have $\kappa\left(\widehat{Q}_{\ell+1:t+1}\right) = O(1)$ with RandBCGS2, thereby completing the proof.

The main component of the proof is to derive the bound

$$\|\mathbf{v}_{1:j} - \bar{\mathbf{q}}_{1:j} R_{1:j,1:j}\|_2 = O(\epsilon)\|\mathbf{v}_{1:j}\|_2,$$

where $\bar{\mathbf{q}}_{1:j}$ is an *exactly* orthogonal matrix and $R_{1:j,1:j}$ is the upper-triangular matrix computed by BCGS2-HH, since such bound was proved for HH in [33, Section 5.2.4]. Since such result is not directly available in [7], we show how to obtain such a bound below.

According to [7, Theorem 6.1], we first note that our computed matrices $\mathbf{q}_{1:j}$ and $R_{1:j,1:j}$ satisfy

$$\mathbf{v}_{1:j} + \Delta\mathbf{v}_{1:j} = \mathbf{q}_{1:j} R_{1:j,1:j}, \tag{17}$$

$$\|\Delta\mathbf{v}_{1:j}\|_2 = c_5(\epsilon, n, \widehat{s})\|\mathbf{v}_{1:j}\|_2, \tag{18}$$

$$\|I - \mathbf{q}_{1:j}^T \mathbf{q}_{1:j}\|_2 = c_6(\epsilon, n, \widehat{s}), \tag{19}$$

for some constants $c_5(\epsilon, n, \widehat{s})$ and $c_6(\epsilon, n, \widehat{s})$.

Given the singular value decomposition of $\mathbf{q}_{1:j} = U\Sigma V^T$, its perturbation $\Delta\mathbf{q}_{1:j}$ from the exactly-orthonormal vectors $\bar{\mathbf{q}}_{1:j}$ (i.e., $\Delta\mathbf{q}_{1:j} = \bar{\mathbf{q}}_{1:j} - \mathbf{q}_{1:j}$) can be bounded as

$$\begin{aligned} \|\Delta\mathbf{q}_{1:j}\|_2 &= \|\bar{\mathbf{q}}_{1:j} - \mathbf{q}_{1:j}\|_2 \\ &= \|U(I - \Sigma)V^T\|_2 \\ &= \|I - \Sigma\|_2 \le c_7(\epsilon, n, \widehat{s}). \end{aligned} \tag{20}$$

where the singular values of the exactly-orthonormal vectors $\bar{\mathbf{q}}_{1:j}$ are all ones (i.e., $\bar{\mathbf{q}}_{1:j} = UV^T$). Using Weyl's inequality, (19) implies that for each $i \in \{1, \ldots, j\}$,

$$\sqrt{1 - c_6(\epsilon, n, \widehat{s})} \le \Sigma_{i,i} \le \sqrt{1 + c_6(\epsilon, n, \widehat{s})}. \tag{21}$$

Hence, the constant $c_7(\epsilon, n, \widehat{s})$ is given by

$$c_7(\epsilon, n, \widehat{s}) = \max\{\sqrt{1 + c_6(\epsilon, n, \widehat{s})} - 1, 1 - \sqrt{1 - c_6(\epsilon, n, \widehat{s})}\} = O(\epsilon).$$

Now, by substituting $\mathbf{q}_{1:j} = \bar{\mathbf{q}}_{1:j} - \Delta\mathbf{q}_{1:j}$ into (17), we obtain

$$\bar{\mathbf{q}}_{1:j} R_{1:j,1:j} = \mathbf{v}_{1:j} + \Delta\mathbf{v}_{1:j} + \Delta\mathbf{q}_{1:j} R_{1:j,1:j},$$

which allows us to derive the bound,

$$\|R_{1:j,1:j}\|_2 \le \frac{1 + c_5(\epsilon, n, \widehat{s})}{1 - c_7(\epsilon, n, \widehat{s})}\|\mathbf{v}_{1:j}\|_2,$$

because

$$\begin{aligned} &(1 - c_7(\epsilon, n, \widehat{s}))\|R_{1:j,1:j}\|_2 \\ &= (1 - c_7(\epsilon, n, \widehat{s}))\|\bar{\mathbf{q}}_{1:j} R_{1:j,1:j}\|_2 \\ &= \|\mathbf{v}_{1:j} + \Delta\mathbf{v}_{1:j} + \Delta\mathbf{q}_{1:j} R_{1:j,1:j}\|_2 - c_7(\epsilon, n, \widehat{s})\|R_{1:j,1:j}\|_2 \\ &\le \|\mathbf{v}_{1:j}\|_2 + \|\Delta\mathbf{v}_{1:j}\|_2 + (\|\Delta\mathbf{q}_{1:j}\|_2 - c_7(\epsilon, n, \widehat{s}))\|R_{1:j,1:j}\|_2 \\ &\le \|\mathbf{v}_{1:j}\|_2 + \|\Delta\mathbf{v}_{1:j}\|_2 \qquad (22) \\ &\le \|\mathbf{v}_{1:j}\|_2 + c_5(\epsilon, n, \widehat{s})\|\mathbf{v}_{1:j}\|_2 \qquad (23) \\ &= (1 + c_5(\epsilon, n, \widehat{s}))\|\mathbf{v}_{1:j}\|_2. \end{aligned}$$

where the inequalities (22) and (23) follows due to $\|\Delta\mathbf{q}_{1:j}\|_2 \le c_7(\epsilon, n, \widehat{s})$ and $\|\Delta\mathbf{v}_{1:j}\|_2 = c_5(\epsilon, n, \widehat{s})\|\mathbf{v}_{1:j}\|_2$ from (20) and (17), respectively.

Thus, we finally have

$$\begin{aligned} &\|\Delta\mathbf{v}_{1:j} + \Delta\mathbf{q}_{1:j} R_{1:j,1:j}\|_2 \\ &\le \|\Delta\mathbf{v}_{1:j}\|_2 + \|\Delta\mathbf{q}_{1:j}\|_2\|R_{1:j,1:j}\|_2 \\ &\le \left(c_5(\epsilon, n, \widehat{s}) + c_7(\epsilon, n, \widehat{s})\frac{1 + c_5(\epsilon, n, \widehat{s})}{1 - c_7(\epsilon, n, \widehat{s})}\right)\|\mathbf{v}_{1:j}\|_2 \\ &= c_8(\epsilon, n, \widehat{s})\|\mathbf{v}_{1:j}\|_2, \end{aligned} \tag{24}$$

where $c_8(\epsilon, n, \widehat{s}) = O(\epsilon)$.

Therefore, there is an exactly-orthogonal matrix $\bar{\mathbf{q}}_{1:j}$ such that the backward error $\Delta\bar{\mathbf{v}}_{1:j}$ from BCGS2-HH satisfies

$$\begin{aligned} &\mathbf{v}_{1:j} + \Delta\bar{\mathbf{v}}_{1:j} = \bar{\mathbf{q}}_{1:j} R_{1:j,1:j}, \\ &\text{with } \|\Delta\bar{\mathbf{v}}_{1:j}\|_2 = c_8(\epsilon, n, \widehat{s})\|\mathbf{v}_{1:j}\|_2. \end{aligned} \tag{25}$$

In contrast, if we compute the Householder QR of $\mathbf{v}_{1:j}$, the standard backward error analysis [40, Theorem 19.4] gives

$$\begin{aligned} &\mathbf{v}_{1:j} + \Delta\tilde{\mathbf{v}}_{1:j} = \tilde{\mathbf{q}}_{1:j}\tilde{R}_{1:j,1:j}, \\ &\text{with } \|\Delta\tilde{\mathbf{v}}_{1:j}\|_2 \le c_9(\epsilon, n, \widehat{s})\|\mathbf{v}_{1:j}\|_2, \end{aligned}$$

where $\tilde{R}_{1:j,1:j}$ is the upper-triangular matrix computed by the HH factorization in finite precision, $\tilde{\mathbf{q}}_{1:j}$ is exactly-orthogonal, and $c_9(\epsilon, n, \widehat{s}) = O(\epsilon)$ is a constant. In other words, the backward error from BCGS2-HH and Householder QR are both bounded by some $O(\epsilon)\|\mathbf{v}_{1:j}\|_2$ terms, and therefore they only differ by a constant factor $c(\epsilon, n, \widehat{s})$. □

Acknowledgment

This work was supported by the 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, by the U.S. Department of Energy, Office of Science, Office of Advanced Scientific Computing Research, Scientific Discovery through Advanced Computing (SciDAC) Program through the FASTMath Institute under Contract No. DE-AC02-05CH11231 at Sandia National Laboratories, and by Sandia Laboratory Directed Research and Development (LDRD). Sandia National Laboratories is a multimission laboratory managed and operated by National Technology and Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International, Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA-0003525. This paper describes objective technical results and analysis. Any subjective views or opinions that might be expressed in the paper do not necessarily represent the views of the U.S. Department of Energy or the United States Government.

References

[1] Y. Saad, M. H. Schultz, GMRES: a generalized minimal residual algorithm for solving nonsymmetric linear systems, SIAM J. Sci. Statist. Comput. 7 (1986) 856–869.

[2] E. Carson, Communication-avoiding Krylov subspace methods in theory and practice, Ph.D. thesis, EECS Dept., U.C. Berkeley (2015).

[3] M. Hoemmen, Communication-avoiding Krylov subspace methods, Ph.D. thesis, EECS Dept., U.C. Berkeley (2010).

[4] E. de Sturler, H. van der Vorst, Reducing the effect of global communication in GMRES(m) and CG on parallel distributed memory computers, Applied Numer. Math. 18 (1995) 441–459.

[5] W. Joubert, G. F. Carey, Parallelizable restarted iterative methods for nonsymmetric linear systems. II: parallel implementation, Int. J. Comput. Math. 44 (1992) 269–290.

[6] I. Yamazaki, A. J. Higgins, E. G. Boman, D. B. Szyld, Two-stage block orthogonalization to improve performance of *s*-step GMRES, in: 2024 IEEE International Parallel and Distributed Processing Symposium (IPDPS), 2024, pp. 26–37.

[7] J. L. Barlow, Reorthogonalized block classical Gram–Schmidt using two Cholesky-based TSQR algorithms, SIAM Journal on Matrix Analysis and Applications 45 (3) (2024) 1487–1517.

[8] A. Stathopoulos, K. Wu, A block orthogonalization procedure with constant synchronization requirements, SIAM J. Sci. Comput. 23 (2002) 2165–2182.

[9] O. Balabanov, Randomized Cholesky QR factorizations, arXiv:2210.09953 (2022).

[10] A. J. Higgins, D. B. Szyld, E. G. Boman, I. Yamazaki, Analysis of randomized Householder-Cholesky QR factorization with multisketching (2023). arXiv:2309.05868.

[11] I. Yamazaki, A. J. Higgins, E. G. Boman, D. B. Szyld, Integrating random sketching into BCGS2 for *s*-step GMRES, Tech. Rep. 2024-03134C, Sandia National Laboratories, SIAM Conference on Parallel Processing for Scientific Computing (2024).

[12] D. P. Woodruff, Sketching as a tool for numerical linear algebra, Foundations and Trends in Theoretical Computer Science 10 (2014) 1–157.

[13] The Trilinos Project Team, The Trilinos Project Website. URL `https://trilinos.github.io`

[14] O. Balabanov, L. Grigori, Randomized block Gram–Schmidt process for the solution of linear systems and eigenvalue problems, SIAM Journal on Scientific Computing 47 (1) (2025) A553–A585.

[15] O. Balabanov, L. Grigori, Randomized Gram-Schmidt process with application to GMRES, SIAM J. Sci. Comput. 44 (3) (2022) A1450–A1474.

[16] E. Timsit, O. Balabanov, L. Grigori, Randomized orthogonal projection methods for Krylov subspace solvers, arXiv:2302.07466 (2023).

[17] J. Demmel, L. Grigori, M. Hoemmen, J. Langou, Communication-optimal parallel and sequential QR and LU factorizations, SIAM J. Sci. Comput. 34 (2012) A206–A239.

[18] M. Anderson, G. Ballard, J. Demmel, K. Keutzer, Communication-avoiding QR decomposition for GPUs, in: 2011 IEEE International Parallel & Distributed Processing Symposium, 2011.

[19] T. Mary, I. Yamazaki, J. Kurzak, P. Luszczek, S. Tomov, J. Dongarra, Performance of random sampling for computing low-rank approximations of a dense matrix on gpus, in: Proc. Int. Conf. High Perf. Comput., Netw., Stor. and Anal. (SC), 2015, pp. 60:1–60:11.

[20] K. Swirydowicz, Random sketching for improving the performance of Gram-Schmidt algorithm on modern architectures, talk presented at the 2023 SIAM Conference on Computational Science and Engineering, Amsterdam, February 27–March 3, 2013.

[21] Y. Nakatsukasa, J. A. Tropp, Fast & accurate randomized algorithms for linear systems and eigenvalue problems, arXiv:2111.00113 (2021).

[22] T. Sarlos, Improved approximation algorithms for large matrices via random projections, in: 47th Annual IEEE Symposium on Foundations of Computer Science (FOCS'06), 2006, pp. 143–152.

[23] P. Martinsson, J. Tropp, Randomized numerical linear algebra: Foundations and algorithms, Acta Numerica 29 (2020) 403–572.

[24] M. A. Heroux, R. A. Bartlett, V. E. Howle, R. J. Hoekstra, J. J. Hu, T. G. Kolda, R. B. Lehoucq, K. R. Long, R. P. Pawlowski, E. T. Phipps, A. G. Salinger, H. K. Thornquist, R. S. Tuminaro, J. M. Willenbring, A. Williams, K. S. Stanley, An overview of the Trilinos project, ACM Trans. Math. Softw. 31 (3) (2005) 397–423.

[25] Z. Bai, D. Hu, L. Reichel, A Newton basis GMRES implementation, IMA J. Numer. Anal. 14 (4) (1994) 563–581.

[26] M. Mohiyuddin, M. Hoemmen, J. Demmel, K. Yelick, Minimizing communication in sparse matrix solvers, in: Proc. Int. Conf. High Perf. Comput., Netw., Stor. and Anal. (SC), 2009, pp. 36:1–36:12.

[27] L. Grigori, S. Moufawad, Communication Avoiding ILU0 Preconditioner, SIAM J. Sci. Comput. 37 (2015) C217–C246.

[28] I. Yamazaki, S. Rajamanickam, E. Boman, M. Hoemmen, M. Heroux, S. Tomov, Domain Decomposition Preconditioners for Communication-avoiding Krylov Methods on a Hybrid CPU-GPU Cluster, in: Proc. Int. Conf. High Perf. Comput., Netw., Stor. and Anal. (SC), 2014, pp. 933–944.

[29] A. Greenbaum, M. Rozložnik, Z. Strakoš, Numerical behaviour of the modified Gram-Schmidt GMRES implementation, BIT Numer. Math. 37 (1997) 706–719.

[30] E. Carson, K. Lund, M. Rozložník, The stability of block variants of classical Gram–Schmidt, SIAM J. Matrix Anal. Appl. 42 (2021) 1365–1380.

[31] Y. Yamamoto, Y. Nakatsukasa, Y. Yanagisawa, T. Fukaya, Roundoff error analysis of the Cholesky QR2 algorithm, Electronic Transactions on Numerical Analysis 44 (2015) 306–326.

[32] Y. Yamamoto, Y. Nakatsukasa, Y. Yanagisawa, T. Fukaya, Roundoff error analysis of the Cholesky QR2 algorithm, Electronic Trans. Numer. Anal. 44 (2015) 306–326.

[33] A. J. Higgins, D. B. Szyld, E. G. Boman, I. Yamazaki, Analysis of randomized Householder-Cholesky QR factorization with multisketching (2025). arXiv:2309.05868.

[34] K. L. Clarkson, D. P. Woodruff, Low rank approximation and regression in input sparsity time, STOC '13, Association for Computing Machinery, New York, NY, USA, 2013, p. 81–90. doi:10.1145/2488608.2488620. URL `https://doi.org/10.1145/2488608.2488620`

[35] I. Yamazaki, S. J. Thomas, M. Hoemmen, E. G. Boman, K. Swirydowicz, J. J. Elliott, Low-synchronization orthogonalization schemes for *s*-step and pipelined Krylov solvers in Trilinos, in: Proc. of SIAM Conf. Parallel Processing for Sci. Comput., 2020, pp. 118–128.

[36] A. Smoktunowicz, J. L. Barlow, J. Langou, A note on the error analysis of classical Gram-Schmidt, Numer. Math. 105 (2006) 299–313.

[37] C. Baker, M. Heroux, Tpetra, and the use of generic programming in scientific computing, Scientific Programming 20 (2) (2012).

[38] The Tpetra Project Team, The Tpetra Project Website. URL `https://trilinos.github.io/tpetra.html`

[39] S. Rajamanickam, S. Acer, L. Berger-Vergiat, V. Dang, N. Ellingwood, E. Harvey, B. Kelley, C. R. Trott, J. Wilke, I. Yamazaki, Kokkos Kernels: Performance portable sparse/dense linear algebra and graph kernels (2021). arXiv:2103.11991.

[40] N. J. Higham, Accuracy and Stability of Numerical Algorithms, 2nd Edition, Society for Industrial and Applied Mathematics, Philadelphia, 2002. doi:10.1137/1.9780898718027.