Received: 16 November 2018 | Revised: 13 September 2020 | Accepted: 4 October 2020

DOI: 10.1002/nla.2343

RESEARCH ARTICLE

WILEY

Low synchronization Gram–Schmidt and generalized minimal residual algorithms

Katarzyna Świrydowicz[1] | **Julien Langou**[2] | **Shreyas Ananthan**[1] | **Ulrike Yang**[3] | **Stephen Thomas**[1]

[1]National Renewable Energy Laboratory, Golden, Colorado

[2]Department of Mathematics, University of Colorado Denver, Denver, Colorado

[3]Lawrence Livermore National Laboratory, Livermore, California

Correspondence
Stephen Thomas, National Renewable Energy Laboratory, 15013 Denver W Pkwy, Golden, CO 80401.
Email: Stephen.Thomas@nrel.gov

Funding information
Exascale Computing Project, Grant/Award Number: 17-SC-20-SC; U.S. Department of Energy, Grant/Award Numbers: DE-AC36-08GO28308, DE-AC52-07NA27344; NSF, Grant/Award Number: 1645514

Summary

The Gram–Schmidt process uses orthogonal projection to construct the $A = QR$ factorization of a matrix. When Q has linearly independent columns, the operator $P = I - Q(Q^TQ)^{-1}Q^T$ defines an orthogonal projection onto $Q^{\perp}$. In finite precision, Q loses orthogonality as the factorization progresses. A family of approximate projections is derived with the form $P = I - QTQ^T$, with correction matrix T. When $T = (Q^TQ)^{-1}$, and T is triangular, it is postulated that the best achievable orthogonality is $\mathcal{O}(\varepsilon)\kappa(A)$. We present new variants of modified (MGS) and classical Gram–Schmidt algorithms that require one global reduction step. An interesting form of the projector leads to a compact WY representation for MGS. In particular, the inverse compact WY MGS algorithm is equivalent to a lower triangular solve. Our main contribution is to introduce a backward normalization lag into the compact WY representation, resulting in a $\mathcal{O}(\varepsilon)\kappa([r_0,\ AV_m])$ stable Generalized Minimal Residual Method (GMRES) algorithm that requires only one global reduce per iteration. Further improvements in performance are achieved by accelerating GMRES on GPUs.

KEYWORDS

Graphics processing unit, Gram–Schmidt process, Krylov methods, massively parallel, scalable GMRES, WY factorization

1 | INTRODUCTION

Krylov solvers are common in applications such as fluid flow simulations based on finite element discretizations of partial differential equations (PDEs). These solvers account for a substantial amount of the total run-time (e.g., ≈30% in Thomas et al.[1]). The focus of this paper is on improving the performance of the Generalized Minimal Residual Method (GMRES).[2] For nonsymmetric matrices, this Krylov solver is often preferred due to its stability and robustness. At the same time, GMRES is hard to parallelize because of the orthogonalization process. We focus our efforts on reformulating Gram–Schmidt orthogonalization algorithms to minimize the number of global reductions and synchronizations.

Let A be an $n \times n$ matrix with real entries and let b be an $n \times 1$ vector. Let x_0 be an initial guess vector. We define the initial residual $r_0 = b - Ax_0$ and $v_1 = r_0/\|r_0\|_2$. The GMRES algorithm for iteratively solving a linear system $Ax = b$ computes the Arnoldi QR factorization of the matrix $[\ r_0,\ AV_m\ ]$, where the columns of Q are the Krylov vectors $v_1,\ v_2,\ \ldots,\ v_{m+1}$, formed by adding a new column to the matrix V_m at each iteration. The Gram–Schmidt algorithm constructs

Abbreviations: CWY, compact WY factorization; GMRES, Generalized Minimal Residual Method; GS, Gram–Schmidt orthogonalization algorithm; HH, Housholder

an orthogonal basis from a set of linearly independent vectors spanning a vector space. The numerical properties of the classical (CGS) and modified (MGS) Gram–Schmidt algorithms for the linear least squares problem were elucidated by Björck,[3] who determined the representation error for the QR factorization of an $n \times m$ matrix A.[1] The columns of Q lose orthogonality in both cases and bounds on $\|I - Q^T Q\|_2$ have been derived and improved by several authors. For the modified algorithm, Björck[3] derived a bound that depends on the condition number $\kappa(A)$. The situation is worse for the classical algorithm where the loss of orthogonality depends on $\kappa^2(A)$ as shown in the papers of Giraud et al.[4,5] For both CGS and MGS, the orthogonality can be restored to the level of machine round-off by reorthogonalization of the columns of Q. Iterative reorthogonalization was discussed in Leon et al.[6] along with the "twice is enough" result of Kahan et al.[5] A subsequent paper by Smoktunowicz et al.[7] introduced a more stable Cholesky-like CGS algorithm using a Pythagorean identity for the normalization of the basis vectors.

We present low synchronization variants of MGS and CGS algorithms that require one global reduction step. CGS-2 (CGS with re-orthogonalization) recursively applies the projector $P^{IC} = I - Q_{j-1} T^{IC}_{j-1} Q^T_{j-1}$, whereas MGS applies the elementary projectors $P^{(j)} = I - q_j q_j^T$. The sequence of elementary projections in MGS may be represented as an inverse compact WY matrix $P^{IM} = I - Q_{j-1} T^{IM}_{j-1} Q^T_{j-1}$, where the $m \times m$ lower triangular matrix $T^{IM}_{j-1} = (\, I + L_{j-1}\,)^{-1}$ is applied as a triangular solver recurrence. We believe that this result, that is, representing MGS as triangular solve, is novel as it has not appeared in the literature. Several authors have been close to this formulation, however, we are unaware of an MGS implementation that rests upon a lower triangular solve. In particular, Björck anticipated the inverse compact WY representation in lemma 5.1 and the corollary appearing in his seminal paper.[3] Ruhe[8] demonstrated that the MGS algorithm is equivalent to Gauss–Seidel iteration, and was very close to the triangular solve idea but this was not explicitly presented in his paper. Walker[9] presents the Householder GMRES and makes the important link between products of orthogonal reflectors $(I - qq^T)$ and the lower triangular matrix solver recurrence and applied this to Householder transformations, possibly for the first time. This leads directly to the compact WY inverse representation of Householder QR. Chiara Puglisi[10] and Joffrain et al.[11] noted the inverse lower triangular matrix relationship in the Householder Compact WY (HH-CWY), published in Schreiber and Van Loan,[12] and also the work of Christian Bischoff.[13]

Björck[14] derived the compact WY representation for MGS, which also appears in Malard and Paige;[15] they noted the Householder connection HH-CWY and recursive construction of the triangular matrix T. Leon, Björck, Gander in the *100 years of Gram–Schmidt* paper[6] (NLAA) did not describe a lower triangular solver variant of MGS in a review of various CGS and MGS forms (row, column) and re-orthogonalization. Sun[16] notes the utility of the factor form of aggregation kernels for T^{-1} in the analysis by Björck and Paige.[17] A series of papers by Paige,[18] Paige and Wülling,[19] and Paige[20] lead to a loss of orthogonality matrix S depending on the triangular T matrix . These papers expand on the work of Giraud, Gratton and Langou[21] and further develop the analysis. The important paper of Paige, Rozložnik and Strakoš[22] establishes the backward stability of MGS-GMRES and discusses the normwise relative backward error stopping criterion Paige and Strakoš (2002).[23]

Following Hernandez et al.,[24,25] we introduce a lag into the CGS-2 algorithm to delay reorthogonalization and normalization (in both CGS-2 and MGS) of the diagonal elements of R. The lagged normalization was first proposed in Kim and Chronopoulos[26] for the s-step Arnoldi eigenvalue algorithm. In addition, the reductions can be overlapped with the computation and pipelined as demonstrated by Ghysels et al.[27] and Yamazaki et al.[28,29] Derivations of low communication algorithms for Gram–Schmidt orthogonalization depend on Björck[3] and Ruhe.[8] The loss of orthogonality relationship for the matrix $Q^T Q$ is also exploited to reduce the computation and communication overhead by computing one column of $L^T_{j-1} = Q^T_{j-1} q_j$ at each Gram–Schmidt iteration. Thus, our CGS-2 and inverse compact WY MGS algorithms reduce the number of global reductions per iteration to only one.

In this paper, we assume that every vector is distributed across P processors (MPI ranks) by rows. To illustrate the distribution pattern, let $w_i = [w_{i1}, \ldots, w_{in}]^T$. Processor 1 owns a part of this vector, say $[w_{i1}, \ldots, w_{ia_1}]^T$, processor 2 owns the data $[w_{i(a_1+1)}, \ldots, w_{ia_2}]^T$, and so forth, and processor P owns the data $[w_{i(a_{P-1}+1)}, \ldots, w_{in}]^T$. If $w_i^T w_i$ is computed, each processor computes the partial inner product and communicates it to the remaining processors. This procedure, known as a global reduction, requires *global communication*, as the values are collected from every processor and added. However, not all such operations require global communication. For example, scaling a vector by scalar and AXPY can be executed locally (individually by each processor, on the data it owns) and no communication is needed. The operations that do not require global communication generally scale well on distributed systems.

[1] In this paper, A either denotes a square matrix for the linear system $Ax = b$ or a nonsquare matrix for the QR decomposition.

Both the CGS and MGS algorithms employ vector inner products and norms to construct an orthonormal set of basis vectors. On modern multicore computers global reductions use a tree-like sum of scalar products involving every core (MPI rank) on every node of the machine. The cost of this operation grows with the height of the tree determined by the number of cores. As mentioned previously, the remaining operations required in the orthogonalization are scalable. These consist of the mass inner product $v = X^T y$, vector MAXPY operation $w = w - VH_{:,i}$, and point-to-point sparse matrix-vector product (SpMV) at each iteration. For sparse matrices distributed by rows, the parallel SpMV typically requires local communication between cores. The vector sum is trivially data-parallel. Hence, the global reductions are the limiting factor for the available parallelism in the algorithm.

This paper is organized as follows. Section 2 describes the CGS and MGS algorithms and orthogonal projectors suitable for parallel computation. The compact *WY* representation is also discussed. The third section is devoted to the GMRES algorithm and GPU implementation based on low-synch Gram–Schmidt algorithms. The resulting algorithms replace standard MGS inside GMRES, which is discussed in Section 4. Section 5 describes numerical results.

1.1 | Notation

In this manuscript, standard notation is adopted from classical numerical linear algebra textbooks such as Stewart.[30] Small roman characters denote vectors, that is, q, q_j, and a are column vectors with n entries. When an algorithm updates the vectors in an iterative fashion, superscripts with brackets are employed, that is, $q^{(1)}, q^{(2)}, \ldots, q^{(k)}, \ldots$ are subsequent iterates. Capital letters denote matrices, such as Q, R, L, and A. If the number of columns in the matrix is discussed, this is indicated with a subscript. For example, Q_j is a matrix with n rows and j columns. The ith column of a matrix is indicated with a small letter with a superscript, thus a_i is the ith column of A and q_j is the jth column of Q, which agrees with the notation for vectors. The double subscript ij denotes an entry of the matrix in row i and column j, hence r_{ij} denotes the entry of R in row i and column j. In most algorithms, capital R or U are used for upper triangular matrices (such as R in standard QR factorization) and capital L is used for lower triangular matrices. In addition, throughout this manuscript, $\kappa(A)$ denotes the condition number of matrix A and ε is the machine epsilon.

In certain places in the paper, if the dimensions are important in the discussion or submatrices are discussed, we use *Matlab notation*. For example, $R_{1:k,j-1:j-2}$ denotes submatrix of R consisting of rows from 1 to k and columns from $j-1$ to $j-2$. Colon is used to denote columns and rows, hence $A_{:,j}$ is jth column of j and $U_{i,:}$ is ith row of U.

2 | GRAM–SCHMIDT ORTHOGONALIZATION

Let A be an $n \times m$ ($n \geq m$) nonsingular matrix with QR factorization obtained from the Gram–Schmidt algorithm. Consider one step of the CGS algorithm, with $(j-1)$ orthonormal vectors forming an $n \times (j-1)$ matrix Q_{j-1}, and the vector $a_j := A_{:,j}$, which is to be orthogonalized to produce the jth column $q_j := Q_{:,j}$. The CGS algorithm with re-orthogonalization is presented as Algorithm 1. For each k, a re-orthogonalization is applied (steps 3–5). The notation in Step 7 means that the final vector $a^{(k)}$ is normalized to produce the next column q_j of Q. The normalization coefficient is placed in r_{jj}, leading to the corresponding matrix factors where R_{j-1}, is an $(j-1) \times (j-1)$ upper triangular matrix.

Algorithm 1. CGS Algorithm with re-orthogonalization

Input: Matrices Q_{j-1} and R_{j-1},$(j-1) \times (j-1)$, such that $A_{j-1} = Q_{j-1}R_{j-1}$; column vector a_j
Output: $Q_j = [Q_{j-1},\ q_j]$ and R_j such that $A_j = Q_j R_j$

1: Start $a^{(0)} = a_j$, $r_{1:j-1,j} = 0$.
2: **for** $k = 1, 2, \ldots$ **do**
3: $\quad s^{(k-1)} = Q_{j-1}^T a^{(k-1)}$
4: $\quad r_{1:j-1,j}^{(k)} = r_{1:j-1,j}^{(k-1)} + s^{(k-1)}$
5: $\quad a^{(k)} = a^{(k-1)} - Q_{j-1}\, s^{(k-1)}$
6: **end for**
7: $q_j = a^{(k)} / r_{j,j} = \|a^{(k)}\|_2$
8: $r_j = [r_{1:j-1,j}\ r_{jj}]^T$

$$R_j = \begin{bmatrix} R_{j-1} & r_{1:j-1,j} \\ & r_{jj} \end{bmatrix}, \quad Q_j = \begin{bmatrix} Q_{j-1}, & q_j \end{bmatrix},$$

$r_{1:j-1,j}$ is a $(j-1)\times 1$ vector ($r_j = [r_{1:j-1,j}\ r_{jj}]^T$) and Q_{j-1} is an $n\times(j-1)$ matrix.

Both row and column variants of the algorithm for updates of the R matrix elements exist and are numerically equivalent for the MGS algorithm, as they result in the same representation error Björck,[14] Remark 1, page 300. MGS applies orthogonal projectors as elementary matrices $P^{(j)} = (\ I - q_j\ q_j^T\ )$. For the column-wise MGS algorithm, each iteration k consists of $(j-1)$ updates (in steps 2-5) of Algorithm 2.

Algorithm 2. MGS Algorithm

Input: Matrices Q_{j-1} and R_{j-1}, such that $A_{j-1} = Q_{j-1}R_{j-1}$; column vector a_j
Output: $Q_j = [Q_{j-1},\ q_j]$ and R_j, such that $A_j = Q_jR_j$

1: Start $a^{(0)} = a_j$, $r_j = 0$.
2: **for** $i = 1, \ldots, j-1$ **do**
3: $\quad r_{i,j} = q_i^T a^{(i)}$
4: $\quad a^{(i+1)} = a^{(i)} - r_{i,j}\ q_i$
5: **end for**
6: $q_j = a^{(j)}$
7: $r_{jj} = \|q_j\|_2$
8: $q_j = q_j / r_{jj}$

2.1 | Compact *WY* matrix representation

The product of elementary matrices for the Householder QR factorization can be represented as a compact WY matrix, originally derived by Schreiber and Van Loan[12]

$$P^H = I - YT^HY^T,$$

where P^H is an orthogonal matrix, T^H is unit upper triangular and $Y_{j-1} = [w_1, \ \ldots \ w_{j-1}]$, is a lower trapezoidal matrix with Householder $H = I - w_{j-1}\ w_{j-1}^T$ vectors w_{j-1} as columns. The matrix T^H is computed recursively. Puglisi,[10] introduces an inverse compact WY representation for Householder transformations with a lower triangular matrix T^{IH} and notes that updates may employ the inverse T^{-1} or lower triangular solves. The lower triangular form of the compact WY representation for MGS was presented in Björck.[14] Here the MGS projection matrix P^M is equivalent to the product of orthogonal projectors $P^{(k)} = I - q_k\ q_k^T$, $k = 1, \ \ldots \ , j-1$, $P^M = P^{(j-1)} \ \ldots \ P^{(1)}$,

$$P^M = I - Q_{j-1}\ T_{j-1}^M\ Q_{j-1}^T,$$

where Q_{j-1} is an orthogonal matrix and T_{j-1}^M is lower triangular. The upper triangular form of T^H for Householder and MGS was presented in Malard and Paige.[15] They demonstrate how the matrix T_{j-1}^M can be generated recursively as follows

$$T_0^M = 1$$
$$T_{j-1}^M = \begin{bmatrix} T_{j-2}^M & -T_{j-2}^M\ Q_{j-2}^T\ q_{j-1} \\ 0 & 1 \end{bmatrix}$$

In this case the product of orthogonal matrices is reversed $P^{(k)} = I - q_k\ q_k^T$, $k = j-1, \ \ldots \ , 1$, $P^M = P^{(1)} \ \ldots \ P^{(j-1)}$.

Walker[9] employed a lower triangular matrix inverse for the compact WY Householder algorithm and recognized that the product of Householder reflectors is equivalent to a lower triangular solver recurrence. An inverse T^{IH} matrix form of

TABLE 1 Orthogonal projections and reflections. Number of synchronizations

Algorithm	Projection/reflection	*T* Matrix	Synchs
CGS	$P^C = I - Q_{j-1}Q_{j-1}^T$	$T^C = I$	1
CGS-2	$P^{IC} = I - Q_{j-1}T_{j-1}^{IC}Q_{j-1}^T$	$L_{:,j-1}^T = Q_{j-1}^T q_{j-1}$, $T_{j-1}^{IC} = I - L_{j-1} - L_{j-1}^T$	1
MGS-CWY	$P^M = I - QT^MQ^T$	$t_{:,j-1} = Q_{j-1}^T q_{j-1}$, $t_{:,j-1} = -T_{j-2}^M\, t_{:,j-1}$, $T_{j-1,j-1}^M = 1$	1
MGS-ICWY	$P^{IM} = I - Q_{j-1}T_{j-1}^{IM}Q_{j-1}^T$	$L_{:,j-1}^T = Q_{j-1}^T q_{j-1}$, $T_{j-1}^{IM} = (I + L_{j-1})^{-1}$	1
HH-CWY	$P^H = I - YT^HY^T$	$T_{:,j-1}^H = w_{j-1}^T\, Y_{j-2}\, T_{j-2}^H$, $T_{j-1,j-1}^H = 1$	2
HH-ICWY	$P^{IH} = I - YT^{IH}Y^T$	$T^{-1} + T^{-T} = Y^TY$, $T_{j-1,j-1}^{-1} = w_{j-1}^T w_{j-1}/2$, $T_{i,j}^{-1} = w_i^T w_j$	2

the Householder compact *WY* matrix, referred to as the *UT* transform, was derived by Joffrain et al.[11] The inverse compact *WY* representation for Householder and Gram–Schmidt is also discussed in Sun,[16] who derives the matrix inverse form of T^H for Householder and notes the utility of the factor form of aggregation kernels for T^{-1} in the analysis by Björck and Paige.[17]

The inverse of the compact *WY* matrix can be written using the lower triangular matrix, $T_{j-1}^{IM} = (\, I + L_{j-1}\,)^{-1}$. The matrix $U_{j-1} = L_{j-1}^T$ can be computed one column at a time at the *j*th iteration for column *j* of the MGS algorithm. The elements of the strictly upper triangular matrix U_{j-1} contain the inner products $u_{ij} = q_i^T q_j$, $j > i$, and $u_{ij} = 0$, $j \le i$. A summary of the different compact *WY* (CWY) and inverse compact *WY* (ICWY) reflectors for Householder (HH) and Gram–Schmidt projectors are given in Table 1. The number of global synchronizations (global reductions) for the parallel implementations are also provided when combined with a lagged normalization.

2.2 | Classical Gram–Schmidt

In the CGS algorithm, the projection matrix P^C is applied as a matrix-vector product $Q_{j-1}^T a_j$ to the *j*th column of the matrix *A*, followed by a second matrix-vector product and vector subtraction.

$$P^C = I - Q_{j-1}\, Q_{j-1}^T. \tag{1}$$

The matrix T^C is the identity in this projector. In this case, the resulting loss of orthogonality between the columns of Q_{j-1} can be severe, where $\|I - Q^TQ\| \approx \mathcal{O}(\varepsilon)\kappa^2(A)$. In order to achieve $\mathcal{O}(\varepsilon)$, the algorithm was traditionally iterated twice to re-orthogonalize the columns of *Q*.

Ruhe[8] explicitly unrolled the loop recurrence as in Appendix A. Expanding this idea, for two iterations of CGS, the projector takes the form

$$P^{IC} = I - [\, Q_{j-2},\, q_{j-1} - Q_{j-2}\, Q_{j-2}^T\, q_{j-1}\,]\, T_{j-1}^{IC}\, [\, Q_{j-2},\, q_{j-1}\,]^T. \tag{2}$$

The small $(j-1)\times(j-1)$ correction matrix T_{j-1}^{IC} can be constructed recursively, as follows

$$(\, Q_{j-1}^T Q_{j-1}\,)^{-1} = T_{j-1}^{IC} = I - L_{j-1} - L_{j-1}^T, \tag{3}$$

with the strictly lower triangular matrix L_{j-1} at iteration $j-1$. For our recursive formulation of the CGS-2 algorithm, the $(j-1)\times(j-1)$ correction matrix T_{j-1}^{IC} is split and applied over two iterations and in the first step, takes the form

$$T_{j-1}^{IC} = \begin{bmatrix} I & T_{:,j-2}^{IC} \\ T_{j-2,:}^{IC} & t_{j-1,j-1} \end{bmatrix} = \begin{bmatrix} I & 0 \\ -w^T\, \gamma^{-1} & 1 \end{bmatrix}, \tag{4}$$

with $w^T\gamma^{-1} = L_{j-2,:}$. Thus we have, $r_{1:j-2,j} = z$, where

$$\left[w, \quad z\right] = \left[Q_{j-2}^T\, q_{j-1}, \quad Q_{j-2}^T\, a_j\right], \quad \left[r_{j-1,j-1}, \quad r_{j-1,j}\right] = \left[(\, q_{j-1}^T q_{j-1} - w^T w\,), \quad (\, q_{j-1}^T\, a_j - w^T z\,)\right]. \tag{5}$$

To avoid cancellation errors, the diagonal element $\gamma = r_{j-1,j-1}^{1/2}$ is obtained from the Pythagorean identity

$$q_{j-1}^T q_{j-1} - w^T w = (\, \|q_{j-1}\|_2 - \|w\|_2\,)\,(\, \|q_{j-1}\|_2 + \|w\|_2\,). \tag{6}$$

The update of column R_{j-1}, corresponding to L_{j-1}^T, is also lagged to iteration $j-1$ and follows from the update in Step 7 in Algorithm 1 above and the development in Appendix A.

$$r_{1:j-2,j-1} = r_{1:j-2,j-1} + w = r_{1:j-2,j-1} + Q_{j-2}^T q_{j-1}. \tag{7}$$

Here, the lagged correction is given by the matrix-vector product $w = Q_{j-2}^T q_{j-1}$. Only one global communication step is needed to compute the vectors w and z, together with the inner products $q_{j-1}^T q_{j-1}$ and $q_{j-1}^T\, a_j$. For the projector P^{IC}, the matrix $Q_{j-2}\, Q_{j-2}^T$ is never explicitly formed and instead appears within the local inner products

$$\left[w^T w, \quad w^T z\right] = \left[q_{j-1}^T\, Q_{j-2}\, Q_{j-2}^T\, q_{j-1}, \quad q_{j-1}^T\, Q_{j-2}\; Q_{j-2}^T\, a_j\right]. \tag{8}$$

The re-orthogonalization step has been lagged or delayed together with the normalization as proposed in Hernandez et al.[24] In effect, a pipeline is created by the introduction of a delayed update. However, the authors note that their DCGS-2 algorithm could provoke a loss of orthogonality instability because the vector q_2 is not re-orthogonalized with respect to q_1 when the pipeline begins. The DCGS-2 algorithm specifically lacks the $w^T w$ and $w^T z$ correction terms in our Algorithm 3, which employs the matrix T_{j-1}^{IC} with the updated $q_{j-1} - Q_{j-2} Q_{j-2}^T q_{j-1}$ and also introduces this delay. However, the Q_{j-1} in the projection step for R results in $\mathcal{O}(\varepsilon)$ representation error. In practice, our one synchronization CGS-2 algorithm achieves full $\|I - Q_{j-1}^T Q_{j-1}\| \approx \mathcal{O}(\varepsilon)$ orthogonality.

Algorithm 3. CGS (CGS-2) algorithm with normalization lag and re-orthogonalization lag

Input: Matrices Q_{j-1} and R_{j-1},$A_{j-1} = Q_{j-1}R_{j-1}$; column vector $q_j = a_j$
Output: Q_j and R_j, such that $A_j = Q_j R_j$

1: if $j = 1$ return
2: $[\, r_{j-1,j-1},\, r_{j-1,j}\,] = q_{j-1}^T[\, q_{j-1},\, q_j\,]$ ◁ Global synch
3: **if** $j > 2$ **then**
4: $\quad [\, w,\, z\,] = Q_{j-2}^T\, [\, q_{j-1},\, q_j\,]$, ◁ same global synch
5: $\quad [\, r_{j-1,j-1},\, r_{j-1,j}\,] = [\, r_{j-1,j-1} - w^T w,\, r_{j-1,j} - w^T\, z\,]$
6: $\quad r_{1:j-2,j-1} = r_{1:j-2,j-1} + w$ ◁ Lagged R update
7: **end if**
8: $r_{j-1,j-1} = \{\, r_{j-1,j-1}\,\}^{1/2}$ ◁ Lagged norm
9: if j>2 $q_{j-1} = q_{j-1} - Q_{j-2}\, w$ ◁ Lagged Reorthogonalization
10: $q_{j-1} = q_{j-1}/r_{j-1,j-1}$ ◁ Lagged Normalization
11: $r_{j-1,j} = r_{j-1,j}/r_{j-1,j-1}$
12: $r_{1:j-2,j} = z$ ◁ Apply recursive projection
13: $q_j = q_j - Q_{j-1}\, r_j$

2.3 | Inverse compact *WY*-MGS

The inverse compact *WY* representation of the MGS projector is equivalent to the lower triangular matrix inverse form of the projector

$$P^{IM}\, a_j = (\, I - Q_{j-1}\, \tilde{Q}_{j-1}^T) a_j = (\, I - Q_{j-1}\, (\, I + L_{j-1}\,)^{-1}\, Q_{j-1}^T\,)\, a_j, \tag{9}$$

$$P^{IM}\, a_j = (\, I - Q_{j-1}\, T_{j-1}^{IM}\, Q_{j-1}^T\,)\, a_j, \tag{10}$$

where

$$T_{j-1}^{IM} = (\, I + L_{j-1}\,)^{-1}. \tag{11}$$

The above expression implies

$$T_{j-1}^{IM} Q_{j-1}^T = \tilde{Q}_{j-1}^T, \quad (I + L_{j-1})^{-1} Q_{j-1}^T = \tilde{Q}_{j-1}^T, \tag{12}$$

$$Q_{j-1} = \tilde{Q}_{j-1}(\, I + L_{j-1}\,)^T = \tilde{Q}_{j-1}(\, I + L_{j-1}^T\,) = \tilde{Q}_{j-1}\,(\, I + U_{j-1}\,), \tag{13}$$

where the matrix $\tilde{Q}_{j-1}$ is defined in the statement of lemma 5.1 and corollary from the backward error analysis presented by Björck,[3] with

$$u_{ij} = q_i^T\, q_j, \quad j > i, \quad u_{ij} = 0, \quad j \le i, \tag{14}$$

with matrix product

$$Q_{j-1} = \tilde{Q}_{j-1}\,(\, I + U_{j-1}\,), \tag{15}$$

and the column vectors of

$$\tilde{Q}_{j-1} = \left[\tilde{q}_1, \quad \tilde{q}_2, \quad \dots\ , \quad \tilde{q}_{j-1}\right], \tag{16}$$

obtained from the composition of the orthogonal projectors $P_j = (\, I - q_j\, q_j^T\,)$

$$\tilde{q}_i = (\, I - q_1\, q_1^T\,) \cdots (\, I - q_{i-1}\, q_{i-1}^T\,)\, q_i, \quad i = 1,\ \dots\ , j-1. \tag{17}$$

The corollary of lemma 5.1 states that

$$P^{IM} = (\, I - Q_{j-1}\, \tilde{Q}_{j-1}^T) = (\, I - q_{j-1}\, q_{j-1}^T\,) \cdots (\, I - q_2\, q_2^T\,)\,(\, I - q_1\, q_1^T\,), \tag{18}$$

which is equivalent to the inverse compact WY representation of the projector.

When the matrix L_{j-1} is computed one column at a time and the triangular matrix T_{j-1}^{IM} is applied to the vector $Q_{j-1}^T a_j$, a lower triangular solver may replace matrix inversion in the MGS algorithm. A lower triangular solve version of MGS is given by Algorithm 4. The question of backward stability for the lower triangular solver version of MGS can be addressed by appealing to the forward stability analyses of Higham[31] for step 4 of the algorithm.

A synchronization step can be eliminated if the normalization in steps 8 through 9 is lagged to the $(j+1)$st iteration. A lagged-MGS algorithm that employs the inverse compact WY representation of the projection matrix P^{IM} is given by Algorithm 5.

2.4 | Compact *WY*-MGS

In order to derive the compact WY representation of the MGS algorithm, with lower triangular T_{j-1}^M, the matrix projection P^M can be written as

$$\begin{aligned} P^M &= (\, I - q_{j-1} q_{j-1}^T\,)\,(\, I - Q_{j-2}\, T_{j-2}^M\, Q_{j-2}^T\,) \\ &= I - q_{j-1} q_{j-1}^T - Q_{j-2}\, T_{j-2}^M\, Q_{j-2}^T + q_{j-1}\, q_{j-1}^T\, Q_{j-2}\, T_{j-2}^M\, Q_{j-2}^T, \end{aligned}$$

where T^M_{j-1} is the lower triangular matrix of inner products of the columns of Q_{j-1}. The *WY* representation is then given by the recursive matrix form derived by Björck,[14]

$$P^M = I - \begin{bmatrix} Q_{j-2} & q_{j-1} \end{bmatrix} \begin{bmatrix} T^M_{j-2} & 0 \\ -q^T_{j-1} Q_{j-2} T^M_{j-2} & 1 \end{bmatrix} \begin{bmatrix} Q^T_{j-2} \\ q^T_{j-1} \end{bmatrix},$$

and where

$$P^M = I - Q_{j-1} T^M_{j-1} Q^T_{j-1}.$$

A parallel-MGS algorithm that requires only one global synchronization step can be based on the above compact *WY* representation combined with a lagged normalization of the diagonal of the upper triangular matrix *R*. This is given as Algorithm 6.

In the lagged triangular solve variant of MGS (Algorithm 5), a single triangular matrix solve is applied in step 7, with computational complexity $\mathcal{O}(m^2)$. In contrast, the compact *WY* MGS Algorithm 6 requires two matrix-vector products each with the same $\mathcal{O}(m^2)$ cost. Therefore, the cost of the lower triangular solve orthogonalization kernel is slightly less.

Algorithm 4. Triangular solve MGS left-looking (columns)

Input: Matrices Q_{j-1} and R_{j-1}, $A_{j-1} = Q_{j-1}R_{j-1}$; column vector $q_j = a_j$; matrix L_{j-2} (if j>2)
Output: Q_j and R_j, such that $A_j = Q_jR_j$; matrix L_{j-1}

1: **for** $j = 1, 2, \ldots n$ **do**
2: **if** $j > 1$ **then**
3: $L^T_{:,j-1} = Q^T_{j-1}\, q_{j-1}$ ◁ Global synchronization
4: $r_j = (I + L_{j-1})^{-1}\, Q^T_{j-1}\, a_j$ ◁ Lower triangular solve
5: $a^{(j)}_j = a_j - Q_{j-1}\, r_j$
6: **end if**
7: $q_j = a^{(j)}$
8: $r_{jj} = \|q_j\|_2$
9: $q_j = q_j / r_{jj}$
10: **end for**

Algorithm 5. Inverse compact *WY* MGS algorithm with normalization lag

Input: Matrices Q_{j-1}, and R_{j-1}, $A_{j-1} = Q_{j-1}R_{j-1}$; column vector $q_j = a_j$; matrix L_{j-2} (if j>2)
Output: Q_j and R_j, such that $A_j = Q_jR_j$; matrix L_{j-1}

1: if $j = 1$ return
2: $[\, L^T_{:,j-1},\ r_j\,] = Q^T_{j-1}[q_{j-1}\, q_j]$ ◁ Global synchronization
3: $r_{j-1,j-1} = \|q_{j-1}\|_2$
4: $q_{j-1} = q_{j-1} / r_{j-1,j-1}$ ◁ Lagged normalization
5: $r_{j-1,j} = r_{j-1,j} / r_{j-1,j-1}$
6: $L^T_{:,j-1} = L^T_{:,j-1} / r_{j-1,j-1}$
7: $r_{1:j-1,j} = (\, I + L_{j-1}\,)^{-1}\, r_{1:j-1,j}$ ◁ Lower triangular solve
8: $q_j = q_j - Q_{j-1}\, r_{1:j-1,j}$

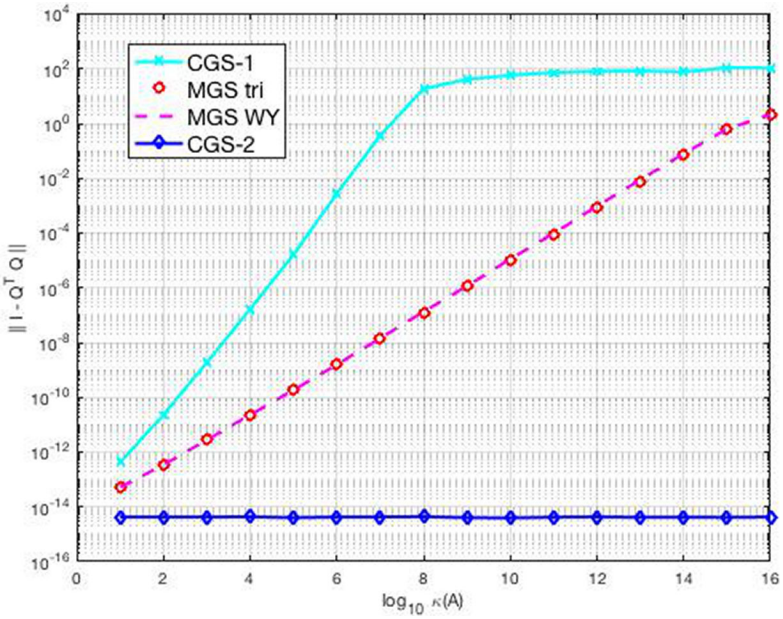


FIGURE 1 Loss of orthogonality for CGS Algorithm 1 and Algorithms 3–6

Algorithm 6. Compact *WY* MGS algorithm with normalization lag

Input: Matrices Q_{j-1}, R_{j-1} and T_{j-2}, $A_{j-1} = Q_{j-1}R_{j-1}$; column vector $q_j = a_j$; matrix T_{j-2} (if j>2)
Output: Q_j and R_j, such that $A_j = Q_jR_j$; matrix T_{j-1}

1: if $j = 1$ return
2: $[\, t_{:,j-1},\ r_j \,] = Q_{j-1}^T [q_{j-1}\ q_j]$ ◁ Global synchronization
3: $r_{j-1,j-1} = \|q_{j-1}\|_2$
4: $q_{j-1} = q_{j-1}/r_{j-1,j-1}$ ◁ Lagged normalization
5: $r_{j-1,j} = r_{j-1,j}/r_{j-1,j-1}$
6: $t_{:,j-1} = t_{:,j-1}/r_{j-1,j-1}$, $T^M_{j-1,j-1} = 1$
7: $t_{:,j-1} = -T^M_{j-2}\, t_{:,j-1}$ ◁ Recursion matvec
8: $r_{1:j-1,j} = T^T_{j-1}\, r_{1:j-1,j}$ ◁ Projection matvec
9: $q_j = q_j - Q_{j-1}\, r_{1:j-1,j}$

The loss of orthogonality for the Algorithms 4 to 6 has been computed using $\|I - Q^TQ\|_F$ for randomly generated matrices A of dimension 2000×200. The matrices A are computed as $A = U\,D\,V^T$ where U $(n \times m)$ and V $(m \times m)$ are obtained from QR factorizations of normally distributed random matrices. The diagonal matrix D of size $m \times m$ is computed with diagonal elements

$$D = \mathrm{diag}\begin{pmatrix} d_1 & d_2 & \cdots & d_n \end{pmatrix}, \quad d_i = 10^{\alpha(i-1)}, \quad \alpha = \frac{\log_{10}\kappa(A)}{m-1}.$$

The result is a sequence of matrices with condition number ranging from 1 to 10^{16}. The loss of orthogonality for CGS, CGS-2 and the MGS kernels above is plotted in Figure 1. Another measure of the loss of orthogonality was introduced by Paige[20] as $\|S\|_2$, where $S = (\, I + L^T\,)^{-1}L^T$. The norm remains close to $\mathcal{O}(\varepsilon)$ for orthogonal vectors and increases to one as the linear independence of the column vectors of Q_{j-1} is lost.

The standard CGS algorithm rapidly loses orthogonality at the rate $\mathcal{O}(\varepsilon)\kappa^2(A)$. The lower triangular inverse compact *WY* MGS Algorithm 5 exhibits the identical level of orthogonality as the compact *WY* MGS Algorithm 6 and these clearly follow the $\mathcal{O}(\varepsilon)\kappa(A)$ bound.

3 | LOW SYNCHRONIZATION MGS–GMRES ALGORITHM

Given a large sparse linear system of equations $Ax = b$, with residual vector $r = b - Ax$, the GMRES algorithm of Saad and Schultz[2] employs the Arnoldi Gram–Schmidt algorithm to generate orthogonal Krylov vectors $v_i := V_i, v_1 = r_0/\beta, \beta = \|r_0\|_2$, where $r_0 = b - Ax_0$, and x_0 is the initial guess. In order to orthogonalize the vectors spanning the Krylov subspace

$$\mathcal{K}_m = \{ v_1, Av_1, A^2v_1, \ldots A^{m-1}v_1 \}.$$

The algorithm produces an orthogonal matrix V_m and the matrix decomposition $V_m^T A V_m = H_m$

$$AV_m = V_m H_m + h_{m+1,m} v_{m+1} e_{m+1}^T = V_{m+1}\overline{H}_m.$$

Indeed, it was observed by Paige et al.[22] that the algorithm can be viewed as the Gram–Schmidt QR factorization of a matrix formed by adding a new column to V_m in each iteration

$$\begin{bmatrix} r_0, & AV_m \end{bmatrix} = V_{m+1} \begin{bmatrix} \|r_0\| e_1, & H_{m+1} \end{bmatrix}.$$

The loss of orthogonality bounds for GMRES will depend on the condition number $\kappa([r_0, AV_m])$ of the above Arnoldi matrix.

Algorithm 7. MGS–GMRES

Input: Matrix A; right-hand side vector b; initial guess vector x_0
Output: Solution vector x

1: $r_0 = b - Ax_0, v_1 = r_0/\|r_0\|_2$.
2: **for** $i = 1, 2, \ldots$ **do**
3: $z = Av_i, w = z$
4: **for** $j = 1, \ldots, i$ **do**
5: $h_{j,i} = (z, v_j)$
6: $w = w - h_{j,i} v_j$
7: **end for**
8: $h_{i+1,i} = \|w\|_2$
9: $v_{i+1} = w/h_{i+1,i}$
10: Apply Givens rotations to H_i
11: **end for**
12: $y = \text{argmin}\|(H_m y - \|r_0\|_2 e_1)\|_2$
13: $x = x_0 + V_m y$

The GMRES algorithm with MGS orthogonalization is presented as Algorithm 7 below. The MGS GMRES Algorithm 7 computes the orthogonalization in steps 4 to 7. The normalization steps 8 and 9 are completed afterwards, looking forward one iteration to compute $h_{i+1,i}$ and v_{i+1}. Ghysels et al.[27] proposed a Pythagorean form of the normalization $\|w\|_2$ to avoid a global reduction

$$h_{i+1,i} = \{ \|z\|_2^2 - \|H_{1:i,i}\|_2^2 \}^{1/2}. \tag{19}$$

This computation is unstable and suffers from numerical cancellation because it does not account for the loss of orthogonality in V and approximates the inner product

$$h_{i+1,i}^2 = (z - V_i H_i)^T (z - V_i H_i).$$

Both Ghysels et al.[27] and Yamazaki et al.[28] describe pipelined algorithms relying on CGS that incorporate an iteration lag in the normalization step to avoid the unstable normalization above. However, these algorithms are somewhat unstable because the loss of orthogonality for CGS is $\mathcal{O}(\varepsilon)\kappa^2([r_0,\ AV_m])$, see Reference 5. The authors employ a change of basis QR factorization, $V_m = Z_{:,1:k}G_{1:k,1:m}$ and shifts in order to mitigate these instabilities. However, the pipeline depths still must remain relatively small.

Algorithm 8. One synchronization MGS–GMRES with lagged normalization

Input: Matrix A; right-hand side vector b; initial guess vector x_0
Output: Solution vector x

1: $r_0 = b - Ax_0$, $v_1 = r_0$.
2: $v_2 = Av_1$
3: $(\ V_2,\ R,\ L_2\ ) = \text{mgs}(\ V_2,\ R,\ L_1\ )$
4: **for** $i = 1, 2, \ldots$ **do**
5: $\quad v_{i+2} = Av_{i+1}$ ◁ Matrix-vector product
6: $\quad [\ L^T_{:,i+1},\ r_{i+2}\ ] = V^T_{i+1}[v_{i+1}\ v_{i+2}]$ ◁ Global synchronization
7: $\quad r_{i+1,i+1} = \|v_{i+1}\|_2$
8: $\quad v_{i+1} = v_{i+1}/r_{i+1,i+1}$ ◁ Lagged normalization
9: $\quad r_{1:i+1,i+2} = r_{1:i+1,i+2}/r_{i+1,i+1}$ ◁ Scale for Arnoldi
10: $\quad v_{i+2} = v_{i+2}/r_{i+1,i+1}$ ◁ Scale for Arnoldi
11: $\quad r_{i+1,i+2} = r_{i+1,i+2}/r_{i+1,i+1}$
12: $\quad L^T_{:,i+1} = L^T_{:,i+1}/r_{i+1,i+1}$
13: $\quad r_{1:i+1,i+2} = (\ I + L_{i+1}\ )^{-1}\ r_{1:i+1,i+2}$ ◁ Lower triangular solve
14: $\quad v_{i+2} = v_{i+2} - V_{i+1}\ r_{1:i+1,i+2}$
15: $\quad H_i = R_{i+1}$
16: $\quad$ Apply Givens rotations to H_i
17: **end for**
18: $y_m = \text{argmin}\|(\ H_m y_m - \|r_0\|_2 e_1\ )\|_2$
19: $x = x_0 + V_m y_m$

The MGS-GMRES algorithm is backward stable[22] and orthogonality is maintained to $\mathcal{O}(\varepsilon)\kappa([r_0,\ AV_m])$, where the condition number of the Arnoldi matrix $\kappa([r_0,\ AV_m])$ is defined as the ratio of the largest to smallest singular values.[4] By employing the inverse compact WY MGS algorithm with a normalization lag, the normalization for iteration $i+1$ is delayed. The diagonal of the R matrix in the QR factorization, corresponding to H_i, is updated after the MGS orthogonalization step. For the Arnoldi-QR, the normalization of the vector q_{j-1} in the matrix-vector product $q_j = Aq_{j-1}$ is delayed. Therefore, in Algorithm 5, two additional steps after Step 3 are required, $r_{1:j-1,j} = r_{1:j-1,j}/r_{j-1,j-1}$ and $q_j = q_j/r_{j-1,j-1}$. The lagged inverse compact WY algorithm is implemented below as Algorithm 8 for the ith iteration of MGS-GMRES, The lagged normalization for iteration i is computed in Step 6. Therefore, the column $H_i = R_{i+1}$ of the Hessenberg matrix is computed during iteration i and the matrix $H_{:,i}$ is employed to solve the least squares problem. The loss of orthogonality for MGS–GMRES Algorithm 8 is $\mathcal{O}(\varepsilon)\kappa([r_0,\ AV_m])$.

4 | IMPLEMENTATION DETAILS

The traditional implementation of the MGS process requires i separate inner products (global summations) in iteration i. Therefore, the amount of global communication grows quadratically with the iterations. The iterated classical Gram–Schmidt (ICGS) algorithm with two re-orthogonalization steps requires three global reductions versus one synch CGS-2. The Trilinos–Belos solver stack employs an ICGS-GMRES.[32] Hernandez et al.[24] implemented lagged normalization and re-orthogonalization in DCGS-2. Our one-synch CGS-2 and MGS require less computation and communication than previous algorithms. The one-synch MGS-GMRES algorithm allows us to employ the inverse compact WY MGS algorithm, by combining a mass inner product and lagged normalization into one global reduction MPI_AllReduce per GMRES iteration. The mass inner product for computing $v = X^T y$ is written as

$$v = y \cdot X = y \cdot x_1 + y \cdot x_2 + \cdots + y \cdot x_m.$$

The algorithm for $w = z - V\,H_{1:i,i}$ is implemented with the vector mass AXPY (MAXPY).

$$y = y + X \begin{bmatrix} \alpha_1 & & & \\ & \alpha_2 & & \\ & & & \\ & & & \alpha_m \end{bmatrix} = y + \alpha_1 x_1 + \cdots + \alpha_m x_m.$$

For optimal performance of these two operations on current generation multicore processors, cache-blocking combined with loop unrolling can be exploited. Once the Krylov subspace dimension is sufficiently large, the outer loop of the mass inner product should be unrolled into batches of two, four, or more summations. The overall effect is to expose more work to multiple floating point units and prefetch y together with multiple columns of X into the cache. The outer loop can also be threaded given the typically large vector lengths. Unrolling should have the beneficial effect of increasing the memory bandwidth utilization of the processors. The vector MAXPY should also be unrolled with the elements α_i pre-fetched into the cache. The Hessenberg matrix H is small and dense. Thus, it is laid out in memory by columns for cache access and fast application of the orthogonal rotations in the least-squares problem. The remaining operations inside the one-sync MGS–GMRES are norms, (sparse) matrix-vector products and the preconditioner. The sparse matrix-vector multiply is parallel and can be easily partitioned by rows when using a compressed-sparse row (CSR) storage format.

Both the mass inner product and vector MAXPY are well-suited to a single-instruction multiple-data (SIMD) model of parallel execution. Both operations can be implemented using a single CUDA kernel, although reduction across processes is required for global result. Unrolling of the inner products into batches is applied and permits higher sustained memory bandwidth between the GPU global memory and streaming multiprocessors (SM). This is also the case for the vector MAXPY. A recent paper by Romero et al.[33] also describes a vector MAXPY for the GPU. The PETSc library employs the vector MAXPY for a pipelined GMRES iteration.[27] An important optimization to increase the ratio of flops to data transferred is to fuse two mass inner products in line 6 of Algorithm 8 and perform them in one kernel call.

5 | NUMERICAL RESULTS

The primary motivation for proposing the low synchronization GMRES algorithms was to improve the performance and strong-scaling characteristics of DOE physics simulations at exascale and, in particular, the Nalu CFD solver, Domino et al.[34] The Nalu model solves the incompressible Navier-Stokes equations on unstructured finite-element meshes. The momentum and pressure equations are time-split and solved sequentially. Both the momentum and pressure continuity equation are solved with preconditioned GMRES iterations. The pressure preconditioner employs either smoothed-aggregation or classical Ruge-Stüben algebraic multigrid (C-AMG). The latter is provided by the *hypre*-BoomerAMG[35] library of solvers.[36] These solvers, are an expensive part of the simulation, requiring over 30% of the simulation time.

In order to evaluate the performance of different GMRES solvers, a separate *hypre* linear solver application has been implemented. In our timing studies, matrices were obtained from Nalu wind turbine simulations and tested in our *hypre* solver application. The performance studies involving GPU acceleration were completed on a single node of the Summit-Dev supercomputer at ORNL and on the NREL Eagle system. Summit-Dev is built with IBM S822LC nodes which consist of two 10-core IBM Power-8 processors with 256 GB of DD4 memory and four NVIDIA Tesla P100 GPUs connected by NVlink running at 80 GB/s. The GPUs share 16 GB HBM2 memory. The interconnect consists of two rails of Mellanox EDR Infiniband.

Gram–Schmidt orthogonalization, as emphasized earlier, is the most expensive part of the GMRES algorithm. Hence, the algorithms developed in this paper are applied. These employ the low synchronization CGS and MGS algorithms. For each example, we specify which orthogonalization method was employed.

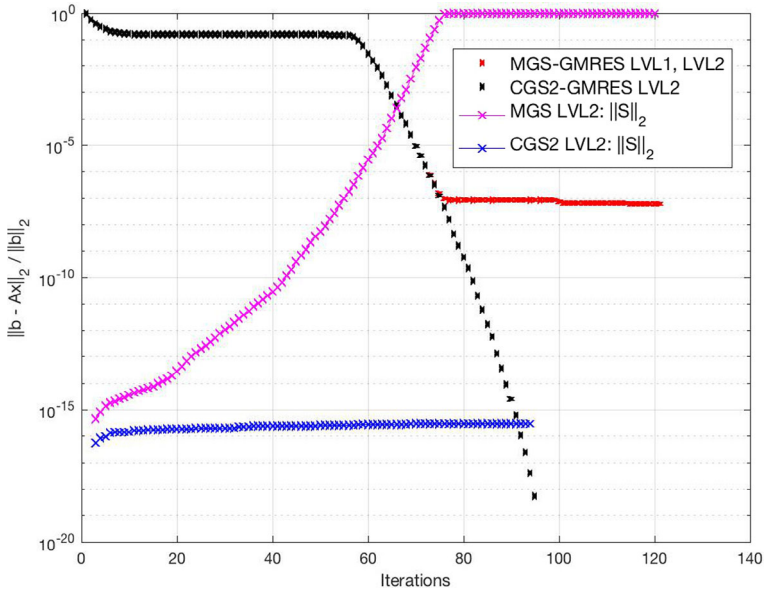


FIGURE 2 Simoncini matrix. Limiting accuracy for modified Gram–Schmidt (MGS)–Generalized Minimal Residual Method (GMRES) (LVL1 refers to the original level-1 blas MGS). LVL2 refers to level-2 blas (new one-reduce algorithm with DGEMM mass-IP). CGS with re-orthogonalization (CGS-2) GMRES (LVL2 refers to the level-2 blas one-reduce CGS-2). Loss of orthogonality metric $\|S\|_2$

5.1 | Numerical stability and accuracy

Linear systems appearing in the literature are employed to evaluate the numerical stability and limiting accuracy for several of the GMRES algorithms described here. All tests were performed in Matlab and the loss of orthogonality metric $\|S\|_2$ that is plotted for the Krylov vectors was derived by Paige.[20] The first problem was proposed by Simoncini and Szyld.[37] A diagonal matrix of dimension 100 is specified as $A = \text{diag}(1 \times 10^{-8}, 2, \ldots, 100)$ with random right-hand side $b = \text{randn}(100, 1)$, that is normalized so that $\|b\|_2 = 1$. The matrix condition number is $\kappa(A) = 1 \times 10^{10}$ and is controlled by the small first diagonal element. A preconditioner is not employed in this case. The problem allows us to specify the limiting accuracy of the GMRES iteration. In particular, Figure 2 plots the implicit residual versus the number of iterations for the standard MGS–GMRES and one-synch MGS-GMRES algorithm. These exhibit identical convergence behavior with the loss of orthogonality metric $\|S\|_2$ increasing to one at 80 iterations. The convergence stalls at this point with the relative residual reaching 1×10^{-7}. In the case of the one synchronization CGS (CGS-2) in Algorithm 3, the loss of orthogonality metric $\|S\|_2$ remains close to machine precision and the convergence curve continues to decrease without stalling to 1×10^{-18}.

A second problem is taken from the Florida (now Texas A&M) sparse matrix collection. The particular system is "thermal1" with dimension 82,654 and number of nonzeros, $nnz(A) = 574,458$. The right-hand side is obtained from $b = Ax$, where $x = [\ 1,\ 1,\ \ldots,\ 1\ ]^T$. For this problem a Ruge-Stüben[38] classical AMG preconditioner is applied. Once again the standard MGS–GMRES and one-synch GMRES algorithms are compared with the CGS-2 GMRES algorithm(s). The implicit residuals and loss of orthogonality metrics are plotted in Figure 3. Observe that when $\|S\|_2 = 1$ that the convergence stalls for the MGS-GMRES algorithms, whereas CGS-2 maintain orthogonality to near the level of machine precision.

5.2 | GPU C/CUDA performance

The results presented in Figure 4 were obtained on Summit-Dev computational cluster (OLCF) using NVCC V9.0.67. All variables are allocated using device memory. The NVIDIA cuSPARSE and cuBLAS libraries implement the vector operations (SpMV, AXPY, SCAL, DOT). It could be assumed that MDOT and MAXPY can be expressed as dense matrix–matrix product (MDOT) and a combination of dense matrix–matrix products (MAXPY), and implemented using cuBLAS routines. However, it was found that this approach was less efficient than our optimized implementation. We speculate that the main reason for suboptimal performance was the need to transpose V in MDOT, resulting in two CUDA kernel calls instead of one for a MAXPY. Hence, the MAXPY and MDOT are implemented as fused operations in CUDA and optimized for maximum performance.

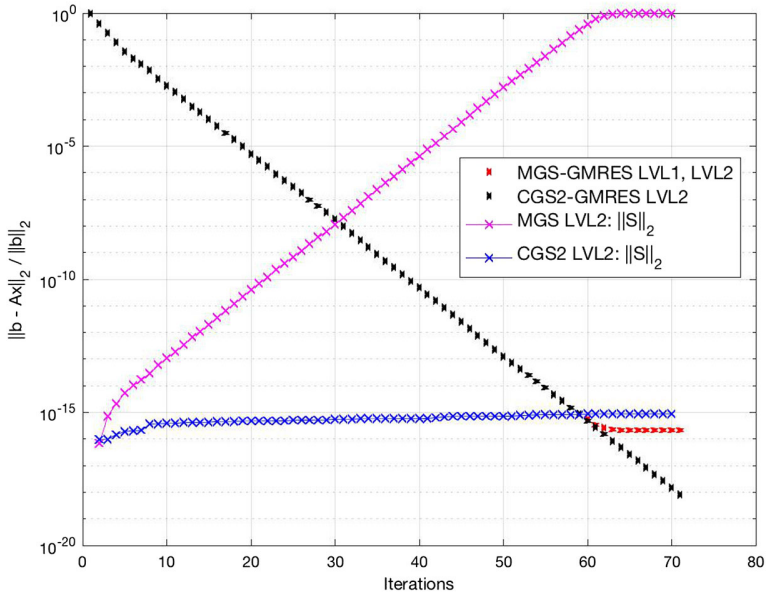


FIGURE 3 Thermal1 matrix. Limiting accuracy for modified Gram–Schmidt (MGS)–Generalized Minimal Residual Method (GMRES) (LVL1 refers to the original level-1 blas MGS). LVL2 refers to level-2 blas (new one-reduce algorithm with DGEMM mass-IP). CGS with re-orthogonalization (CGS-2) GMRES (LVL2 refers to the level-2 blas one-reduce CGS-2). Loss of orthogonality metric $\|S\|_2$

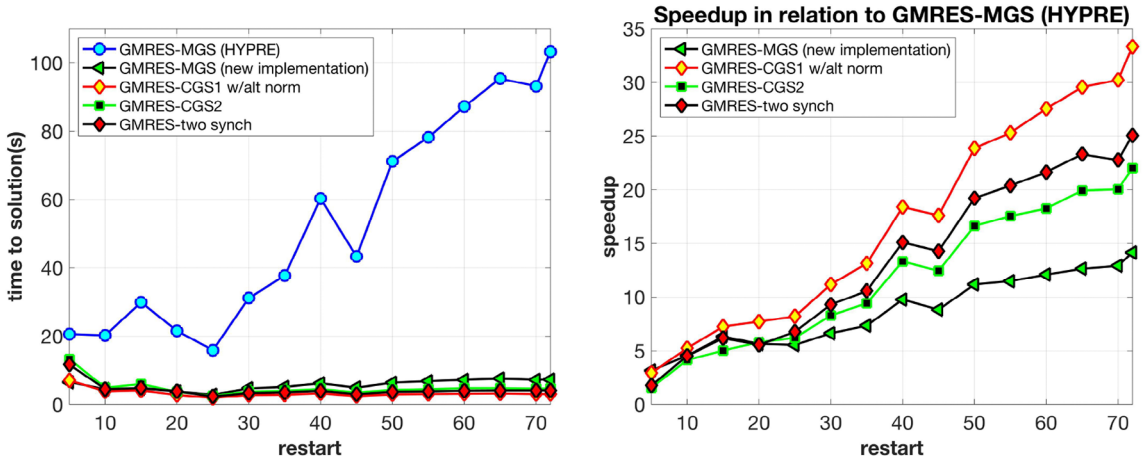


FIGURE 4 Left: Performance of the *hypre* GPU implementation of modified Gram–Schmidt (MGS)–Generalized Minimal Residual Method (GMRES) (m) versus GMRES (m) with alternative orthogonalization strategies. Right: The speedup resulting from replacing *hypre* MGS with alternate low-synch GMRES variants. Tolerance 1×10^{-6}. Linear system from Nalu wind simulation

The linear system employed for GPU performance analysis was extracted from the Nalu Vestas V27 wind turbine simulation described in Thomas et al.[1] The matrix has 675,905 rows and 675,905 columns. The average number of nonzeros per row is 11. The matrix is stored in the standard compressed-sparse row (CSR) format, which is supported by the cuSPARSE library. A preconditioner is not applied in this test.

Five approaches are compared for solving the linear system: (1) MGS-GMRES (m) compiled using CUDA, (2) MGS-GMRES (m) written by the authors for *hypre* using only device memory and reducing unnecessary data copies and convergence checks, (3) GMRES (m) with CGS and Pythagorean normalization (see Equation (19) and (3.3) in Reference 27), (4) CGS-2 GMRES (Algorithm 1), and (5) one-synch GMRES (m) as in Algorithm 3. Restarts are set as $m = 5, 10, \ldots, 65, 70, 72$ with relative tolerance 1×10^{-6}. The only difference between approaches (2)–(5) is how they orthogonalize Krylov vectors. Figure 4 (left) displays the improvement in run-time by using alternative orthogonalization approaches. The improvement is much greater, as expected, for large restart values. Figure 4 (right) indicates the associated speedups.

The left side of Figure 5 is a plot of the ratio of wall-clock run-time taken by Gram–Schmidt to the total run-time of the sparse linear system solver. One can easily observe that the MGS algorithm accounts for 90% of the total run-time. Using

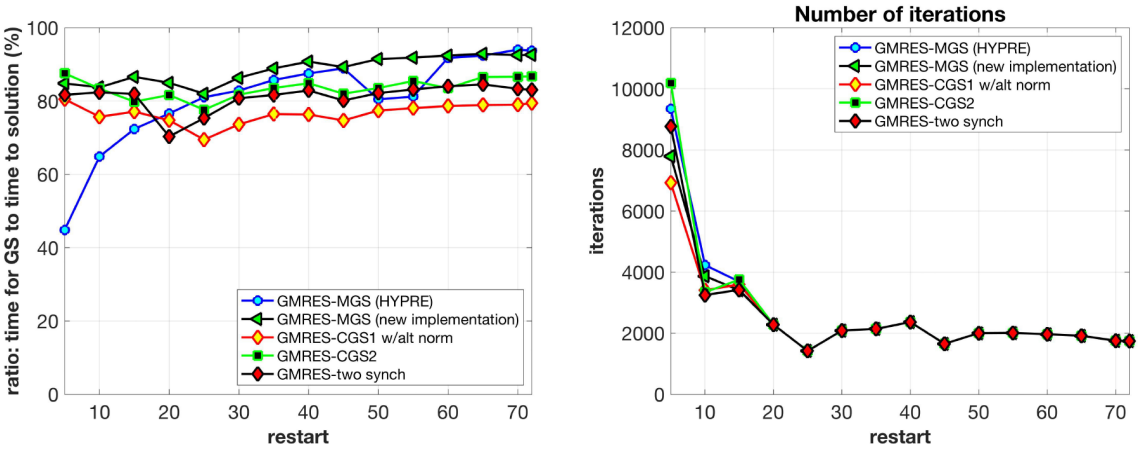


FIGURE 5 Performance of the *hypre* GPU implementation of modified Gram–Schmidt (MGS)–Generalized Minimal Residual Method (GMRES) (m) with MGS and GMRES (m) with alternative orthogonalization strategies. Left: The ratio of the time spent on Gram–Schmidt orthogonalization versus time to solution. Right: Number of iterations for *hypre* MGS-GMRES (m) versus alternate low synch GMRES (m)

TABLE 2 Parabolic FEM matrix, restart: 72

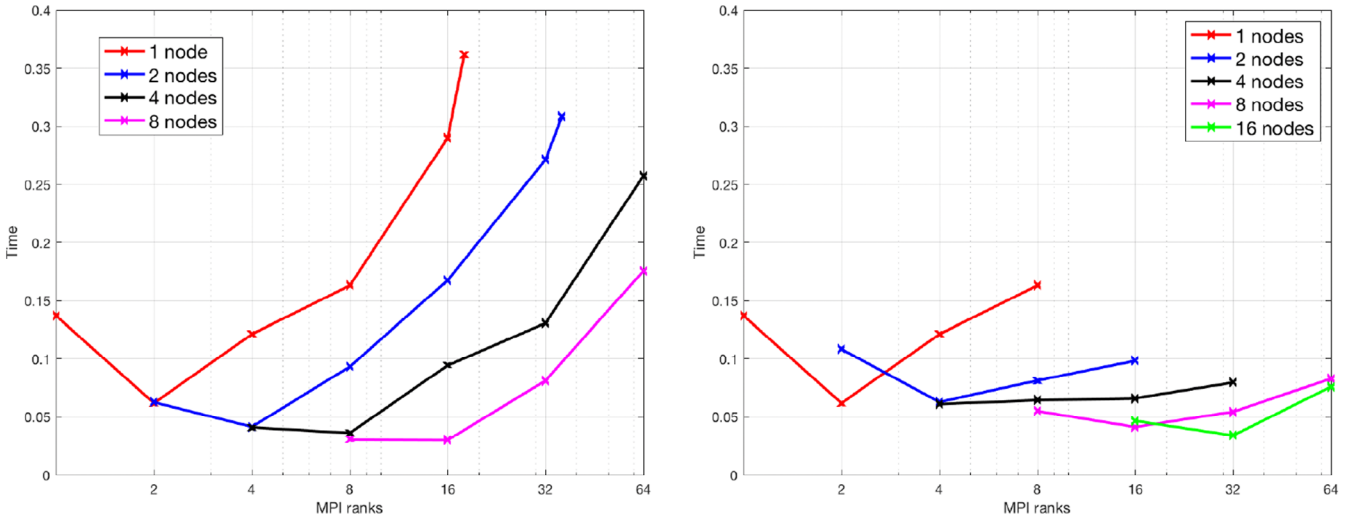
Method	Tolerance	Iters	Orth (s)	Precon	Solve (s)
GMRES-hypre	1e-12	79	0.2604	5.1804	6.4312
GMRES-MGS (new)	1e-12	81	0.1470	4.7869	5.2589
GMRES-CGS Pyth norm	1e-12	81	0.0670	5.1703	5.5660
GMRES-CGS-2	1e-12	81	0.0974	5.1596	5.5811
GMRES-CGS-2 one-synch	1e-12	81	0.0849	5.1803	5.5926

Abbreviations: CGS, classical Gram–Schmidt; CGS-2, CGS with re-orthogonalization; GMRES, Generalized Minimal Residual Method; MGS, modified Gram–Schmidt.

GMRES with different orthogonalization strategies lowers this time down to 40% from 70%. The remainder of the execution time is taken by operations on vectors and matvecs. Hence, an even larger improvement is expected in run-time when using a preconditioner. The right side of Figure 5 displays the speedup resulting from using low-synch orthogonalization methods. The implementation of the *hypre* MGS-GMRES GPU implementation is taken as the baseline reference timing. Figure 5(right) displays the number of GMRES iterations resulting from using low-synch orthogonalization methods. It may be observed that for short restart cycles the number of iterations varies for the different methods. This is expected because of the CGS stability versus CGS-2 and MGS.

The second test was performed on Eagle supercomputer (DOE NREL), equipped with NVIDIA V100 GPUs and Dual Intel Xeon Gold Skylake 6154 (3.0 GHz, 18-cores) CPUs. The "Parabolic finite element method (FEM)" from the Florida sparse matrix collection is used in a second performance test. Total run-times are reported in Table 2. The matrix dimension is $n = 525,825$ with $nnz(A) = 3,674,625$ nonzeros. The *hypre*-BoomerAMG V-cycle with six levels is the preconditioner with l_1 scaled Jacobi smoother on the GPU. The complete description of *hypre*-BoomerAMG settings can be found in the paper https://github.com/kswirydo/LowSynchGMRESAlgorithmsrepository, in a document titled *hypreSettings.pdf*. The total solver (Solve) time is reported along with the Gram–Schmidt (Orth) times and number of GMRES iterations (Iters) required to reach the specified relative tolerance. The lowest run-times are achieved by the GMRES algorithm that uses CGS with Pythagorean norm. However, this method is not applicable for matrices with high condition numbers. The Gram–Schmidt times are 4× faster than the *hypre* MGS-GMRES (for 1×10^{-12}). The table shows that the cost is highly dominated by the preconditioner, which accounts for over 90% of the total runtime. This data is provided to point out that even driving the cost of the Gram–Schmidt procedure to zero is not able to improve the solver runtime if the other parts of the solver are slow.

We emphasize that while the improvement over the existing *hypre* implementation is high for orthogonalization, only one GPU and one MPI rank were employed in these tests. Further parallel strong-scaling studies were conducted using



(a) Hypre Gram-Schmidt Times

(b) Low-Synch Gram-Schmidt Time

FIGURE 6 Gram–Schmidt kernels time on Eagle Skylake + V100 GPU. NREL ABL Mesh $n = 9$M. (a) Hypre Gram–Schmidt times; (b) Low-Synch Gram–Schmidt time

a multi-MPI rank multi-GPU implementation for comparison. Initial testing of the multi-MPI, multi-GPU *hypre* linear solver on the Eagle supercomputer at NREL has demonstrated that the low-synch algorithms result in a reduced communication overhead for the MPI global reductions. Eagle nodes contain Intel Skylake processors, 2×18 core sockets, 64 GB DDR4 memory, inter-connected with an Infiniband network. The run-time spent in the Gram–Schmidt orthogonalization is plotted in Figure 6 for the Nalu ABL $n = 9$M linear system. The *hypre* MGS-GMRES orthogonalization times are compared with the low-synch MGS-GMRES. Clearly, the low-Synch algorithms reduce the Gram–Schmidt time as the node count increases.

6 | CONCLUSIONS

We have presented low-synch CGS and MGS algorithms for $A = QR$ and applied these to the Arnoldi-QR algorithm. A single global synchronization step is now sufficient to achieve $\mathcal{O}(\varepsilon)$ orthogonality in CGS with delayed re-orthogonalization or $\mathcal{O}(\varepsilon)\kappa(A)$ in the MGS algorithm. A new variant of the MGS algorithm relying on a lower triangular solve recurrence was derived and the equivalence with the compact *WY* MGS formulation was established. A one-synchronization MGS-GMRES algorithm employing the compact *WY* MGS algorithm with a lagged normalization step results in a backward-stable algorithm[22] with $\mathcal{O}(\varepsilon)\kappa([r_0, AV_m])$ loss of orthogonality. GPU implementations of these low-synch algorithms achieve up to a 35 times speed-up over the *hypre* MGS-GMRES algorithm.

The execution speed of our GPU low-synch GMRES algorithms implies a new scaling paradigm for large sparse linear solvers. The relative balance in run-time that is split between GMRES (including matrix-vector multiplies) and a preconditioner such as the *hypre*-BoomerAMG V-cycle may now shift in favor of a larger number of inexpensive GMRES iterations combined with a light-weight V-cycle in order to achieve a much lower and optimal run-time. For example, we solved a large $n = 500$K linear system in under six seconds with 100 GMRES iterations, using GPU accelerated matrix-vector multiplies, preconditioned by a less costly V-cycle and l_1 Jacobi smoother on the GPU. The challenge is to extend this scaling result to a large number of nodes with a low number of synchronizations.

An extension of this work is our distributed-memory parallel implementation using MPI and CUDA. We have presented orthogonalization times for our multi-MPI and multi-GPU implementation. The theory developed in this paper and the preliminary results indicate that our approach, while stable, also has the desired scalability properties both in terms of fine and coarse grain parallelism. Further, investigation is needed to establish the scaling properties of the proposed algorithms on $\mathcal{O}(10)K$ nodes.

ACKNOWLEDGEMENTS

This work was authored in part by the National Renewable Energy Laboratory, operated by Alliance for Sustainable Energy, LLC, for the U.S. Department of Energy (DOE) under Contract No. DE-AC36-08GO28308. Funding provided by the Exascale Computing Project (17-SC-20-SC), a collaborative effort of two U.S. Department of Energy organizations (Office of Science and the National Nuclear Security Administration) responsible for the planning and preparation of a capable exascale ecosystem, including software, applications, hardware, advanced system engineering, and early testbed platforms, in support of the nation's exascale computing imperative. The views expressed in the article do not necessarily represent the views of the DOE or the U.S. Government. The U.S. Government retains and the publisher, by accepting the article for publication, acknowledges that the U.S. Government retains a nonexclusive, paid-up, irrevocable, worldwide license to publish or reproduce the published form of this work, or allow others to do so, for U.S. Government purposes. This work was also performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344. *hypre* Version 2.14.0 was employed in our studies. We are most grateful to two anonymous reviewers whose comments greatly improved our manuscript, and lead to a more unified presentation. Our work was inspired by the fundamental contributions made by A. Björck, A. Rühe, and C. C. Paige. This work was funded by the DOE Exascale Computing Project (17-SC-20-SC). Julien Langou was supported by NSF award #1645514.

ORCID

Katarzyna Świrydowicz https://orcid.org/0000-0001-5758-5394

REFERENCES

1. Thomas S, Ananthan S, Yellapantula S, Hu JJ, Lawson M, Sprague MA. A comparison of classical and aggregation-based algebraic multigrid preconditioners for high-fidelity simulation of wind-turbine incompressible flows. SIAM J Sci Stat Comput. 2019;41(5):S196–S219.
2. Saad Y, Schultz MH. GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems. SIAM J Sci Stat Comput. 1986;7(3):856–869.
3. Björck Å. Solving linear least squares problems by Gram-Schmidt orthogonalization. BIT Numer Math. 1967;7(1):1–21.
4. Giraud L, Langou J, Rozloznik M. The loss of orthogonality in the Gram-Schmidt orthogonalization process. Comput Math Appl. 2005;50(7):1069–1075.
5. Giraud L, Langou J, Rozložník M, van den Eshof J. Rounding error analysis of the classical Gram-Schmidt orthogonalization process. Numer Math. 2005;101(1):87–100.
6. Leon SJ, Björck Å, Gander W. Gram-Schmidt orthogonalization: 100 years and more. Numer Linear Algebra Appl. 2013;20(3):492–532.
7. Smoktunowicz A, Barlow JL, Langou J. A note on the error analysis of classical Gram–Schmidt. Numer Math. 2006;105(2):299–313.
8. Ruhe A. Numerical aspects of Gram-Schmidt orthogonalization of vectors. Linear Algebra Appl. 1983;52:591–601.
9. Walker HF. Implementation of the GMRES method using Householder transformations. SIAM J Sci Stat Comput. 1988;9(1):152–163.
10. Puglisi C. Modification of the Householder method based on the compact WY representation. SIAM J Sci Stat Comput. 1992;13(3):723–726.
11. Joffrain T, Low TM, Quintana-Ortí ES, van de Geijn R, Van Zee FG. Accumulating householder transformations, revisited. ACM Trans Math Softw. 2006;32(2):169–179.
12. Schreiber R, Van Loan C. A storage-efficient WY representation for products of householder transformations. SIAM J Sci Stat Comput. 1989;10(1):53–57.
13. Bischof C, Van Loan C. The WY representation for products of Householder matrices. SIAM J Sci Stat Comput. 1987;8(1):s2–s13.
14. Björck Å. Numerics of Gram-Schmidt orthogonalization. Linear Algebra Appl. 1994;197:297–316.
15. Malard J, Paige C. Efficiency and scalability of two parallel QR factorization algorithms. Proceedings of the IEEE Scalable High Performance Computing Conference. Knoxville, TN: IEEE; 1994;615–622.
16. Sun X. Aggregations of elementary transformations. Technical Report DUKE-TR-1996-03. Duke University, Durham, NC: Duke University, 1996.
17. Björck Å, Paige CC. Loss and recapture of orthogonality in the Modfied Gram-Schmidt algorithm. SIAM J Matrix Anal A. 1992;13:176–190.
18. Paige CC. A useful form of unitary matrix obtained from any sequence of unit 2-norm n-vectors. SIAM J Matrix Anal A. 2009;31(2):565–583.
19. Paige CC, Wülling W. Properties of a unitary matrix obtained from a sequence of normalized vectors. SIAM J Matrix Anal A. 2014;35(2):526–545.
20. Paige CC. The effects of loss of orthogonality on large scale numerical computations. In: ICCSA. Springer; New York, NY: 2018. p. 429–439.
21. Giraud L, Gratton S, Langou J. A rank-*k* update procedure for reorthogonalizing the orthogonal factor from modified Gram–Schmidt. SIAM J Matrix Anal A. 2004;25(4):1163–1177.
22. Paige CC, Rozlozník M, Strakoš Z. Modified Gram-Schmidt (MGS), least squares, and backward stability of MGS-GMRES. SIAM J Matrix Anal A. 2006;28(1):264–284.
23. Paige CC, Strakoš Z. Residual and backward error bounds in minimum residual Krylov subspace methods. SIAM J Sci Stat Comput. 2002;23(6):1899–1924.
24. Hernández V, Román JE, Tomás A. Parallel Arnoldi eigensolvers with enhanced scalability via global communications rearrangement. Parallel Comput. 2007;33(7-8):521–540.

25. Hernández V, Román JE, Tomás AA. Parallel variant of the Gram-Schmidt process with reorthogonalization. ParCo 2005. Jülich, Germany: John von Neumann Institute for Computing, 2006; p. 220–228.
26. Kim SK, Chronopoulos AT. An efficient parallel algorithm for extreme eigenvalues of sparse nonsymmetric matrices. Int J Comput Appl. 1992;6(1):98–111.
27. Ghysels P, Ashby TJ, Meerbergen K, Vanroose W. Hiding global communication latency in the GMRES algorithm on massively parallel machines. SIAM J Sci Comput. 2013;35(1):C48–C71.
28. Yamazaki I, Hoemmen M, Luszczek P, Dongarra J. Improving performance of GMRES by reducing communication and pipelining global collectives. Proceedings of the Parallel and Distributed Processing Symposium Workshops (IPDPSW), 2017 IEEE International. Lake Buena Vista, FL: IEEE; 2017;1118–1127.
29. Yamazaki I, Anzt H, Tomov S, Hoemmen M, Dongarra J. Improving the performance of CA-GMRES on multicores with multiple GPUs. Proceedings of the Parallel and Distributed Processing Symposium, 2014 IEEE 28th International. Arizona Grand Resort, PHOENIX (Arizona) USA: IEEE; 2014;382–391.
30. Stewart G. Matrix algorithms: Basic decompositions. Vol 4. Philadelphia: SIAM, 1998.
31. Higham NJ. The accuracy of solutions to triangular systems. SIAM J Numer Anal. 1989;26(5):1252–1265.
32. Bavier E, Hoemmen M, Rajamanickam S, Thornquist H. Amesos2 and Belos: Direct and iterative solvers for large sparse linear systems. Sci Progr NETH. 2012;20(3):241–255.
33. Romero E, Tomás A, Soriano A, Blanquer I. A fast sparse block circulant matrix vector product. Proceedings of the European Conference on Parallel Processing. New York, NY: Springer; 2014. p. 548–559.
34. Domino SP. Design-order, non-conformal low-mach fluid algorithms using a hybrid CVFEM/DG approach. J Comput Phys. 2018;359:331–351.
35. Yang U, Falgout R. Hypre high performance preconditioners. Livermore, CA: Lawrence Livermore National Lab.(LLNL), 2018.
36. Henson VE, Yang UM. BoomerAMG: A parallel algebraic multigrid solver and preconditioner. Appl Numer Math. 2002;41(1):155–177. Developments and Trends in Iterative Methods for Large Systems of Equations - in memorium Rudiger Weiss.
37. Simoncini V, Szyld DB. Theory of inexact Krylov subspace methods and applications to scientific computing. SIAM J Sci Comput. 2003;25(2):454–477.
38. Ruge JW, Stüben K. Algebraic multigrid. Multigrid methods. Philadelphia: SIAM, 1987; p. 73–130.

How to cite this article: Świrydowicz K, Langou J, Ananthan S, Yang U, Thomas S. Low synchronization Gram–Schmidt and generalized minimal residual algorithms. *Numer Linear Algebra Appl*. 2021;28:e2343. https://doi.org/10.1002/nla.2343

APPENDIX. A

Ruhe[8] provides a unique perspective by proving that the iterated CGS and MGS algorithms are equivalent to Gauss–Jacobi and Gauss–Seidel iterations for linear systems. The key observation made by Ruhe[8] is that for the CGS algorithm, the loop-invariants are obtained by unrolling the recurrences as follows,

$$\begin{aligned} a^{(k)} &= a^{(k-1)} - Qs^{(k-1)} \\ &= a^{(k-2)} - Qs^{(k-2)} - Qs^{(k-1)} \\ &= a^{(0)} - Q(\, s^{(0)} + s^{(1)} + \cdots + s^{(k-1)}\,). \end{aligned}$$

Let $r^{(0)} = R_j$ and $a := a^{(0)} = A_{:,j}$, then it follows that

$$\begin{aligned} r^{(k)} &= r^{(k-1)} + s^{(k-1)} \\ &= r^{(k-2)} + s^{(k-2)} + s^{(k-1)} \\ &= r^{(0)} + s^{(0)} + s^{(1)} + \cdots + s^{(k-1)}. \end{aligned}$$

and therefore $a^{(k)} = a - Q\, r^{(k)} = Pa^{(k-1)}$, where P is a projection matrix.

$$\begin{aligned} r^{(k)} &= r^{(k-1)} + Q^T a_j^{(k-1)} \\ &= r^{(k-1)} + Q^T(\, a - Qr^{(k-1)}\,) \end{aligned}$$

$$= r^{(k-1)} + Q^T a - Q^T Q r^{(k-1)}.$$

This is the Gauss–Jacobi iteration applied to the normal equations for the matrix Q and right-hand side vector a

$$Q^T Q\, r = Q^T\, a.$$

The projector applied to columns of the matrix A is then seen to be an approximation of the matrix

$$P = I - Q\,(\,Q^T Q\,)^{-1}\,Q^T.$$

For distributed-memory parallel computation with message-passing, the number of global reduction summation steps in the iterated CGS algorithm can be reduced by employing the Ruhe[8] loop-invariant. In particular, it is possible to write Step 4 of Algorithm 1 by combining the first two iterations of the re-orthogonalization as follows, given $r^{(1)} = Q^T a$

$$\begin{aligned} r^{(2)} &= r^{(1)} + Q^T a - Q^T Q r^{(1)} \\ &= Q^T a + Q^T a - Q^T Q r^{(1)} \\ &= Q^T a + (\,I - Q^T Q\,)(\,Q^T a\,). \end{aligned}$$

The above derivation implies that the projection matrix takes the form,

$$P^{IC} = I - Q(\,2I - Q^T Q\,)Q^T = I - Q T^{IC} Q^T, \quad T^{IC} = I - L - L^T, \tag{A1}$$

where L is a strictly lower triangular matrix

$$Q^T Q = I + L + L^T.$$

In the case of the MGS algorithm, the loop invariant form derived by Ruhe[8] can be used to update the columns of the matrix $r = R_j$ as follows,

$$r^{(k)} = (\,I + L_{:,1:j-1}\,)^{-1} Q^T_{:,1:j-1} a_j - (\,I + L_{:,1:j-1}\,)^{-1} L^T_{:,1:j-1} r^{(k-1)}.$$

A Neumann polynomial links the different Gram–Schmidt algorithms to the stability analyses and the loss of orthogonality relations. In addition, the equivalence of the MGS algorithm and compact WY representation follows from the lower triangular solve algorithm and finite matrix sum

$$T^{IM} = (\,I + L\,)^{-1} = I - L + L^2 - L^3 + L^4 - \cdots + L^{j-1}. \tag{A2}$$

We emphasize that the T^{IC} defined in Equation (A1) is different than the T^{IM} in Equation (A2). To establish the equivalence with the compact WY MGS formulation, consider the lower triangular solution algorithm for $y = (\,I + L\,)^{-1} b$, where the solution vector y is given by

$$\begin{aligned} y_1 &= b_1 \\ y_2 &= b_2 - l_{2,1}\, y_1 \\ y_3 &= b_3 - l_{3,1}\, y_1 - l_{3,2}\, y_2 \\ \vdots &= \cdots \\ y_n &= b_n - l_{n,1}\, y_1 - l_{n,2}\, y_2 - l_{n,3}\, y_3 - \cdots - l_{n,n-1}\, y_{n-1}. \end{aligned}$$

The elements of the matrix L^T contain the inner products $l_{ij} = q_i^T q_j$. Then the lower triangular solution algorithm above generates the higher order inner products as follows

$$
\begin{aligned}
y_1 &= b_1 \\
y_2 &= b_2 - l_{2,1} b_1 \\
y_3 &= b_3 - l_{3,1} b_1 - l_{3,2} \left(b_2 - l_{2,1}\, b_1 \right) \\
&\vdots = \cdots \\
y_n &= b_n - l_{n,1} b_1 - l_{n,2} \left(b_2 - l_{2,1}\, b_1 \right) - l_{n,3} \left(b_3 - l_{3,1}\, b_1 - l_{3,2} \left(b_2 - l_{2,1}\, b_1 \right) \right) - \cdots .
\end{aligned}
$$

APPENDIX. B

The algorithms presented in this manuscript are meant to be used and reproduced. A sample Matlab implementation is available under the URL https://github.com/kswirydo/LowSynchGMRESAlgorithms. Please cite this paper if using the code.