Parallel Computing 112 (2022) 102940

Contents lists available at ScienceDirect

Parallel Computing

journal homepage: www.elsevier.com/locate/parco

Low-synch Gram–Schmidt with delayed reorthogonalization for Krylov solvers

Daniel Bielich [a,*], Julien Langou [b], Stephen Thomas [c], Kasia Świrydowicz [d], Ichitaro Yamazaki [e], Erik G. Boman [e]

[a] University of Knoxville, Tennessee, United States of America
[b] University of Colorado, Denver, United States of America
[c] National Renewable Energy Laboratory, Golden, CO, United States of America
[d] Pacific Northwest National Laboratory, Richland, WA, United States of America
[e] Sandia National Laboratories, United States of America

ARTICLE INFO

Keywords:
Krylov methods
Nonsymmetric
Orthogonalization
Gram–Schmidt
Scalable solvers
Low synchronization
Global reduction
Exascale
Many-core architecture
GPU
Massively parallel

ABSTRACT

The parallel strong-scaling of iterative methods is often determined by the number of global reductions at each iteration. Low-synch Gram–Schmidt algorithms are applied here to the Arnoldi algorithm to reduce the number of global reductions and therefore to improve the parallel strong-scaling of iterative solvers for nonsymmetric matrices such as the GMRES and the Krylov–Schur iterative methods. In the Arnoldi context, the QR factorization is "left-looking" and processes one column at a time. Among the methods for generating an orthogonal basis for the Arnoldi algorithm, the classical Gram–Schmidt algorithm, with reorthogonalization (CGS2) requires three global reductions per iteration. A new variant of CGS2 that requires only one reduction per iteration is presented and applied to the Arnoldi algorithm. Delayed CGS2 (DCGS2) employs the minimum number of global reductions per iteration (one) for a one-column at-a-time algorithm. The main idea behind the new algorithm is to group global reductions by rearranging the order of operations. DCGS2 must be carefully integrated into an Arnoldi expansion or a GMRES solver. Numerical stability experiments assess robustness for Krylov–Schur eigenvalue computations. Performance experiments on the ORNL Summit supercomputer then establish the superiority of DCGS2 over CGS2.

1. Introduction

Let A be an $m \times m$ real-valued matrix. In this manuscript, A is employed in two parallel computations: (1) finding a solution of the linear system $Ax = b$ with a Krylov subspace method such as GMRES [1] and (2) finding the eigenvalues of A using Krylov–Schur [2]. In both instances, the Arnoldi algorithm is used to generate an orthonormal basis V_m for the Krylov subspace $\mathcal{K}_n$ and the matrix $H_{n+1,n}$ such that $AV_n = V_{n+1}H_{n+1,n}$. The Gram–Schmidt algorithm produces a QR decomposition of the matrix $B = [\, b,\ AV_n \,]$ for the Arnoldi algorithm. The size of this basis is $n \ll m$.

The orthogonality of the basis, Q_n, for the Krylov subspace $\mathcal{K}_n(B)$ is desirable for convergence of Krylov methods for linear systems and eigenvalue solvers. However, in finite-precision arithmetic, Q_n might "lose" orthogonality. The loss of orthogonality of the computed basis – as measured by $\|I - Q_n^T Q_n\|_F$ – may deviate substantially from machine precision $\mathcal{O}(\epsilon)$, (see Giraud et al. [3]). When linear independence is completely lost, the Krylov iterations, may fail to converge. For example, the GMRES iteration will stall and fail to converge if linear independence of the Krylov vectors is completely lost. This is the case when $\|S\|_2 = 1$ as described by Paige [4], where the matrix S was introduced in Paige et al. [5]. Also, in order to obtain backward stable eigenvalues from Krylov–Schur, Stewart [2] demonstrates that $\mathcal{O}(\epsilon)$ loss of orthogonality suffices.

Krylov linear system and eigenvalue solvers are often required for extreme scale physics simulations and implemented on parallel (distributed memory) machines. Their strong-scaling is limited by the number and frequency of global reductions, in the form of `MPI_AllReduce`. These communication patterns are expensive [6]. Our new algorithms are designed such that they require only one reduction per iteration to normalize each vector and apply projections. During the Arnoldi expansion, the vectors are processed by the orthogonalization scheme in a "left-looking" fashion [7–9]. The focus here is on methods that process the Krylov vectors one column at a time (as opposed to blocks).

* Corresponding author.
E-mail address: dbielich6888@gmail.com (D. Bielich).

https://doi.org/10.1016/j.parco.2022.102940
Received 15 October 2021; Received in revised form 15 March 2022; Accepted 12 May 2022
Available online 31 May 2022
0167-8191/Published by Elsevier B.V.

For a single program multiple data (SPMD) model of computation, classical Gram–Schmidt (CGS) requires two global reductions for each column vector (a projection step, followed by a vector normalization). In practice, however, CGS leads to numerical instability because the loss of orthogonality can be as large as $\mathcal{O}(\varepsilon)\kappa^2(A)$, where $\kappa(A)$ refers to the 2-norm condition number of the matrix A. This bound was conjectured for a long time and finally proven in two papers [10,11].

To reduce the loss of the orthogonality to machine precision $\mathcal{O}(\varepsilon)$, the CGS projection can be applied twice (CGS2) to reorthogonalize the basis vectors. This is related to the "twice is enough" result from Kahan and Parlett [12], where stability was proven for $n = 2$ vectors by Kahan and Parlett, and the proof was generalized to any n by Giraud et al. [10]. Given the assumption that $c\varepsilon\kappa(A) < 1$ for a given $m \times n$ input matrix A and constant $c = \mathcal{O}(m^2n^3)$, then CGS2 constructs columns orthogonal to machine precision (Theorem 2 in [10]). The improved numerical properties of CGS2 over CGS are at the expense of a 2× increase in floating-point operations (flops), from $2mn^2$ for CGS to $4mn^2$ for CGS2, and an 1.5× increase in global reductions, from 2 for CGS to 3 for CGS2.

Extensive numerical results are presented for the Krylov–Schur algorithm to demonstrate the numerical stability and accuracy of DCGS2-Arnoldi. Strong-scaling results are presented for the ORNL Summit supercomputer to demonstrate that the DCGS2 algorithm improves the CGS2 compute times by a factor of up to 2× on many-core architectures such as GPUs, while maintaining the same loss of orthogonality as the original CGS2-Arnoldi algorithm.

2. Low-synch Gram–Schmidt algorithms

A comprehensive review of Gram–Schmidt algorithms and their computational costs is given in [13]. These costs will be important considerations in strong scaling studies of these new algorithms on the ORNL Summit supercomputer.

The development of low-synch Modified Gram–Schmidt (MGS) and low-synch CGS2 was largely driven by applications that need stable, yet scalable solvers. Both MGS and CGS2 are stable orthogonalization schemes for a GMRES solver. Indeed, CGS2 produces an $\mathcal{O}(\varepsilon)$ loss of orthogonality, which suffices for GMRES to converge. Paige et al. [5] show that, despite a $\mathcal{O}(\varepsilon)\kappa(B)$ loss of orthogonality MGS-GMRES is backward stable for the solution of linear systems. Here, $\kappa(B)$ is the condition number $\kappa(B) = \sigma_{\max}(B)/\sigma_{\min}(B)$, where $\sigma_i(B)$ are the singular values of the matrix B.

While MGS performs half the number of flops of CGS2, the need for scalability has rendered MGS impractical as shown as early as 1998 in a study on a Cray T3D by Frayssé et al. [14]. For these two reasons (stable and more scalable), CGS2 is currently the de-facto orthogonalization scheme for iterative methods and is, for example, included in Trilinos [15], PETSc [16], Hypre [17], and Ginko [18].

As previously stated, the CGS2 algorithm requires three global reductions per iteration: one for the first projection, another for the second projection (reorthogonalization) and a third for the normalization. Our one-synch DCGS2 delays the reorthogonalization and the normalization of the current vector to the next iteration. The reorthogonalization delay to the next iteration in the QR framework is called "Stephen's trick" and is named after the third author of this paper.

To save one reduction, Kim and Chronopoulos [19] proposed either to use the "Pythagorean trick", (explained in the third paragraph of Section 3,) or to delay the normalization to the next iteration. This idea can be used in all Gram–Schmidt algorithms. With either one of these tricks, the number of synchronizations is reduced for CGS from 2 to 1, for CGS2 from 3 to 2, and for Level-2 MGS from 2 to 1, etc. Carson et al. [20] generalize the Pythagorean trick to block Gram–Schmidt algorithms.

Delaying the normalization and/or reorthogonalization to the next iteration implies some complications for an Arnoldi expansion that are not present in a QR factorization algorithm. Whereas, in a QR factorization the columns to be processed are independent from Q and R, the Arnoldi algorithm relies on a "finished" vector q to perform Aq and then provides this vector to the orthogonalization scheme. Because these computations are delayed, Arnoldi is passed an "unfinished" vector q to perform Aq. Therefore, the computation of q is completed at the next iteration. Aq and other quantities are then corrected accordingly. The impact of delaying normalization in the Arnoldi iteration is explained in Hernandez et al. [21] and the resulting algorithm is called "Arnoldi with Delayed Reorthogonalization" (ADR). In this paper, a correction is introduced with the delayed reorthogonalization, and is called the "Arnoldi trick". A delayed classical Gram–Schmidt algorithm with reorthogonalization, was proposed by Hernandez et al. [21]. This is similar to our algorithm (DCGS2), and referred to as DCGS2-HRT. The DCGS2-HRT-Arnoldi algorithm is missing Stephen's trick and the Arnoldi correction trick and is therefore quite unstable.

One-reduce MGS and CGS Gram–Schmidt algorithms are presented in [22]. These are based upon the application of an operator

$$P = I - Q\,T\,Q^T$$

where Q is $m \times n$, I is the identity, and T is an $n \times n$ correction matrix. To obtain a one-reduce algorithm, the normalization is either delayed to the next iteration or employs the "Pythagorean trick". T is obtained for MGS from $L = \text{tril}(\,Q^TQ,\,-1\,)$, the strictly lower triangular part of Q^TQ. Note that, because Q has almost orthonormal columns, the norm of L is small, and T is close to I.

In the case of the one-reduce CGS algorithm,

$$P = I - Q\,T\,Q^T, \quad T = I$$

which, assuming Q has linearly independent columns, is the orthogonal projection onto the orthogonal complement of Q. Then, one can also employ

$$T = 2I - Q^TQ = I - L - L^T$$

where T is derived from two applications of the operator

$$P = (\,I - Q\,Q^T\,)(\,I - Q\,Q^T\,) = I - Q\,(\,2I - Q^TQ\,)\,Q^T$$

Note that T above is the first order approximation of

$$P = I - Q\,T\,Q^T, \quad T = (\,Q^TQ\,)^{-1} = (\,I + L + L^T\,)^{-1}$$

The inverse compact WY form for MGS is the lower triangular matrix $T = (I + L)^{-1}$, analogous to Puglisi [23]. An upper triangular T for MGS is derived in [24] and corresponds to the Schreiber and Van Loan compact WY Householder [25] representation. This was recently generalized to block Gram–Schmidt algorithms by Barlow [26]. The inverse compact WY form of MGS was recently derived in [22]. One can also use $T = I - L$ when $\|L^p\|_F = \mathcal{O}(\varepsilon^p)\kappa_F^p(B)$ in the MGS-GMRES algorithm, which is a truncated Neumann series for $(I + L)^{-1}$. While there are many possible MGS variants, these are performing one pass of the operator P on a column vector.

Our DCGS2 algorithm performs two passes of the operator on each vector. This results in $\mathcal{O}(\varepsilon)$ loss of orthogonality that the one-pass variants cannot achieve. There are two reasons for this. The first reason is that, if Q does not have orthonormal columns, then $P = I - QQ^T$ is not the orthogonal projection onto the orthogonal complement of Q; however, provided Q has almost orthonormal columns, repeated applications of the operator P leads to the orthogonal projection on the orthogonal complement of Q. In other words, if $\rho(I - Q^TQ) < 1$, (where ρ is the spectral radius,) then it follows that

$$\lim_{k\to\infty}(\,I - QQ^T\,)^k = (\,I - Q\,(\,Q^TQ\,)^{-1}\,Q^T\,)$$

Therefore, repeating the projection step enables correction for the potential lack of orthogonality in Q. The second reason is that, even with the "perfect" projection P, given a, a column of A, the computation of w with $w = Pa$ may suffer from cancellation errors leading to w being relatively far from orthogonal with Q, therefore a second pass (which

will, provided that A is numerically nonsingular, have provably less cancellation errors), is necessary.

To obtain a one-synch algorithm and two passes per vector, in our DCGS2 algorithm, the second pass is delayed to the next iteration and grouped with the first pass on the next vector. The resulting algorithm combines the two steps, (1) first pass on current vector and (2) second pass on previous vector, into one global reduction and is referred to as the delayed DCGS2. DCGS2 is explained in detail in Section 3.

3. DCGS2 algorithm for the QR decomposition

In this section, the classical Gram–Schmidt algorithm to compute the QR decomposition of an $m \times n$ matrix A is presented. Algorithm 1 displays the steps for the jth iteration of CGS2. First, the column vector a_j is projected onto the orthogonal complement of $Q_{1:j-1}$

$$\begin{aligned} w_j &= \left(I - Q_{1:j-1}\, Q_{1:j-1}^T \right) a_j \\ &= a_j - Q_{1:j-1}\, S_{1:j-1,j} \quad \text{where} \quad S_{1:j-1,j} = Q_{1:j-1}^T a_j, \end{aligned}$$

followed by a second application of the operator in the form

$$u_j = w_j - Q_{1:j-1}\, C_{1:j-1,j} \quad \text{where} \quad C_{1:j-1,j} = Q_{1:j-1}^T w_j.$$

Finally, the vector u_j is normalized to produce the vector q_j,

$$q_j = u_j/\alpha_j \quad \text{where} \quad \alpha_j = \| u_j \|_2.$$

The QR decomposition of $A_{1:j} = Q_{1:j} R_{1:j,1:j}$, is produced at the end of the jth iteration, where

$$R_{1:j-1,j} = S_{1:j-1,j} + C_{1:j-1,j} \quad \text{and} \quad R_{j,j} = \alpha_j.$$

The jth iteration of CGS2 is displayed in Algorithm 1. The three global reductions appear in steps 1, 3, and 5.

Algorithm 1 Classical Gram–Schmidt with reorthogonalization (CGS2)

```
        // first projection
1:      S_{1:j-1,j} = Q_{1:j-1}^T a_j          // global reduction
2:      w_j = a_j - Q_{1:j-1} S_{1:j-1,j}

        // second projection
3:      C_{1:j-1,j} = Q_{1:j-1}^T w_j          // global reduction
4:      u_j = w_j - Q_{1:j-1} C_{1:j-1,j}

        // normalization
5:      α_j = || u_j ||_2                      // global reduction
6:      q_j = u_j / α_j

        // representation R_j
7:      R_{1:j-1,j} = S_{1:j-1,j} + C_{1:j-1,j}
8:      R_{j,j} = α_j
```

The jth iteration of DCGS2 is displayed in Algorithm 2. In order to perform one global reduction, the second projection and normalization are delayed to the next iteration. To understand how to integrate DCGS2 in a QR factorization, refer to the implementation https://github.com/dbielich/DCGS2.git.

The norm of u_j, α_j, is computed using the "Pythagorean trick". The explanation goes as follows

$$\begin{aligned} \alpha_j^2 &= u_j^T u_j \\ &= \left(w_j - Q_{1:j-1}\, C_{1:j-1,j} \right)^T \left(w_j - Q_{1:j-1}\, C_{1:j-1,j} \right) \\ &= w_j^T w_j - 2 C_{1:j-1,j}^T\, C_{1:j-1,j} \\ &+ C_{1:j-1,j}^T \left(Q_{1:j-1}^T\, Q_{1:j-1} \right) C_{1:j-1,j} \\ &= w_j^T w_j - C_{1:j-1,j}^T\, C_{1:j-1,j} \\ &- C_{1:j-1,j}^T \left(I - Q_{1:j-1}^T\, Q_{1:j-1} \right) C_{1:j-1,j} \\ &= w_j^T w_j - C_{1:j-1,j}^T\, C_{1:j-1,j} \end{aligned}$$

where $C_{1:j-1,j} = Q_{1:j-1}^T\, w_j$ is employed. Now, neglecting the term with $\left(I - Q_{1:j-1}^T\, Q_{1:j-1} \right)$ and using the fact that $\beta_j = w_j^T w_j$, the norm of the updated vector u_j, α_j, is computed as

$$\alpha_j = \left\{ \beta_j - C_{1:j-1,j}^T\, C_{1:j-1,j} \right\}^{1/2}$$

This is the Pythagorean trick and corresponds to Step 2 in Algorithm 2.

Note that the normalization was at first delayed to the next iteration. The initial idea was to have a pipeline of length three. In a single reduction, each iteration would perform: (1) the first projection of the current vector a_j, (2) the second projection of the previous vector w_{j-1} (delayed by one iteration), (3) the normalization of the previous–previous vector u_{j-2} (delayed by two iterations). The code with a pipeline of length three would have been quite complex. The Pythagorean trick reduces the length by one. Thus, the pipeline is of length two (instead of three) because the Pythagorean trick allows us to compute the norm of u_{j-1} at iteration j by utilizing w_{j-1} (which is available at the start of iteration j) instead of u_{j-1} (which is not available at the start of iteration j). Hence the Pythagorean trick in DCGS2 leads to a pipeline of length two. Initially it was thought that the "Pythagorean trick" should be avoided because it could lead to a less stable algorithm. Numerical experiments were performed with the Pythagorean trick (see Section 6.1), and the results were satisfactory, thus the Pythagorean trick was deemed appropriate in this context.

For the column vector a_j, the scalar $S_{j-1,j}$ is computed with $S_{j-1,j} = w_{j-1}^T\, a_j$ instead of $q_{j-1}^T\, a_j$. This is Stephen's trick, it captures the computation $q_{j-1}^T a_j$ using w_{j-1} instead of q_{j-1} and results in a corrected projection step within the Gram–Schmidt process,

$$\begin{aligned} q_{j-1}^T\, a_j &= \frac{1}{\alpha_{j-1}} \left(w_{j-1} - Q_{1:j-2} C_{1:j-2,j-1} \right)^T a_j \\ &= \frac{1}{\alpha_{j-1}} \left(w_{j-1}^T a_j - C_{1:j-2,j-1}^T\, Q_{1:j-2}^T\, a_j \right) \\ &= \frac{1}{\alpha_{j-1}} \left(S_{j-1,j} - C_{1:j-2,j-1}^T\, S_{1:j-2,j} \right) \end{aligned}$$

The correction of the vector norm corresponds to Steps 3 and 6 in Algorithm 2 given below.

Algorithm 2 Delayed Classical Gram–Schmidt with reorthogonalization (DCGS2)

```
1:      [ Q_{1:j-2}, w_{j-1} ]^T [ w_{j-1}, a_j ]      // global reduction

        S_{1:j-2,j} = Q_{1:j-2}^T a_j        and   S_{j-1,j} = w_{j-1}^T a_j
        C_{1:j-2,j-1} = Q_{1:j-2}^T w_{j-1}  and   β_{j-1} = w_{j-1}^T w_{j-1}

        // delayed correction steps
2:      α_{j-1} = { β_{j-1} - C_{1:j-2,j-1}^T C_{1:j-2,j-1} }^{1/2}
3:      S_{j-1,j} = (1/α_{j-1}) ( S_{j-1,j} - C_{1:j-2,j-1}^T S_{1:j-2,j} )

        // projection & delayed reorthogonalization
4:      [ u_{j-1}, w_j ] = [ w_{j-1}, a_j ] - Q_{1:j-2} [ C_{1:j-2,j-1}, S_{1:j-2,j} ]
5:      q_{j-1} = (1/α_{j-1}) u_{j-1}
6:      w_j = w_j - q_{j-1} S_{j-1,j}

        // representation R_{j-1}
7:      R_{1:j-2,j-1} = S_{1:j-2,j-1} + C_{1:j-2,j-1}
8:      R_{j-1,j-1} = α_{j-1}
```

For the nth iteration, CGS2 is applied and incurs two additional global reductions.

4. DCGS2 algorithm for the Arnoldi expansion

Algorithm 3 displays CGS2 for the Arnoldi expansion. The only difference from the QR decomposition in Algorithm 1 is that the next basis vector v_j is generated by applying a matrix–vector product to the previously normalized column vector q_{j-1}. At the end of iteration $j-1$, in exact arithmetic, the matrices would satisfy the Arnoldi expansion,

$$A\,Q_{1:j-2} = Q_{1:j-1}\,H_{1:j-1,1:j-2}. \tag{1}$$

Algorithm 3 Arnoldi (CGS2)

// generation of next vector
$v_j = A\,q_{j-1}$

// first projection
$S_{1:j-1,j} = Q_{1:j-1}^T\,v_j$ *// global reduction*
$w_j = v_j - Q_{1:j-1}\,S_{1:j-1,j}$

// second projection
$C_{1:j-1,j} = Q_{1:j-1}^T\,w_j$ *// global reduction*
$u_j = w_j - Q_{1:j-1}\,C_{1:j-1,j}$

// normalization
$\alpha_j = \|\,u_j\,\|_2$ *// global reduction*
$q_j = \frac{1}{\alpha_j}\,u_j$

// representation H_j
$H_{1:j-1,j} = S_{1:j-1,j} + C_{1:j-1,j}$
$H_{j,j} = \alpha_j$

A one-reduction DCGS2-Arnoldi will now be derived. The representation error and loss of orthogonality are maintained at the same level as the CGS2-Arnoldi.

With lagged vector updates, the next basis vector is generated by applying a matrix–vector product to the current vector. Namely, the next vector v_j is computed as $A\,w_{j-1}$ by using the vector w_{j-1} instead of q_{j-1}, where q_{j-1} is the previously constructed orthogonal column. Thus, an effective strategy is required to compute w_j from Aw_{j-1} and also to generate the Hessenberg matrix H_j in the Arnoldi expansion.

After a delay of one iteration, the vector q_{j-1}, is computed using w_{j-1} as follows

$$q_{j-1} = \frac{1}{\alpha_{j-1}}\,(\,w_{j-1} - Q_{1:j-2}C_{1:j-2,j-1}\,) \tag{2}$$

Eq. (2) may also be interpreted as a QR factorization of the matrix $W_{1:j-1}$, with columns $[\,w_1,\,\ldots,\,w_{j-1}\,]$

$$Q_{1:j-1} = W_{1:j-1}C_{1:j-1,1:j-1}^{-1}, \tag{3}$$

where C is an upper triangular matrix.

Multiplying (2) by A from the left, it follows that

$$\begin{aligned} v_j &= A\,q_{j-1} \\ &= \frac{1}{\alpha_{j-1}}\,(\,A\,w_{j-1} - A\,Q_{1:j-2}\,C_{1:j-2,j-1}\,) \\ &= \frac{1}{\alpha_{j-1}}\,(\,A\,w_{j-1} - Q_{1:j-1}\,H_{1:j-1,1:j-2}\,C_{1:j-2,j-1}\,) \end{aligned} \tag{4}$$

Next the vector w_j is computed, which is the vector produced after projection of v_j onto the basis vectors in $Q_{1:j-1}$,

$$\begin{aligned} w_j &= A\,q_{j-1} - Q_{1:j-1}\,Q_{1:j-1}^T\,A\,q_{j-1} \\ &= \frac{1}{\alpha_{j-1}}\,(\,Aw_{j-1} - Q_{1:j-1}H_{1:j-1,1:j-2}C_{1:j-2,j-1}\,) \\ &\quad -Q_{1:j-1}Q_{1:j-1}^T \\ &\quad\times\ \frac{1}{\alpha_{j-1}}\,(\,Aw_{j-1} - Q_{1:j-1}\,H_{1:j-1,1:j-2}\,C_{1:j-2,j-1}\,) \end{aligned} \tag{5}$$

$$\begin{aligned} w_j &= \frac{1}{\alpha_{j-1}}\,\left(\,A\,w_{j-1} - Q_{1:j-1}\,Q_{1:j-1}^T\,A\,w_{j-1}\,\right) \\ &\quad + \frac{1}{\alpha_{j-1}}\,Q_{1:j-1}\,\left(\,I - Q_{1:j-1}^T\,Q_{1:j-1}\,\right)\,H_{1:j-1,1:j-2}\,C_{1:j-2,j-1} \end{aligned}$$

The last term is dropped from (5), for two reasons,

- DCGS2 is constructed such that the loss of orthogonality $\|I - Q_{1:j-1}^T\,Q_{1:j-1}\|_F$ is $\mathcal{O}(\varepsilon)$, and
- $C_{1:j-2,j-1}/\alpha_{j-1}$ is expected to be $\mathcal{O}(\varepsilon)\kappa(A)$. Hence, when $\kappa(A) \le \mathcal{O}(1/\varepsilon)$, the norm of the term is $\mathcal{O}(1)$.

Therefore, at this point (5) becomes an approximation and

$$\begin{aligned} w_j &= \frac{1}{\alpha_{j-1}}\,\left(\,A\,w_{j-1} - Q_{1:j-1}\,Q_{1:j-1}^T\,A\,w_{j-1}\,\right) \\ &= \frac{1}{\alpha_{j-1}}\,\left(\,A\,w_{j-1} - Q_{1:j-2}\,Q_{1:j-2}^T\,A\,w_{j-1}\,\right) \\ &\quad - \frac{1}{\alpha_{j-1}}\,q_{j-1}\,q_{j-1}^T\,A\,w_{j-1} \end{aligned}$$

Noting that $S_{1:j-2,j} = Q_{1:j-2}^T\,A\,w_{j-1}$, it follows that

$$w_j = \frac{1}{\alpha_{j-1}}\,\left(\,A\,w_{j-1} - Q_{1:j-2}\,S_{1:j-2,j} - q_{j-1}\,q_{j-1}^T\,A\,w_{j-1}\,\right)$$

Finally, from (2), it is possible to compute

$$\begin{aligned} q_{j-1}^T\,A\,w_{j-1} &= \frac{1}{\alpha_{j-1}}\,(\,w_{j-1} - Q_{1:j-2}\,C_{1:j-2,j-1}\,)^T\,Aw_{j-1} \\ &= \frac{1}{\alpha_{j-1}}\,\left(\,w_{j-1}^T A\,w_{j-1} - C_{1:j-2,j-1}^T\,Q_{1:j-2}^T\,A\,w_{j-1}\,\right) \\ &= \frac{1}{\alpha_{j-1}}\,\left(\,S_{j-1,j} - C_{1:j-2,j-1}^T\,S_{1:j-2,j}\,\right) \end{aligned}$$

where $S_{j-1,j} = w_{j-1}^T Aw_{j-1}$. This step is Stephen's trick in the context of Arnoldi.

After substitution of this expression, it follows that

$$\begin{aligned} w_j &= \frac{1}{\alpha_{j-1}}\,(\,A\,w_{j-1} - Q_{1:j-2}\,S_{1:j-2,j}\,) \\ &\quad - \frac{1}{\alpha_{j-1}^2}\,q_{j-1}\,\left(\,S_{j-1,j} - C_{1:j-2,j-1}^T\,S_{1:j-2,j}\,\right) \\ &= \frac{1}{\alpha_{j-1}}\,(A\,w_{j-1} - Q_{1:j-1}\,S_{1:j-1,j}) \end{aligned} \tag{6}$$

where

$$S_{j-1,j} = \frac{1}{\alpha_{j-1}}\,\left(\,S_{j-1,j} - C_{1:j-2,j-1}^T\,S_{1:j-2,j}\,\right).$$

is corrected prior to its application. The $(j-1)$th column of the Hessenberg matrix H is computed as follows and satisfies the Arnoldi relation (1). First, reorder (6) into a factorization form

$$\begin{aligned} A\,w_{j-1} &= Q_{1:j-2}\,S_{1:j-2,j} \\ &\quad + \frac{1}{\alpha_{j-1}}\,q_{j-1}\,\left(\,S_{j-1,j} - C_{1:j-2,j-1}^T\,S_{1:j-2,j}\,\right) + \alpha_{j-1}\,w_j \end{aligned} \tag{7}$$

From (2), it also follows that

$$w_j = Q_{1:j-1}\,C_{1:j-1,j} + \alpha_j\,q_j \tag{8}$$

which represents the orthogonalization of the vector w_j. By replacing w_j in (7) with the expression in (8), obtain

$$\begin{aligned} Aw_{j-1} &= Q_{1:j-2}\,S_{1:j-2,j} \\ &\quad + \frac{1}{\alpha_{j-1}}\,q_{j-1}\,\left(\,S_{j-1,j} - C_{1:j-2,j-1}^T S_{1:j-2,j}\,\right) \\ &\quad + \alpha_{j-1}\,Q_{1:j-1}\,C_{1:j-1,j} + \alpha_j\,\alpha_{j-1}\,q_j \\ Aw_{j-1} &= Q_{1:j-2}\,(\,S_{1:j-2,j} + \alpha_{j-1}\,C_{1:j-2,j}\,) \\ &\quad + \frac{1}{\alpha_{j-1}}\,q_{j-1}\,\left(\,S_{j-1,j} - C_{1:j-2,j-1}^T S_{1:j-2,j}\,\right) \\ &\quad + \alpha_{j-1}\,C_{j-1,j}\,q_{j-1} + \alpha_j\,\alpha_{j-1}\,q_j. \end{aligned} \tag{9}$$

This is the representation of $A\,w_{j-1}$ in the Krylov subspace spanned by the orthogonal basis vectors $Q_{1:j}$ using the matrices S and C. However, the representation of $A\,q_{j-1}$ in $Q_{1:j}$ with the matrix H is still required. Namely, write (5) as

$$\begin{aligned} Aq_{j-1} &= \frac{1}{\alpha_{j-1}}\left(Aw_{j-1} - Q_{1:j-2}\,H_{1:j-2,1:j-2}\,C_{1:j-2,j-1}\right) \\ &\quad - \frac{1}{\alpha_{j-1}}\,q_{j-1}\,H_{j-1,j-2}\,C_{j-2,j-1} \end{aligned}$$

$H_{j-1,j-1}$ is now computed using C_{j-2} and $H_{1:j-1,j-2}$.

Replacing Aw_{j-1} with (9), it follows that

$$\begin{aligned} Aq_{j-1} &= Q_{1:j-2}\left(\frac{1}{\alpha_{j-1}}\,S_{1:j-2,j} + C_{1:j-2,j}\right) \\ &\quad + \frac{1}{\alpha_{j-1}^2}\,q_{j-1}\left(S_{j-1,j} - C_{1:j-2,j-1}^T\,S_{1:j-2,j}\right) \\ &\quad + \alpha_j\,q_j - \frac{1}{\alpha_{j-1}}\,Q_{1:j-2}\,H_{1:j-2,1:j-2}\,C_{1:j-2,j-1} \\ &\quad + C_{j-1,j}\,q_{j-1} - \frac{1}{\alpha_{j-1}}\,H_{j-1,j-2}\,C_{j-2,j-1}\,q_{j-1} \end{aligned} \tag{10}$$

To summarize

1. $A\,q_{j-1} = Q_{1:j-2}\,A_{1:j-2,j} + q_{j-1}\,s_{j-1,j} + \alpha_j\,q_j$ is a standard QR decomposition obtained by a Gram–Schmidt process.
2. $C_{1:j-2,j}$ and $C_{j-1,j}$ are the standard reorthogonalization terms in the representation equation,
3. $C_{1:j-2,j-1}^T\,S_{1:j-2,j}$ is Stephen's trick.
4. $H_{1:j-2,1:j-2}C_{1:j-2,j-1}$ and $H_{j-1,j-2}C_{j-2,j-1}$ are the representation error correction terms.
5. $\frac{1}{\alpha_{j-1}}$ is due to using unnormalized quantities and these must be corrected by scaling.

Items 1, 2 and 3 are present in both the QR decomposition and Arnoldi expansion. Items 4 and 5 are specific to Arnoldi.

According to (10), in order to obtain the $(j-1)$th column of the Arnoldi relation (1), the column $H_{1:j,j-1}$ is computed as follows

$$\begin{aligned} H_{1:j-2,j-1} &= \frac{1}{\alpha_{j-1}}\,S_{1:j-2,j} + C_{1:j-2,j} \\ &\quad - \frac{1}{\alpha_{j-1}}\,H_{1:j-2,1:j-2}\,C_{1:j-2,j-1} \\ &= C_{1:j-2,j} + \frac{1}{\alpha_{j-1}}\left(S_{1:j-2,j} - H_{1:j-2,1:j-2}\,C_{1:j-2,j-1}\right) \\ H_{j-1,j-1} &= \frac{1}{\alpha_{j-1}^2}\left(S_{j-1,j} - C_{1:j-2,j-1}^T\,S_{1:j-2,j}\right) + C_{j-1,j} \\ &\quad - \frac{1}{\alpha_{j-1}}\,H_{j-1,j-2}\,C_{j-2,j-1} \\ H_{j,j-1} &= \alpha_j \end{aligned}$$

Finally, the DCGS2-Arnoldi is presented in Algorithm 4.

Algorithm 4 Arnoldi (DCGS2)

1: $[\,Q_{1:j-2},\,w_{j-1}\,]^T\,[\,w_{j-1},\,Aw_{j-1}\,]$ *// global reduction*

2: $S_{1:j-2,j} = Q_{1:j-2}^T Aw_{j-1}$ and $S_{j-1,j} = w_{j-1}^T Aw_{j-1}$

3: $C_{1:j-2,j-1} = Q_{1:j-2}^T w_{j-1}$ and $\beta_{j-1} = w_{j-1}^T w_{j-1}$

// delayed correction terms

4: $\alpha_{j-1} = \left\{\,\beta_{j-1} - C_{1:j-2,j-1}^T C_{1:j-2,j-1}\,\right\}^{1/2}$

5: $S_{j-1,j} = \frac{1}{\alpha_{j-1}}\left(\,S_{j-1,j} - C_{1:j-2,j-1}^T\,S_{1:j-2,j}\,\right)$

// projection & delayed reorthogonalization

6: $[\,u_{j-1}, w_j\,] = [\,w_{j-1}, Aw_{j-1}\,] - Q_{1:j-2}\,[\,C_{1:j-2,j-1}, S_{1:j-1,j}\,]$

7: $q_{j-1} = \frac{1}{\alpha_{j-1}}\,u_{j-1}$

8: $w_j = \frac{1}{\alpha_{j-1}}\left(w_j - q_{j-1}\,S_{j-1,j}\right)$

// representation H_{j-1}

9: $H_{1:j-2,j-2} = K_{1:j-2,j-2} + C_{1:j-2,j-1}$

10: $K_{1:j-1,j-1} = \frac{1}{\alpha_{j-1}}\left(S_{1:j-1,j} - H_{1:j-1,1:j-2}\,C_{1:j-2,j-1}\right)$

11: $H_{j-1,j-2} = \alpha_{j-1}$

Step 6 in Algorithm 4 can be separated for their mathematical derivation, but is combined into a Level 2.5 BLAS operation (one DGEMM with 2 columns versus two DGEMV). Thus, the $j-1$th orthogonal column is employed in step 9 and not in step 7. The Level 2.5 DGEMM operation is of size $j-2\times 2$, (ignoring the $j-1$th column of step 9) and once q_{j-1} is available, a DAXPY is performed to finish the current projection on w_j. In addition, the $1/\alpha_{j-1}$ scaling is applied in one step. For practical implementation, refer to https://github.com/dbielich/DCGS2.git.

5. Computation and communication costs

The computation and communication costs of the algorithms are listed in Tables 1 and 2. Although theoretically equivalent, they exhibit different behavior in finite precision arithmetic. All the schemes, except MGS, are based upon cache-blocked matrix operations. MGS applies elementary rank–1 projection matrices sequentially to a vector and does not take advantage of the DGEMM matrix–matrix multiplication kernel. In addition, this algorithm requires one global reduction (MPI_AllReduce) in the inner-most loop to apply a rank-1 projection matrix. Thus, j global reductions are required at iteration $j-1$. The implementation of ICWY-MGS batches the projections together and computes one row of the strictly lower triangular matrix, see Świrydowicz et al. [22],

$$L_{k-1,1:k-2} = \left(\,Q_{1:k-2}^T\,q_{k-1}\,\right)^T.$$

The resulting inverse compact WY operator P is given by

$$P\,a = \left(\,I - Q_{1:j-1}\,T_{1:j-1,1:j-1}\,Q_{1:j-1}^T\,\right)\,a$$

where the triangular correction matrix is given by

$$T_{1:j-1,1:j-1} = (\,I + L_{1:j-1,1:j-1}\,)^{-1}, \quad T_{1:j-1,1:j-1} \approx (\,Q_{1:j-1}^T\,Q_{1:j-1}\,)^{-1}$$

The implied triangular solve requires an additional $(j-1)^2$ flops at iteration $j-1$ and thus leads to a slightly higher operation count compared to the original MGS orthogonalization scheme, the operation $Q_{1:k-2}^T q_{k-1}$ increases ICWY-MGS complexity by mn^2 ($3mn^2$ total) but reduces synchronizations from $j-1$ at iteration j to 1. This reasoning also follows for the CWY-MGS algorithm, which is 1.5× more expensive compared to the sequential implementation of MGS. However, only one global reduction is required per iteration, and the amount of inter-process communication does not depend upon the number of rank–1 projections applied at each iteration.

In the case of the DCGS2 algorithm, the symmetric correction matrix T_{j-1} was derived in Appendix 1 of [22] and is given by

$$T_{1:j-1,1:j-1} = I - L_{1:j-1,1:j-1} - L_{1:j-1,1:j-1}^T$$

This form of the operator was employed in the s-step and pipelined GMRES described in Yamazaki et al. [27]. When the matrix $T_{1:j-1,1:j-1}$ is split into $I - L_{1:j-1,1:j-1}$ and $L_{1:j-1,1:j-1}^T$ and applied across two iterations of the DCGS2 algorithm, the resulting loss of orthogonality is $\mathcal{O}(\varepsilon)$ in practice.

Block generalizations of the DGCS2 and CGS2 algorithm are presented in Carson et al. [20,28]. These papers generalize the Pythagorean trick to block form and derive BCGS-PIO and BCGS-PIP algorithms with the more favorable communication patterns described herein. An

Table 1
Cost per iteration for Gram–Schmidt algorithms. Where p is the number of processes used.

Scheme	Flops per iter	Synchs	Bandwidth
MGS Level 1	$4(m/p)j$	j	j
MGS Level 2	$6(m/p)j + j^2$	1	$2j$
CGS	$4(m/p)j$	2	j
CGS2	$8(m/p)j$	3	$2j$
CGS2 (lagged norm)	$8(m/p)j$	2	$2j$
DCGS2-HRT	$8(m/p)j$	1	$2j$
DCGS2 (QR)	$8(m/p)j$	1	$2j$
DCGS2 (Arnoldi)	$8(m/p)j + j^2$	1	$2j$

Table 2
Total cost of Gram–Schmidt algorithms. Where p is the number of processes used.

Scheme	Flops per iter	Synchs	Bandwidth
MGS Level 1	$2(m/p)n^2$	$\frac{1}{2}n^2$	$\frac{1}{2}n^2$
MGS Level 2	$3(m/p)n^2 + \frac{1}{3}n^3$	n	n^2
CGS	$2(m/p)n^2$	$2n$	$\frac{1}{2}n^2$
CGS2	$4(m/p)n^2$	$3n$	n^2
CGS2 (lagged norm)	$4(m/p)n^2$	$2n$	n^2
DCGS2-HRT	$4(m/p)n^2$	n	n^2
DCGS2 (QR)	$4(m/p)n^2$	n	n^2
DCGS2 (Arnoldi)	$4(m/p)n^2 + \frac{1}{3}n^3$	n	n^2

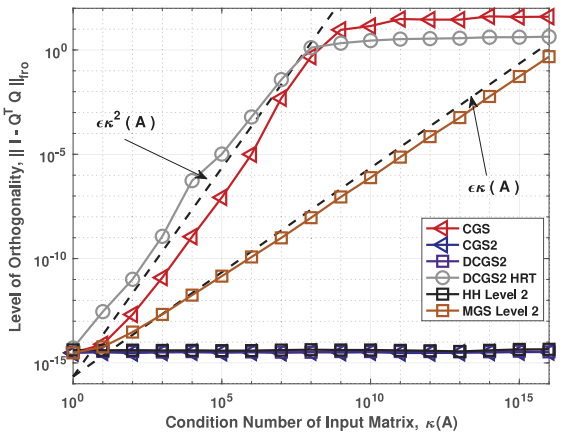

Fig. 1. Loss of orthogonality comparison for a QR factorization, $m = 500$, $n = 100$.

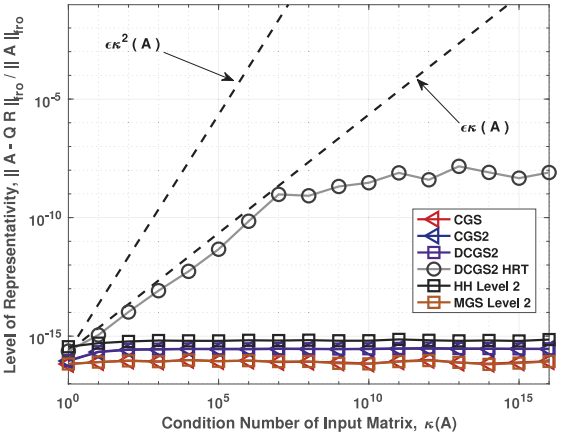

Fig. 2. Loss of representation comparison with increasing condition numbers, $m = 500$, $n = 100$.

analysis of the backward stability of the these block Gram–Schmidt algorithms is also presented.

Table 3
Loss of orthogonality (LOO).

Scheme	LOO	Proven
MGS Level 1	$\mathcal{O}(\varepsilon)\kappa(A)$	[30]
MGS Level 2	$\mathcal{O}(\varepsilon)\kappa(A)$	Conjectured
CGS	$\mathcal{O}(\varepsilon)\kappa^2(A)$	[10]
CGS2	$\mathcal{O}(\varepsilon)$	[10]
CGS2 (lagged norm)	$\mathcal{O}(\varepsilon)$	Conjectured
DCGS2-HRT	$\mathcal{O}(\varepsilon)\kappa^2(A)$	Conjectured
DCGS2 (QR)	$\mathcal{O}(\varepsilon)$	Conjectured
DCGS2 (Arnoldi)	$\mathcal{O}(\varepsilon)$	Conjectured

It is important to note that there are a variety of ways to implement a compact WY MGS algorithm. The correction matrix T can be formed recursively from a block triangular inverse, as in the compact WY MGS derived by Björck [29]. Barlow [26] employs the inverse compact ICWY form of Puglisi [23]. However, a lagged normalization is not applied and two reductions are required. These are summarized in [22]. For all results reported herein employing MGS, these are based upon CWY-MGS. Except when using the one-reduce GMRES solver with lagged normalization, where ICWY-MGS is applied. For this reason Tables 1, 2, and 3 refer to the MGS implementation as Level 2 versus ICWY-MGS. In addition, MGS Level 1 refers to the sequential BLAS implementation.

Both CGS and CGS2 are based upon matrix–vector operations. CGS applies a single projection, and then normalizes, requiring two separate steps. This projection step consists of two `DGEMV` kernel calls and one `DDOT` for the normalization. CGS suffers from at least an $\mathcal{O}(\varepsilon)\kappa^2(A)$ loss of orthogonality. CGS2 achieves $\mathcal{O}(\varepsilon)$ through two passes (see Fig. 1). The additional projection within CGS2 accounts for one additional global reduction per iteration and an additional $4(m/p)j$ operations.

DCGS2 requires one reduction and employs matrix–matrix multiplies for the computation in a tall-and-skinny `DGEMM`. This leads to the higher sustained execution rate of DCGS2 (e.g. 2× the GigaFlop/sec). In the context of Arnoldi, DCGS2 requires an additional j^2 flops at the jth iteration. The additional cost is due to the Arnoldi representation trick described in Section 4. The representation error correction terms require an additional n^2 operations from a matrix–vector product with the Hessenberg matrix.

6. Numerical results

In this section, the numerical stability of the Arnoldi algorithm is investigated for the different orthogonalization schemes. In addition to the method studied in Section 5 (MGS, CGS, CGS2, DCGS2, DCGS2-HRT), we also study Householder-based methods (HH).

The methodology for the numerical stability analysis is presented in Section 6.1 along with the experiments. The same methodology is employed in Section 7. Four stability metrics are examined,

1. representation error
2. loss of orthogonality
3. forward error in the eigenvalue solutions, < threshold
4. dimension of converged invariant subspace, < threshold

The metrics (1) and (2) are sufficient to analyze the stability of an orthogonalization scheme. However to give a broader perspective the metrics (3) and (4) are also examined. Additional metrics that can be considered are:

1. convergence of GMRES
2. achievable backward error of GMRES
3. number of eigenvalue-pairs (Ritz values) with a backward error < threshold (see Hernandez et al. [21])

The convergence of GMRES and the achievable backward error are informative metrics, however, Paige et al. [5] proved that GMRES (with one right-hand side) only needs an $\mathcal{O}(\varepsilon)\,\kappa(B)$ LOO to converge.

Table 4
Differential operator specs for $k = 50$, $m = k^2 = 2500$.

$\|A\|_2$	7.99e+00
Cond(A)	3.32e+02
Cond(V)	3.96e+11
Cond(W)	3.74e+11
$\|A^TA - AA^T\|_F / \|A\|_F^2$	2.81e−04
$\max_i($ Cond(λ_i))	1.46e+10
$\min_i($ Cond(λ_i))	2.38+02

Therefore, GMRES is tolerant of "bad" orthogonalization schemes and is not a stringent enough test.

The number of eigenvalue-pairs with a backward error less than a threshold should not be used to assess the quality of an orthogonalization scheme because, for example, a scheme that always returns the same eigenvalue-pair n times would score the highest possible score (n) according to this metric, while performing very poorly in any reasonable metric.

6.1. Manteuffel matrix and experimental stability methodology

The matrix generated by "central differences" introduced by Manteuffel [31] is employed in a series of tests designed for the Krylov–Schur eigenvalue solver based upon DCGS2–Arnoldi. This matrix is convenient for computing the forward error solution because the explicit computation of each eigenvalue is possible, thus the comparison against a converged eigenvalue is possible. For $m \times m$ block diagonal matrices M and N, where M and N have $k \times k$ sub-blocks such that $m = k^2$. M is positive definite and N is skew-symmetric. An explicit formulation of M and N is given in [31]. The Manteuffel matrix is expressed as the sum

$$A = \frac{1}{h^2} M + \frac{\beta}{2h} N \tag{11}$$

where the matrix blocks and k^2 eigenvalues are generated by

$$\lambda_{\ell,j} = 2\left[2 - \sqrt{1 - \left(\frac{\beta}{2}\right)^2}\left(\cos\left(\frac{\ell\pi}{L}\right) + \cos\left(\frac{j\pi}{L}\right)\right)\right] \tag{12}$$

For $\ell = 1, \ldots, k$ and $j = 1, \ldots, k$. As can be seen in (12), β is a scalar that governs the spectrum of the eigenspace. L is defined by the domain of the differential operator, $[0, L] \times [0, L]$ and $h = L/(k+1)$ is the discretization parameter. For the experiments within this section, $\beta = 0.5$ and $L = k + 1$ so that $h = 1$ ($\beta \leq 2$ implies all eigenvalues are real). For $k = 50$, relevant numerical metrics are summarized in Table 4. Here, V and W are the left and right eigenvectors. The Manteuffel matrix is employed to evaluate the convergence of Krylov–Schur.

The Arnoldi residual and error metrics employed herein are described in Hernández et al. [32]. Fig. 4 displays the loss of orthogonality $\| I_{j-1} - Q_{1:j-1}^T Q_{1:j-1} \|_F$, while Fig. 3 is a plot of the Arnoldi relative representation error, from (1)

$$RRE(j) = \frac{\left\| AQ_{1:j-2} - Q_{1:j-1} H_{1:j-1,1:j-2} \right\|_F}{\|A\|_F}$$

Each of the algorithms, except for the DCGS2-HRT presented in [21], achieve machine precision level relative error. Fig. 4 displays the loss of orthogonality for each scheme. It was noted earlier that CGS exhibits an $\mathcal{O}(\epsilon)\kappa^2(A)$ loss of orthogonality. The plot illustrates that DCGS2-HRT follows CGS while the other algorithms construct orthonormal columns to the level of machine precision.

The results from a Krylov–Schur eigenvalue experiment to evaluate the convergence properties of the different Arnoldi algorithms are plotted in Fig. 5. The solver relies upon the Schur decomposition of the Hessenberg matrix H_n generated in the Arnoldi expansion. To assess the convergence rates, the Arnoldi residual (13) is compared to the absolute error tolerance. The approximate eigenvector (or Ritz vector)

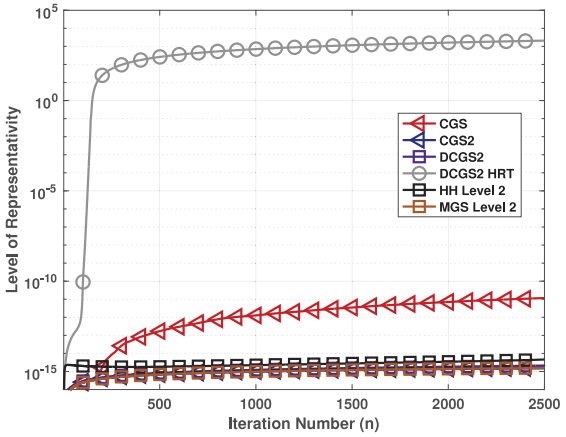

Fig. 3. Normed representation residual for the Arnoldi expansion on the convection–diffusion operator matrix ($\beta = 0.5, m = 2500$).

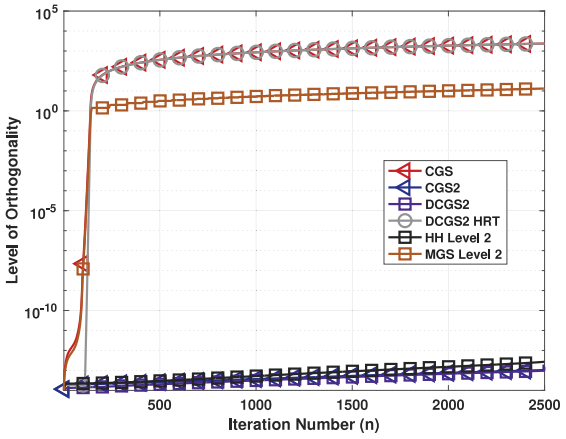

Fig. 4. Level of orthogonality of the Arnoldi expansion on the convection–diffusion operator matrix ($\beta = 0.5, m = 2500$).

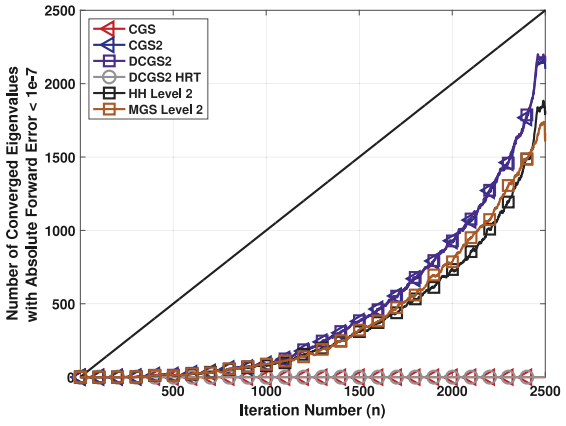

Fig. 5. Number of converged eigenvalues with absolute forward error less than 10^{-7}.

associated with the eigenvalue λ_i is defined by $z_i = V_n\, y_i$, where y_i is the corresponding eigenvector of H_n, see [21].

$$\| (A - \lambda_i I) z_i \|_2 = |H_{n+1,n}| \, | e_n^T y_i | < \text{tol} \tag{13}$$

where tol = 1e−7. If this threshold is satisfied, the iteration is considered to have found an invariant subspace and the associated diagonal element in the Schur triangular matrix $T_{l,l}$ is an eigenvalue. The representation error and loss of orthogonality can be easily computed. It is

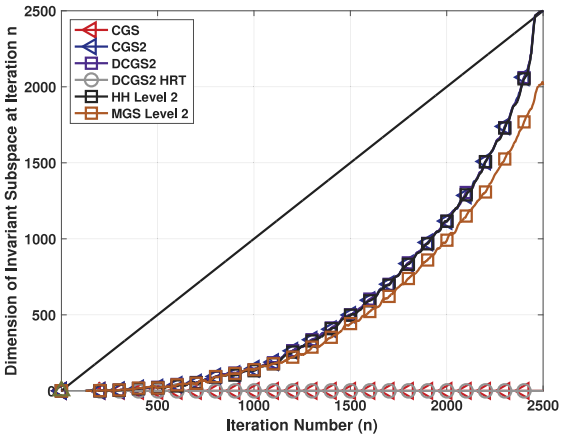

Fig. 6. Dimension of invariant subspace computed at iteration n.

important to note if these quantities are not close to machine precision, a converged invariant subspace has not been found. After the size of the invariant subspace has been found, the k^2 eigenvalues from the formula in (12) are computed and compared with the "converged" eigenvalues in the Schur triangular matrix T. In addition, rather than computing the same eigenvalue twice, the multiplicity is obtained to determine if the Krylov–Schur algorithm has computed the same eigenvalue, or unique eigenvalues in the decomposition. The exact multiplicity of any given eigenvalue was always found.

The plot in Fig. 5 displays the number of converged eigenvalues at each restart n of the Arnoldi algorithm according to the absolute forward error $|\lambda_i - T_{l,l}| < \text{tol}$, where $\text{tol} = 1e-7$. In practice, at each iteration n, for each l from 1 to n, the λ_i are scanned for each i closest to $T_{l,l}$, which has not been found for a previous l, and which satisfies the tolerance is selected. Our code will return an error flag if an iteration returns more eigenvalues than the expected multiplicity. The flag was never triggered during our experiments. At iteration n, at most n eigenvalues are found and these correspond to the solid black line. There are three reasons why the number of eigenvalues found is not n. First, the eigenvalues must be converged. At iteration n, in exact arithmetic, all eigenvalues would have been found. For iteration $n < m$, the number of eigenvalues found is between 0 and n. Second, the forward error is sensitive to the condition number of the eigenvalues. Some eigenvalues have condition number of the order 1e+10, (see Table 4), therefore, using $\varepsilon = 2.2e{-}16$ accuracy, our algorithms are not expected to find all eigenvalues at iteration $m = 2,500$. The maximum number of eigenvalues found is about 2100 with CGS2 and DCGS2 methods. This condition number problem is present at any restart n and is intrinsic when using a forward error criteria. Third, the Arnoldi factorization could have errors in fundamental quantities such that the loss of orthogonality and representation error are large. This may affect the number of eigenvalues found at restart n.

Fig. 6 displays, at each restart n, the size of the invariant subspace found. None of the methods can follow this line, but as the full Arnoldi expansion of the Manteuffel matrix is approached, any scheme that maintains orthogonality can continually find new eigenvalues, or new directions to search. Comparing both plots illustrates that in practice, when eigenvalues are not known, looking at the size of the invariant subspace can be a good metric. Note that between the two plots, there is a small gap for the error formula at a restart of $n = 2500$, where this gap is not present in the invariant subspace plot. The different Arnoldi variants cannot find all of the invariant subspaces, which is due to the condition number of the eigenvalues. Comparing the different QR factorization schemes and the invariant subspace found, although it loses orthogonality, Arnoldi with MGS can still find new search directions. Arnoldi based on Householder (HH), CGS2 and DCGS2, can find a subspace that spans the entire space, but for this matrix MGS still performs well and generates a subspace size close to 2000.

Table 5
Comparison of various orthogonalization schemes using an Arnoldi expansion of length $n = 75$ on 635 matrices from the Suite-Sparse collection.

Scheme	RRE $< 1e{-}7$	LOO $< 1e{-}7$	Invariant subspace
DCGS2	631	621	**9844**
DCGS2 HRT	435	463	7168
CGS2	**635**	622	9677
CGS	**635**	374	8370
HH Level 2	**635**	**635**	9783
MGS Level 2	634	519	9580

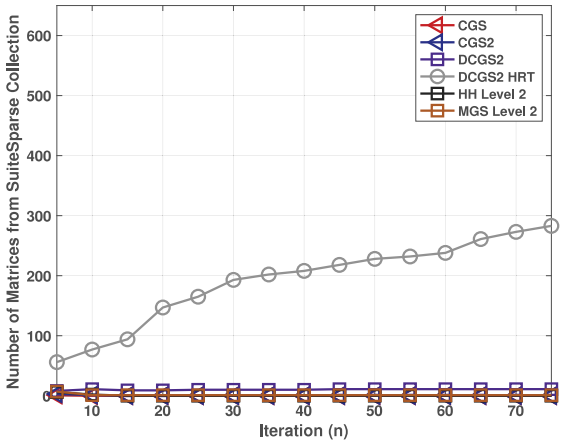

Fig. 7. Comparison of Arnoldi representation error for Suite-Sparse matrices at a given iteration n, $\|AQ - QH\|/\|A\| > 1e-7$.

6.2. Matrix market

The Arnoldi factorization algorithms are now compared for matrices gathered from the Suite-Sparse collection maintained by Tim Davis at Texas A&M University [33]. A total of 635 matrices were chosen by the following criteria: (1) number of nonzeros $<$ 500,000, (2) the matrix is `REAL`, (3) the matrix is `UNSYMMETRIC` and (4) the number of columns and rows $>$ 100. The Krylov basis is computed for each of the 635 matrices in the collection. The representation error and loss of orthogonality are computed for every 5 columns until 75 (making sure the dimension of any matrix is not exceeded). Meaning the restart in an Arnoldi expansion varies from $n = 5$ to $n = 75$ in increments of 5.

Figs. 7 and 8 display these metrics for each of the schemes. At each iteration the tolerance is set to 1e−7. If the representation error or loss of orthogonality is above this threshold the matrix is flagged. The y-axis represents the total number of matrices above the given threshold and the x-axis indicates the Krylov subspace dimension (restart m) employed by the Arnoldi expansion.

Fig. 8 clearly indicates that Krylov vectors generated using CGS and MGS lose orthogonality at different rates. It is observed that the DCGS2-HRT curve falls between these. For the Manteuffel matrix, DCGS2-HRT appears to perform more like CGS and lies somewhere in between. It is important to note, in Fig. 7, that DCGS2-HRT does not maintain a low representation error for the Arnoldi expansion. This is also apparent in Fig. 2.

With a restart of $n = 75$, these metrics are plotted in Table 5. The additional metric displayed is the size of the invariant subspace found, described in Section 6.1.

7. Parallel performance results

Parallel performance results are now presented for the Summit Supercomputer at Oak Ridge National Laboratory. Each node of Summit consists of two 22-core IBM Power 9 sockets and six NVIDIA Volta

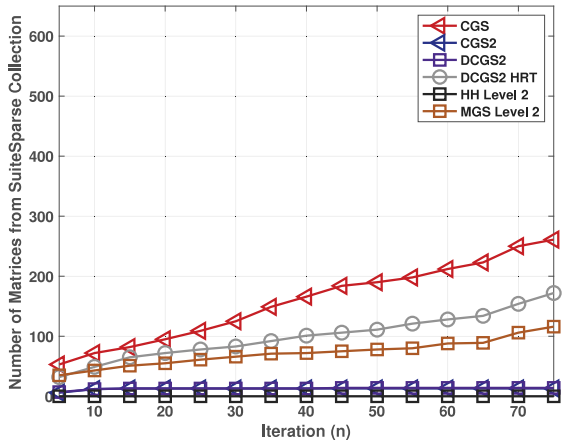

Fig. 8. Comparison of loss of Orthogonality at a given iteration n for Suite-Sparse matrices. $\|I - Q^T Q\| > 1e-7$.

100 GPUs. CGS2 and DCGS2 were implemented and tested using the Trilinos-Belos iterative solver framework [15,34]. Therefore, although NVIDIA V100 results are presented here, the implementation is portable to different hybrid node architectures with a single code base.

To summarize, DCGS2 achieves faster compute times than CGS2 for two reasons. First, the former employs either matrix–vector or matrix–matrix kernels, which provide greater potential for data reuse. For tall-and-skinny matrices, employed by `DGEMV` and `DGEMM`, computational time is often limited by data movement, and matrix–matrix type kernels often achieve faster execution rates. Therefore, DCGS2 is faster than CGS2, even on a single GPU. Second, on multiple GPUs, the low-synch algorithm decreases the number of global-reductions. Therefore, a greater speedup is achieved on a large number of GPUs. In this section, the execution rates on a single and multiple GPUs are compared.

Delayed schemes (DCGS2, and ICWY-MGS with lagged normalization) have a start-up phase, they are in some sense running one iteration behind non-delayed schemes (CGS2). For a fair comparison of performance between delayed schemes (DCGS2, and ICWY-MGS with lagged normalization) and non-delayed schemes (CGS2), a cleanup step is applied for the last iteration of the delayed schemes, so that the Arnoldi factorization returned by the non-delayed and the delayed schemes are the same.

7.1. Single GPU performance

Table 6 provides the execution rates in GigaFlops/sec of the main computational kernels on a single GPU, with an increasing number of rows or columns, as reported in and columns respectively. Within the plot,

- `MvTransMv` computes the dot-products, e.g., `DGEMV` to compute $S_{1:j-1,j} = Q^T_{1:j-1} a_j$ in CGS2 or `DGEMM` to compute $[Q_{1:j-2}, w_{j-1}]^T$ $[w_{j-1}, a_j]$ in DCGS2.
- `MvTimesMatAddMv` updates the vectors by applying the projection, e.g., `DGEMV` to compute $w_j = q_j - Q_{1:j-1} S_{1:j-1,j}$ in DCGS2 or `DGEMM` to compute

$$[u_{j-1}, w_j] = [w_{j-1}, a_j] - Q_{1:j-2} [C_{1:j-2,j-1}, S_{1:j-1,j}]$$

- `MvDot` computes `DDOT` product of two vectors, and is used to compute the normalization factor $\alpha_{j-1} = \|u_j\|_2$.

Memory bandwidth utilization is a predictor of performance. For example, at the jth iteration of CGS2, `MvTransMV` reads the $m \times (j-1)$ matrix $Q_{1:j-1}$ and the input vector a_j of length m, then writes the result back to the output vector $S_{1:j-1,j}$, while performing $(2m-1)\times(j-1)$ flops

Table 6
Execution rate (GigaFlops/s) of BLAS kernels (1 node, 1 GPU).

Scheme	Number of rows, n, in millions				
	1	2	4	8	16
(a) Fixed number of columns $n = 50$.					
DCGS2					
MVTimes GF/s	300.3	319.1	302.3	320.3	331.4
MVTrans GF/s	201.4	211.8	191.4	150.2	126.8
Total GF/s	215.1	232.3	218.2	193.3	174.8
CGS2					
MVTimes GF/s	132.0	136.4	153.4	163.8	169.5
MVTrans GF/s	128.5	141.0	146.4	135.3	122.2
Total GF/s	126.4	135.4	146.7	145.5	139.6

Scheme	Number of columns, n				
	100	120	140	160	180
(b) Fixed number of rows $m = 5e+6$.					
DCGS2					
MVTimes GF/s	353.7	362.8	369.8	375.6	379.7
MVTrans GF/s	169.2	164.3	160.3	158.7	155.6
Total GF/s	221.2	219.8	218.4	218.5	216.6
CGS2					
MVTimes GF/s	182.1	186.9	190.7	193.1	195.7
MVTrans GF/s	152.0	153.6	153.4	153.3	153.2
Total GF/s	163.8	167.1	168.6	169.7	170.8

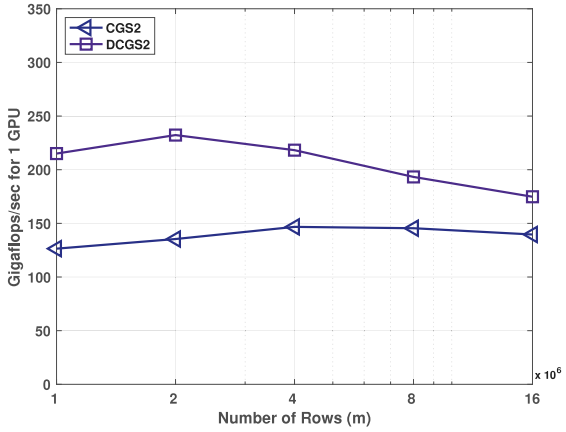

Fig. 9. Execution rate (Gigaflops/sec, 1 GPU). $n = 50$ columns, varying number of rows m.

with two flops per read, assuming the vectors remain in caches, or two flops per two reads and one write with a read–write of vector elements for each multiply-add. Thus, on the V100 with a memory bandwidth of 800 GB/s, 200 GigaFlops/sec is expected from this kernel in double precision. Figs. 9 and 10 display kernel compute times on one Summit node using one GPU as the number of rows or columns is varied.

Note that DCGS2 combines two `MvTransMv` calls with a single input and output vector into a call to `MvTimesMatAddMv` with two input and output vectors. This can double the potential peak speed (i.e. the $m \times (j-1)$ matrix is read only once to perform four flops per read). Table 6 indicates that for a large number of rows or columns, that DCGS2 increases the execution rate by up to 1.7 and 1.4×, respectively.

7.2. Strong-scaling performance

The speedups obtained by DCGS2 for the two main kernels are presented in Tables 8–10, while Figs. 11 and 12 and displays the GigaFlops/sec execution rate achieved by the kernels on 30 nodes, using 6 GPUs per node. Figs. 13 and 14 represent a strong-scaling study and display the time to solution while varying the number of GPUs for

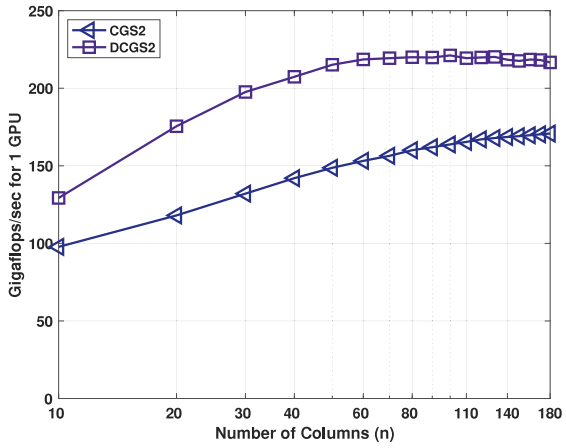

Fig. 10. Execution rate (Gigaflops/sec, 1 GPU). m = 25e+6 rows, varying number of columns n.

Table 7
Execution rate (GigaFlops/s) of BLAS kernels (30 nodes, 6 GPUs per node).

Scheme	Number of rows, n, in millions				
	128	256	512	1024	2048
(a) Fixed number of columns n = 50. GigaFlops/s					
DCGS2					
MVTimes GF/s	194.2	248.9	285.9	312.5	324.2
MVTrans GF/s	129.4	170.4	179.9	154.1	128.6
Total GF/s	124.3	168.9	195.4	189.4	173.2
CGS2					
MVTimes GF/s	99.6	126.6	145.1	158.9	167.1
MVTrans GF/s	73.9	100.8	124.9	128.4	119.3
Total GF/s	42.8	98.2	123.7	134.3	133.9

Scheme	Number of columns, n				
	100	120	140	160	180
(b) Fixed number of rows n = 25e+6. GigaFlops/s					
DCGS2					
MVTimes GF/s	121.6	135.5	151.2	159.7	168.9
MVTrans GF/s	61.1	76.9	89.1	89.1	107.6
Total GF/s	61.4	74.4	84.4	90.1	103.4
CGS2					
MVTimes GF/s	122.2	95.2	101.3	99.3	90.4
MVTrans GF/s	38.1	39.4	49.6	56.5	52.6
Total GF/s	36.4	38.9	47.7	54.5	53.4

a fixed matrix size. One run or trial for each node count was employed for the algorithms in order to collect the performance data on Summit.

- Table 8 displays the speedup (ratio of DCGS2 to CGS2 compute time) for `MvTransMv`. Because DCGS2 employs fewer global reductions, as the number of GPUs increases, the speedup obtained by `MvTransMv` increases, reaching up to 2.20× faster times on 192 GPUs.
- Table 9 displays the speedup for the `MvTimesMatAddMv` kernel. DCGS2 merges two `MvTransMv` calls into one `MvTimesMatAddMv` and achieves 2× speedup on a single GPU. With more GPUs, the number of local rows and speedup decrease. However, the compute time is dominated by the `MvTimesMatAddMv` kernel.

Table 10 displays the speedup obtained by DCGS2, when varying the number of rows. By combining matrix–vector products with global reductions, the speedup obtained by DCGS2 in some instances was significant, up to 3.6× faster.

Figs. 13 and 14 display the time to solution for the GMRES linear solvers. The latter achieves improved strong scaling due to the merged `MvTimesMatAddMv` kernel.

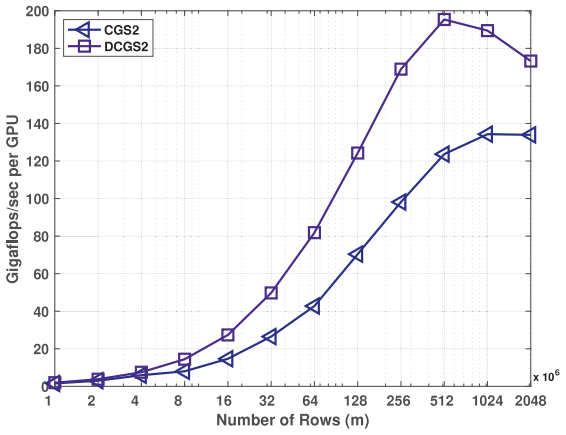

Fig. 11. Execution rate per node (30 nodes, 6 GPUs per node), varying number of rows m.

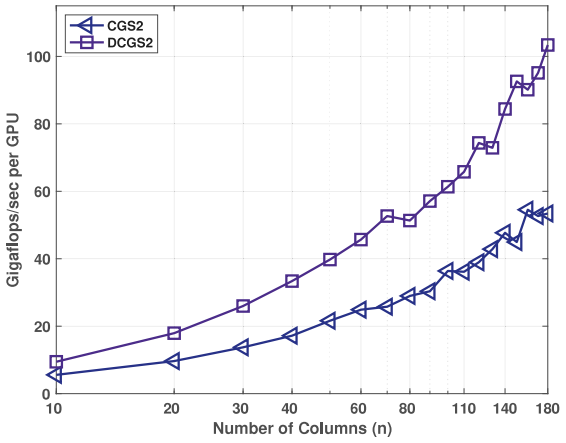

Fig. 12. Execution rate per node (30 nodes, 6 GPUs per node), varying number of columns n.

Table 8
Speedup of DCGS2 over CGS2 for MvTransMv and MvDot.

# GPUs	Number of rows, n, in millions				
	1	5	10	25	50
6	1.7	1.6	1.6	1.3	1.1
12	1.9	1.8	1.7	1.5	1.3
24	2.1	1.7	1.7	1.6	1.5
48	2.1	1.9	1.9	1.8	1.6
96	2.0	2.1	4.0	1.8	1.8
192	2.2	2.1	2.3	2.1	2.2

Table 9
Speedup of DCGS2 over CGS2 for MvTimesMatAddMv.

# GPUs	Number of rows, n, in millions				
	1	5	10	25	50
6	0.8	2.0	2.0	1.9	2.0
12	1.9	2.0	1.9	2.0	1.9
24	1.3	0.9	1.2	2.0	2.0
48	1.1	0.9	0.9	1.9	2.0
96	1.1	0.8	5.2	1.1	1.3
192	0.9	0.7	0.7	0.9	1.3

Table 7 displays the GigaFlops/sec execution rates obtained by CGS2 and DCGS2, along with the BLAS kernels for a fixed matrix size (m = 25e+6 and n = 50). The `MvTransMv` operation requires a global reduce, while the `DGEMM` operations do not require communication. DCGS2 always outperforms CGS2 in these runs. `MvTimesMatAddMv`

Table 10
Overall speedup of DCGS2 over CGS2.

# GPUs	Number of rows, n, in millions				
	1	5	10	25	50
6	1.1	1.6	1.7	1.5	1.3
12	1.1	1.8	1.7	1.6	1.5
24	1.1	1.5	1.8	1.7	1.6
48	1.2	1.1	1.8	1.8	1.7
96	1.2	1.3	**4.0**	1.8	1.8
192	1.4	1.3	1.7	1.9	**2.1**

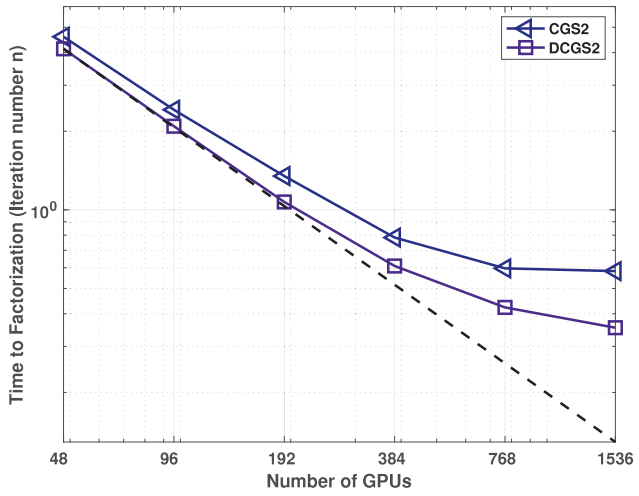

Fig. 13. GMRES time to solution on 3D Laplace matrix. $m = 750^3$, $n = 100$ (256 nodes, 6 GPUs per node).

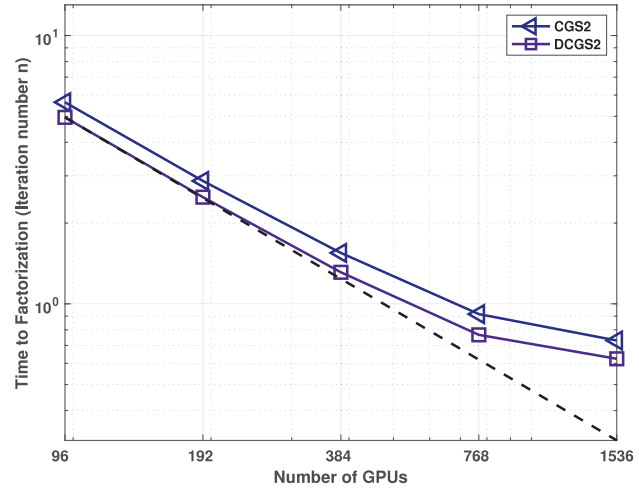

Fig. 14. GMRES time to solution on 3D Laplace matrix. $m = 1000^3$, $n = 100$ (256 nodes, 6 GPUs per node).

perform similarly for both schemes on 96 and 192 nodes. CGS2 exhibits an increase in speed that matches DCGS2. For a large number of rows or columns, DCGS2 obtains about 66 GigaFlops/sec per node at 192 nodes or about 6% of the single GPU sustained execution rate of 200 GigaFlops/sec.

Figs. 13 and 14 are strong scaling experiments for the 3D Laplace equation with dimension $m = 750^3$ and $m = 1000^3$. The simulations employ from 8 to 256 Summit compute nodes. The GMRES solver is run in five trials for each of the node counts using 6 GPUs per node, in non-dedicated runs on Summit. The average compute times are plotted in these Figures. A fixed number of $n = 100$ iterations are

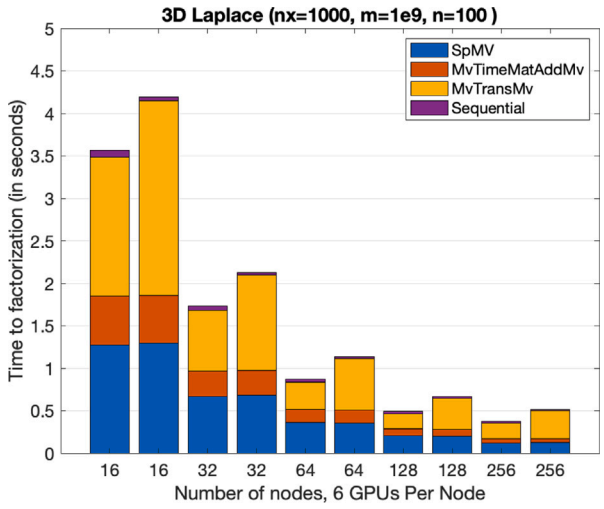

Fig. 15. GMRES Gram–Schmidt time. $m = 1000^3$, $n = 100$ (256 nodes, 6 GPUs per node). Left DCGS2. Right CGS2.

performed, without a restart, and a preconditioner is not applied. The DCGS2-GMRES yields lower run times and exhibits better strong scaling characteristics (see Fig. 15).

8. Conclusion

For distributed-memory computation, two passes of classical Gram–Schmidt (CGS2) was the method of choice for Krylov solvers requiring machine precision level representation errors and loss of orthogonality. However, the algorithm requires three global reductions for each column of the QR decomposition computed and thus the strong-scaling behavior can deviate substantially from linear as the number of MPI ranks increases on Exascale class supercomputers such as the ORNL Summit. In this paper, a new variant of CGS2 that requires only one global reduction per iteration was applied to the Arnoldi algorithm. Our numerical results have demonstrated that DCGS2 obtains the same loss of orthogonality and representation error as CGS2, while our strong-scaling studies on the Summit supercomputer demonstrate that DCGS2 obtains a speedup of 2× faster compute time on a single GPU, and an even larger speedup on an increasing number of GPUs, reaching 2.2× lower execution times on 192 GPUs. The impact of DCGS2 on the strong scaling of Krylov linear system solvers is currently being explored, and a block variant is also being implemented following the review article of Carson et al. [20].

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgment

This research was supported by the Exascale Computing Project, USA (17-SC-20-SC), a collaborative effort of the U.S. Department of Energy Office of Science and the National Nuclear Security Administration. The National Renewable Energy Laboratory is operated by Alliance for Sustainable Energy, LLC, for the U.S. Department of Energy (DOE) under Contract No. DE-AC36-08GO28308. Sandia National Laboratories is a multimission laboratory managed and operated by National Technology & Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International Inc., for the U.S. Department of Energy National Nuclear Security Administration under contract DE-NA0003525.

A portion of this research used resources of the Oak Ridge Leadership Computing Facility, that is a DOE Office of Science User Facility supported under Contract DE-AC05-00OR22725 and using computational resources sponsored by the Department of Energy's Office of Energy Efficiency and Renewable Energy and located at the National Renewable Energy Laboratory.

Julien Langou was supported by NSF, USA award #2004850.

References

[1] Y. Saad, M.H. Schultz, GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems, SIAM J. Sci. Stat. Comput. 7 (3) (1986) 856–869.

[2] G.W. Stewart, A Krylov–Schur algorithm for large eigenproblems, SIAM J. Matrix Anal. Appl. 23 (3) (2001) 601–614.

[3] L. Giraud, J. Langou, M. Rozložník, On the loss of orthogonality in the Gram-Schmidt orthognalization process, Comput. Math. Appl. 50 (2005) 1069–1075, http://dx.doi.org/10.1016/j.camwa.2005.08.009.

[4] C.C. Paige, The effects of loss of orthogonality on large scale numerical computations, in: Proceedings of the International Conference on Computational Science and its Applications, Springer-Verlag, 2018, pp. 429–439.

[5] C.C. Paige, M. Rozložník, Z. Strakoš, Modified gram-Schmidt (MGS), least squares, and backward stability of MGS-GMRES, SIAM J. Matrix Anal. Appl. 28 (1) (2006) 264–284, http://dx.doi.org/10.1137/050630416.

[6] A. Bienz, W. Gropp, L. Olson, Node-aware improvements to AllReduce, in: Proceedings of the 2019 IEEE/ACM Workshop on Exascale MPI, ExaMPI, Association for Computing Machinery, 2019, pp. 1–10.

[7] J.J. Dongarra, F.G. Gustavson, A. Karp, Implementing linear algebra algorithms for dense matrices on a vector pipeline machine, SIAM Rev. 26 (1) (1984) 91–112.

[8] J. Ortega, C. Romine, The ijk forms of factorization methods II. Parallel systems, Parallel Comput. 7 (2) (1988) 149–162.

[9] J. Dongarra, S. Hammarling, D. Walker, Key concepts for parallel out-of-core LU factorization, Comput. Math. Appl. 35 (7) (1998) 13–31, Advanced Computing on Intel Architectures.

[10] L. Giraud, J. Langou, M. Rozložník, J. van den Eshof, Rounding error analysis of the classical Gram-Schmidt orthogonalization process, Numer. Math. 101 (1) (2005) 87–100.

[11] A. Smoktunowicz, J.L. Barlow, J. Langou, A note on the error analysis of classical Gram-Schmidt, Numer. Math. 105 (2) (2006) 299–313.

[12] B.N. Parlett, The Symmetric Eigenvalue Problem, Society for Industrial and Applied Mathematics, 1998, http://dx.doi.org/10.1137/1.9781611971163.

[13] S.J. Leon, Å. Björck, W. Gander, Gram-Schmidt orthogonalization: 100 years and more, Numer. Linear Algebra Appl. 20 (3) (2013) 492–532.

[14] V. Frayssé, L. Giraud, H.K. Aroussi, On the influence of the orthogonalization scheme on the parallel performance of GMRES, Tech. Rep. TR-PA-98-07, CERFACS, 1998.

[15] E. Bavier, M. Hoemmen, S. Rajamanickam, H. Thornquist, Amesos2 and belos: Direct and iterative solvers for large sparse linear systems, Sci. Program. 20 (3) (2012) 241–255.

[16] S. Balay, W.D. Gropp, L.C. McInnes, B.F. Smith, Efficient management of parallelism in object oriented numerical software libraries, in: E. Arge, A.M. Bruaset, H.P. Langtangen (Eds.), Modern Software Tools in Scientific Computing, Birkhäuser Press, 1997, pp. 163–202.

[17] R.D. Falgout, J.E. Jones, U.M. Yang, The design and implementation of hypre, a library of parallel high performance preconditioners, in: A.M. Bruaset, A. Tveito (Eds.), Numerical Solution of Partial Differential Equations on Parallel Computers, Springer Berlin Heidelberg, Berlin, Heidelberg, 2006, pp. 267–294.

[18] H. Anzt, T. Cojean, G. Flegar, F. Göbel, T. Grützmacher, P. Nayak, T. Ribizel, Y.M. Tsai, E.S. Quintana-Ortí, Ginkgo: A Modern Linear Operator Algebra Framework for High Performance Computing, Tech. Rep. 2006.16852, arXiv, 2020, p. pp.30.

[19] S.K. Kim, A.T. Chronopoulos, An efficient parallel algorithm for extreme eigenvalues of sparse nonsymmetric matrices, Int. J. Supercomput. Appl. 6 (1) (1992) 98–111.

[20] E. Carson, K. Lund, M. Rozložník, S. Thomas, An overview of block Gram-Schmidt methods and their stability properties, 2020, arXiv:2010.12058.

[21] V. Hernández, J. Román, A. Tomas, Parallel arnoldi eigensolvers with enhanced scalability via global communications rearrangement, Parallel Comput. 33 (2007) 521–540.

[22] K. Swirydowicz, J. Langou, S. Ananthan, U. Yang, S. Thomas, Low synchronization Gram-Schmidt and generalized minimal residual algorithms, Numer. Linear Algebra Appl. 28 (2020) 1–20.

[23] C. Puglisi, Modification of the householder method based on the compact WY representation, SIAM J. Sci. Stat. Comput. 13 (3) (1992) 723–726.

[24] J. Malard, C.C. Paige, Efficiency and scalability of two parallel QR factorization algorithms, in: Proceedings of IEEE Scalable High Performance Computing Conference, 1994, pp. 615–622.

[25] R. Schreiber, C.V. Loan, A storage-efficient WY representation for products of householder transformations, SIAM J. Sci. Stat. Comput. 10 (1) (1989) 53–57.

[26] J.L. Barlow, Block modified Gram–Schmidt algorithms and their analysis, SIAM J. Matrix Anal. Appl. 40 (4) (2019) 1257–1290.

[27] I. Yamazaki, S. Thomas, M. Hoemmen, E.G. Boman, K. Świrydowicz, J.J. Elliott, Low-synchronization orthogonalization schemes for s-step and pipelined Krylov solvers in Trilinos, in: Proceedings of the 2020 SIAM Conference on Parallel Processing for Scientific Computing, SIAM, 2020, pp. 118–128, http://dx.doi.org/10.1137/1.9781611976137.11.

[28] E. Carson, K. Lund, M. Rozložník, The stability of block variants of classical gram-Schmidt, Tech. Rep. 6–2021, Czech Academy of Sciences, 2021.

[29] Å. Björck, Numerics of Gram-Schmidt orthogonalization, Linear Algebra Appl. 197 (1994) 297–316.

[30] A. Björck, Solving least squares problems by Gram–Schmidt orthogonalization, BIT 7 (1967) 1–21.

[31] T.A. Manteuffel, Adaptive procedure for estimating parameters for the nonsymmetric Tchebychev iteration, Numer. Math. 31 (2) (1978) 183–208.

[32] V. Hernández, J. Román, A. Tomas, V. Vidal, Krylov-Schur Methods in SLEPc, Tech. Rep. SLEPc Technical Report STR-7, University of Valencia, 2007.

[33] T.A. Davis, Y. Hu, The university of florida sparse matrix collection, ACM Trans. Math. Software 38 (1) (2011) http://dx.doi.org/10.1145/2049662.2049663.

[34] M. Heroux, et al., An overview of the trilinos project, ACM Trans. Math. Software (2005).