Parallel Computing 65 (2017) 1–20

Contents lists available at ScienceDirect

Parallel Computing

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

The communication-hiding pipelined BiCGstab method for the parallel solution of large unsymmetric linear systems

S. Cools[*], W. Vanroose

Applied Mathematics Research Group, Department of Mathematics and Computer Science, University of Antwerp, Middelheimlaan 1, B-2020 Antwerp, Belgium

ARTICLE INFO

Article history:
Received 7 November 2016
Revised 4 April 2017
Accepted 13 April 2017
Available online 14 April 2017

Keywords:
Parallellization
Global communication
Latency hiding
Krylov subspace methods
Bi-Conjugate Gradient Stabilized
Residual replacement

ABSTRACT

A High Performance Computing alternative to traditional Krylov subspace methods, pipelined Krylov subspace solvers offer better scalability in the strong scaling limit compared to standard Krylov subspace methods for large and sparse linear systems. The typical synchronization bottleneck is mitigated by overlapping time-consuming global communication phases with local computations in the algorithm. This paper describes a general framework for deriving the pipelined variant of any Krylov subspace algorithm. The proposed framework was implicitly used to derive the pipelined Conjugate Gradient (p-CG) method in *Hiding global synchronization latency in the preconditioned Conjugate Gradient algorithm* by P. Ghysels and W. Vanroose, Parallel Computing, 40(7):224–238, 2014. The pipelining framework is subsequently illustrated by formulating a pipelined version of the BiCGStab method for the solution of large unsymmetric linear systems on parallel hardware. A residual replacement strategy is proposed to account for the possible loss of attainable accuracy and robustness by the pipelined BiCGStab method. It is shown that the pipelined algorithm improves scalability on distributed memory machines, leading to significant speedups compared to standard preconditioned BiCGStab.

1. Introduction

At present, Krylov subspace methods fulfill the role of standard linear algebra solution methods in many high-performance computing (HPC) applications where large and sparse linear systems need to be solved. The Conjugate Gradient (CG) method [29] can be considered as the earliest member of this well-known class of iterative solvers. However, the CG method is restricted to the solution of symmetric and positive definite (SPD) systems. Several variants of the CG method have been proposed that allow for the solution of more general classes of unsymmetric and indefinite linear systems. These include e.g. the Bi-Conjugate Gradient (BiCG) method [23], the Conjugate Gradient Squared (CGS) method [38], and the widely used BiCGStab method which was introduced as a smoother converging version of the aforementioned methods by H.A. Van der Vorst in 1992 [41].

Motivated by the vast numbers of processors in contemporary petascale and future exascale HPC hardware, research on the scalability of Krylov subspace methods on massively parallel hardware has become increasingly prominent over the last years. The practical importance of Krylov subspace methods for solving sparse linear systems is reflected in the High Performance Conjugate Gradient (HPCG) benchmark used for ranking HPC systems introduced in 2013 [16,17]. This new

[*] Corresponding author.
E-mail addresses: siegfried.cools@uantwerpen.be (S. Cools), wim.vanroose@uantwerpen.be (W. Vanroose).

http://dx.doi.org/10.1016/j.parco.2017.04.005
0167-8191/© 2017 Elsevier B.V. All rights reserved.

ranking is based on sparse matrix-vector computations and data access patterns, rather than the dense matrix algebra used in the traditional High Performance LINPACK (HPL) benchmark.

Krylov subspace algorithms are build from three basic components, namely: *sparse matrix-vector products* (SPMVS) Ax, *dot-products* (DOT-PRODS) or inner products (x, y) between to vectors, and AXPY operations $x \leftarrow ax + y$ that combine *scalar multiplication* ax and *vector addition* $x + y$. Single-node Krylov solver performance is dominated by the computation of the SPMV, which has the highest floating-point operation (FLOP) count of the algorithmic components. However, the main bottleneck for efficient parallelization is typically not the easy-to-parallelize SPMV application, but the dot-product computation that requires a global reduction over all processors. These global communications are commonly performed through a 2-by-2 reduction tree, which requires $\mathcal{O}(\log_2(P))$ time, where P is the number of processors in the machine, see [24]. The resulting global synchronization phases cause severe communication overhead on parallel machines. Hence, dot-product computations are communication-bound, and are thus effectively the most time-consuming component of Krylov subspace algorithms for large scale linear systems.

A significant amount of research has been devoted to the reduction and elimination of the synchronization bottlenecks in Krylov subspace methods over the last decades. The idea of reducing the number of global communication phases and hiding the communication latency in Krylov subspace methods on parallel computer architectures was first presented by Chronopoulos and Gear in [8] and further elaborated in a variety of papers, among which [2,12,20]. In 2002, Yang et al. [43–45] proposed the so-called *improved* versions of the standard BiCG, BiCGStab and CGS algorithms, which reduce the number of global reductions to only one per iteration. Furthermore, the *s*-step formulation of Krylov subspace methods [3–5,7,9,10] allows to reduce the total number of global synchronization points by a factor of *s*.

In addition to avoiding communication by reducing the number of global synchronization points, research on hiding global communication latency by overlapping communication and computations was performed by, among others, Demmel et al. [15], De Sturler et al. [13] and Ghysels et al. [24,25]. The latter introduces the class of pipelined Krylov subspace methods, which in addition to *removing* costly global synchronization points (communication-avoiding), aim at *overlapping* the remaining global communication phases by the SPMV and the preconditioner application (communication-hiding). In this way idle core time is minimized by performing useful computations simultaneously to the time-consuming global communication phases, cf [18,34].

The current paper aims to extend the work on pipelined Krylov subspace methods. The article is structured as follows. Section 2 introduces a high-level framework for the derivation of a pipelined method starting from the original Krylov subspace algorithm. Section 3 illustrates the use of this framework by deriving the new pipelined BiCGStab (p-BiCGStab) method. The inclusion of a preconditioner, which is conveniently simple for pipelined Krylov subspace methods, is also discussed here. Experimental results with the p-BiCGStab method on a wide range of problems are proposed in Section 4 to validate the mathematical equivalence between the traditional and pipelined methods. A residual replacement strategy is proposed to improve the numerical accuracy and robustness of the pipelined BiCGStab solver. Section 5 compares the parallel performance of the traditional and pipelined BiCGStab methods on a moderately-sized cluster and comments on the numerical accuracy of the pipelined BiCGStab solution. Finally, conclusions are formulated in Section 6.

2. General framework for deriving a pipelined Krylov subspace algorithm

This section provides the outline of a generic work plan for deriving the pipelined version of any traditional Krylov subspace method. The framework is based on the two main properties of the pipelining technique: *avoiding communication*, achieved by reducing the number of global reductions where possible, and *hiding communication* by overlapping local computational work (AXPYS, SPMVS) with the global communication required for the composition of dot-products. The framework below was implicitly used for the derivation of existing pipelined algorithms, such as the pipelined p-CG and p-CR methods in [25], as well as the l^1-GMRES algorithm proposed in [24]. The different steps in the theoretical framework are illustrated by its application to the pipelined CG method.

2.1. Step 1: Avoiding communication

The primary goal in the first step is the merging of global reduction phases (DOT-PRODS) to reduce the number of global synchronization points, and hence, the overall time spent in communication. Algorithm 1 shows a schematic illustration of the original Krylov subspace algorithm (left) and indicates the operations that are performed in Step 1 (right) to obtain the communication-avoiding (or CA, for short) Krylov subspace method. The latter is presented schematically in Algorithm 3 (left). The following sequence of steps (a) to (d) may be repeated any number of times to merge multiple global reduction phases, if required.

1(a) Identify two global reduction phases that are candidates for merging. The global reduction phase that is located before the intermediate SPMV computation (S1) is denoted (R1) and the subsequent reduction phase that is executed after the SPMV is called (R2).
E.g.: CG: reduction (R1) is found on line 10 in Algorithm 2, while (R2) is on line 5. The intermediate SPMV is located on line 4.

1(b) Introduce a recurrence for the SPMV computation (S1) that is computed in between the global reduction phases (R1) and (R2). To achieve this, substitute the SPMV vector by its respective recurrence, which is computed before (S1).

Algorithm 1 Standard Krylov subspace method **Step 1: Avoiding communication**.

```
1:  function KRYLOV($A$, $b$, $x_0$)
2:      ...                                                          1(a) identification
3:      for $i = 0, 1, 2, \ldots$ do
4:          ...                                                      1(d) new SPMVS        (S2)
5:          begin reduction DOT-PRODS $(x, y)$ end reduction                                (R1)
6:          ...
7:          computation SPMVS $Ax$                                   1(b) recurrences      (S1)
8:          ...
9:          begin reduction DOT-PRODS $(x, y)$ end reduction         1(c) reformulation    (R2)
10:         ...
11:     end for
12: end function
```

Algorithm 2 Standard CG (preconditioned).

```
1:  function PREC-CG($A$, $M^{-1}$, $b$, $x_0$)
2:      $r_0 := b - Ax_0$; $u_0 := M^{-1}r_0$; $p_0 := u_0$
3:      for $i = 0, 1, 2, \ldots$ do
4:          computation $s_i := Ap_i$
5:          begin reduction $(s_i, p_i)$ end reduction
6:          $\alpha_i := (r_i, u_i)/(s_i, p_i)$
7:          $x_{i+1} := x_i + \alpha_i p_i$
8:          $r_{i+1} := r_i - \alpha_i s_i$
9:          computation $u_{i+1} := M^{-1}r_{i+1}$
10:         begin reduction $(r_{i+1}, u_{i+1})$ end reduction
11:         $\beta_{i+1} := (r_{i+1}, u_{i+1})/(r_i, u_i)$
12:         $p_{i+1} := u_{i+1} + \beta_{i+1} p_i$
13:     end for
14: end function
```

Algorithm 3 CA-Krylov subspace method **Step 2: Hiding communication**.

```
1:  function CA-KRYLOV($A$, $b$, $x_0$)
2:      ...                                                          2(a) identification
3:      for $i = 0, 1, 2, \ldots$ do
4:          ...
5:          computation SPMVS $Ax$                                   2(b) recurrences      (S2)
6:          ...
7:          begin reduction DOT-PRODS $(x, y)$ end reduction         2(c) reformulation    (R1)
8:          ...
9:          ...                                                      2(d) new SPMVS        (S3)
10:     end for
11: end function
```

Define new SPMV variables (S2) where needed and compute these SPMVS directly below their corresponding recursive vector definitions.
E.g.: CG: the SPMV (S1) on line 4 of Algorithm 2 is replaced by $s_i = Au_i + \beta_i s_{i-1}$. The new SPMV variable $w_i = Au_i$ (S2) is introduced directly below the definition of u_{i+1} on line 9.

1(c) Use the new recurrence defined in (b) to reformulate the dot-products of the (R2) reduction phase independently of any intermediate variables that are computed between (R1) and (R2).
E.g.: CG: occurrences of s_i and p_i in (R2) on line 5 of Algorithm 2 are replaced by their respective recurrences (see lines 4 and 12). The expression for α_i is reformulated accordingly.

1(d) Move the (R2) global reduction phase upward to merge it with the (R1) global reduction into a single global communication phase.
E.g.: CG: the reduction (R2) on line 5 is independent of the intermediate variables s_i and p_{i+1}. It can thus be moved upward (to the previous iteration) to compute α_{i+1} right below β_{i+1}.

E.g.: CG-CG: Step 1 results in the communication avoiding Conjugate Gradient method by Chronopoulos and Gear [8], denoted as CG-CG for short, outlined in Algorithm 4.

Algorithm 4 Chronopoulos & Gear CG (preconditioned).

```
1:  function PREC-CG-CG($A$, $M^{-1}$, $b$, $x_0$)
2:      $r_0 := b - Ax_0$; $u_0 := M^{-1}r_0$; $w_0 := Au_0$
3:      $\alpha_0 := (r_0, u_0)/(w_0, u_0)$; $\beta_0 := 0$; $\gamma_0 := (r_0, u_0)$
4:      for $i = 0, 1, 2, \dots$ do
5:          $p_i := u_i + \beta_i p_{i-1}$
6:          $s_i := w_i + \beta_i s_{i-1}$
7:          $x_{i+1} := x_i + \alpha_i p_i$
8:          $r_{i+1} := r_i - \alpha_i s_i$
9:          computation $u_{i+1} := M^{-1}r_{i+1}$
10:         computation $w_{i+1} := Au_{i+1}$
11:         begin reduction $\gamma_{i+1} := (r_{i+1}, u_{i+1})$; $\delta := (w_{i+1}, u_{i+1})$ end reduction
12:         $\beta_{i+1} := \gamma_{i+1}/\gamma_i$
13:         $\alpha_{i+1} := (\delta/\gamma_{i+1} - \beta_{i+1}/\alpha_i)^{-1}$
14:     end for
15: end function
```

For preconditioned Krylov subspace methods the SPMV phases (S1) and (S2) may also include the (right) preconditioner application, which is typically computed right before the SPMV. This is exemplified by the CG algorithms, and is illustrated for the BiCGStab method in Section 3.6.

2.2. Step 2: Hiding communication

The aim of the second step is the simultaneous execution (overlapping) of the global reduction communication phases with independent local SPMV computations to hide communication time behind useful computations and hence minimize worker idling. Step 2 is illustrated in Algorithm 3 and exemplified by the CG-CG method in Algorithm 4. Starting from the communication-avoiding Krylov subspace method that is represented schematically by Algorithm 3 (left), the operations (a)-(d) in Step 2 required to obtain the pipelined Krylov subspace method are indicated in Algorithm 3 (right). The following reformulations ultimately leads to Algorithms 5–6.

Algorithm 5 Pipelined Krylov subspace method **Final pipelined algorithm.**

```
1:  function PIPE-KRYLOV($A$, $b$, $x_0$)
2:      ...
3:      for $i = 0, 1, 2, \dots$ do
4:          ...
5:          begin reduction DOT-PRODS $(x, y)$                          (R1)
6:          computation SPMVS $Ax$                  [overlapping]       (S3)
7:          end reduction
8:          ...
9:      end for
10: end function
```

2(a) Following Step 1, each SPMV phase (S2) is followed by a corresponding global reduction phase (R1). These SPMV/DOT-PROD pairs are possibly separated by intermediate AXPY operations.
E.g.: CG-CG: in Algorithm 4 the SPMV (S2) on line 9–10 is directly followed by the global reduction phase (R1) required for the DOT-PRODS on line 11.

2(b) Introduce a recurrence for the SPMV vector in (S2) through substitution by its recursive characterization, which is computed before (S2). Define new SPMV variables where needed (S3).
E.g.: CG-CG: the SPMV (S2) on line 10 in Algorithm 4 is replaced by the recurrence $w_{i+1} = w_i - \alpha_i AM^{-1}s_i$. Here the SPMV variable is defined as $z_i = AM^{-1}s_i$. However, using the recurrence for s_i, z_i can in turn be computed recursively as $z_i = AM^{-1}w_i + \beta_i z_{i-1}$. The new SPMV $n_i = AM^{-1}w_i$ (S3) is introduced below the recursive definition of w_{i+1} on line 10.

2(c) Use the new recursive definition obtained in (b) to reformulate the dot-products of the (R1) reduction phase independently of the new SPMV variables that are computed in (S3).
E.g.: CG-CG: the (R1) phase on line 11 in Algorithm 4 is independent of the SPMV variable n_i.

Algorithm 6 Pipelined CG (preconditioned).

```
1:  function PREC-P-CG($A$, $M^{-1}$, $b$, $x_0$)
2:      $r_0 := b - Ax_0$; $u_0 := M^{-1}r_0$; $w_0 := Au_0$
3:      for $i = 0, 1, 2, \dots$ do
4:          begin reduction $\gamma_i := (r_i, u_i)$; $\delta := (w_i, u_i)$
5:          computation $m_i := M^{-1}w_i$
6:          computation $n_i := Am_i$
7:          end reduction
8:          if $i > 0$ then
9:              $\beta_i := \gamma_i/\gamma_{i-1}$; $\alpha_i := (\delta/\gamma_i - \beta_i/\alpha_{i-1})^{-1}$
10:         else
11:             $\beta_i := 0$; $\alpha_i := \gamma_i/\delta$
12:         end if
13:         $z_i := n_i + \beta_i z_{i-1}$
14:         $q_i := m_i + \beta_i q_{i-1}$
15:         $s_i := w_i + \beta_i s_{i-1}$
16:         $p_i := u_i + \beta_i p_{i-1}$
17:         $x_{i+1} := x_i + \alpha_i p_i$
18:         $r_{i+1} := r_i - \alpha_i s_i$
19:         $u_{i+1} := u_i - \alpha_i q_i$
20:         $w_{i+1} := w_i - \alpha_i z_i$
21:     end for
22: end function
```

2(d) Insert the new SPMV variables (S3) defined in (b) right after the global reduction phase (R1). These SPMV computations can now be overlapped with the global synchronization (R1), since they do not make use of the results computed in the (R1) phase.
E.g.: CG-CG: the SPMV $n_i = AM^{-1}w_i$ (S3) is inserted below the (R1) global reduction phase on line 11. Since there are no dependencies between these phases, they can be overlapped.

The resulting pipelined algorithm is shown in Algorithm 5. The proposed framework allows for the derivation of a pipelined Krylov subspace method with a length-one pipeline, similar to the p-CG and p-CR methods described in [25]. A general framework for derivation of methods with longer pipeline lengths, cf. the p-GMRES(l) method described in [24], is highly non-trivial and is left as future work.

E.g.: p-CG: Applying Step 2 to the CG-CG algorithm yields the communication-hiding pipelined CG method (p-CG) by Ghysels et al. [25]. This method is shown in Algorithm 6.

3. Derivation of the pipelined BiCGStab algorithm

3.1. The standard BiCGStab algorithm

The traditional Biconjugate Gradient Stabilized method (BiCGStab), shown in Algorithm 7, was developed as a fast and smoothly converging variant of the BiCG and CGS methods [41]. It presently serves as the standard *workhorse* Krylov subspace method for the iterative solution of nonsymmetric linear systems $Ax = b$, where A is a general real or complex matrix with generic spectral properties, be it positive definite, positive or negative semidefinite, or indefinite.

Algorithm 7 performs two SPMV applications (lines 4 and 8) and a total of 4 recursive vector updates (lines 7, 11, 12 and 15) in each iteration. When implemented on a parallel machine, the AXPYS used to compute the recurrences are local operations, requiring no communication between individual workers. The SPMVS can be considered semi-local operations, since only limited communication between neighboring workers is required for boundary elements. In the case of stencil application, or the application of the banded sparse matrix structures that typically result from the discretization of PDEs, data locality can be exploited to ensure limited communication is required for computing the SPMV. Hence, the SPMV operations are considered to be primarily compute-bound. The traditional BiCGStab algorithm additionally features three global reduction steps to compute the dot-products required in the calculation of the scalar variables α_i (line 5–6), ω_i (line 9–10) and β_i (line 13–14). On parallel machines these dot-products require global communication among all workers to assemble the locally computed dot-product fractions and redistribute the final scalar result to all workers.

3.2. Step 1: Avoiding global communication: towards CA-BiCGStab

Starting from the original BiCGStab Algorithm 7, first the number of global communication phases is reduced. To this aim the dot-product for the computation of α_i (line 5) is merged with the global reduction phase required to compute β_i

Algorithm 7 Standard BiCGStab.

```
1:  function BICGSTAB($A$, $b$, $x_0$)
2:      $r_0 := b - Ax_0$; $p_0 := r_0$
3:      for $i = 0, 1, 2, \dots$ do
4:          computation $s_i := Ap_i$
5:          begin reduction $(r_0, s_i)$ end reduction
6:          $\alpha_i := (r_0, r_i)/(r_0, s_i)$
7:          $q_i := r_i - \alpha_i s_i$
8:          computation $y_i := Aq_i$
9:          begin reduction $(q_i, y_i)$; $(y_i, y_i)$ end reduction
10:         $\omega_i := (q_i, y_i)/(y_i, y_i)$
11:         $x_{i+1} := x_i + \alpha_i p_i + \omega_i q_i$
12:         $r_{i+1} := q_i - \omega_i y_i$
13:         begin reduction $(r_0, r_{i+1})$ end reduction
14:         $\beta_i := (\alpha_i/\omega_i)(r_0, r_{i+1})/(r_0, r_i)$
15:         $p_{i+1} := r_{i+1} + \beta_i(p_i - \omega_i s_i)$
16:     end for
17: end function
```

(line 13). Rewriting the intermediate SPMV $s_i = Ap_i$ by using the recurrence for p_i on line 15, we obtain:

$$\begin{aligned} s_i = Ap_i &= A(r_i + \beta_{i-1}(p_{i-1} - \omega_{i-1}s_{i-1})) \\ &= w_i + \beta_{i-1}(s_{i-1} - \omega_{i-1}z_{i-1}), \end{aligned} \tag{1}$$

where we use that $s_{i-1} = Ap_{i-1}$, and the auxiliary variables $w_i = Ar_i$ and $z_i = As_i$ are defined. Subsequently, the new recurrence for s_i (1) is applied to rewrite the dot-product (r_0, s_i) (line 5), so that α_i becomes independent of the intermediate variables s_i and p_{i+1}, i.e.,

$$\begin{aligned} (r_0, s_i) &= (r_0, w_i + \beta_{i-1}(s_{i-1} - \omega_{i-1}z_{i-1})) \\ &= (r_0, w_i) + \beta_{i-1}(r_0, s_{i-1}) - \beta_{i-1}\omega_{i-1}(r_0, z_{i-1}). \end{aligned} \tag{2}$$

By substituting (2) into the definition of $\alpha_i = (r_0, r_i)/(r_0, s_i)$, the new expression for α_i becomes

$$\alpha_i = \frac{(r_0, r_i)}{(r_0, w_i) + \beta_{i-1}(r_0, s_{i-1}) - \beta_{i-1}\omega_{i-1}(r_0, z_{i-1})}, \tag{3}$$

which requires the dot-products (r_0, r_i), (r_0, w_i), (r_0, s_{i-1}) and (r_0, z_{i-1}), but no longer uses the dot-product (r_0, s_i). Since α_i is now independent of the intermediate variables s_i and p_{i+1}, it can be moved upward, and its global reduction phase can be merged with the global reduction required to compute β_i (line 13). The SPMV $y_i = Aq_i$ (line 8) can also be replaced by a recurrence by using the new variables $w_i = Ar_i$ and $z_i = As_i$, i.e.,

$$y_i = Aq_i = A(r_i - \alpha_i s_i) = w_i - \alpha_i z_i. \tag{4}$$

Hence, the total number of SPMVs to be computed is two, similar to the original BiCGStab algorithm. After a minor reordering of operations, this leads to the communication-avoiding version of BiCGStab shown in Algorithm 8. Note that this algorithm resembles the BiCGStab variant proposed in [30].

Although the total number of dot-products in Algorithm 8 is two higher than in the original BiCGStab algorithm, the number of global reduction phases was reduced from three in Algorithm 7 to only two in Algorithm 8. The first reduction phase in Algorithm 8 (line 9) is unaltered compared to Algorithm 7 (line 9). In the second global reduction the dot-products (r_0, r_{i+1}), (r_0, w_{i+1}), (r_0, s_i) and (r_0, z_i) are communicated simultaneously. This uses slightly more memory bandwidth, but leads to only one global synchronization point to compute both α_{i+1} and β_i in Algorithm 8 (line 14). The extra bandwidth usage for performing multiple dot-product communications in the same global synchronization is virtually negligible, since communication is limited to scalars only.

Note that the expression (3) for α_{i+1} can alternatively be rewritten as

$$\alpha_{i+1} = \left(\frac{1}{\omega_i} + \frac{(r_0, w_{i+1})}{(r_0, r_{i+1})} - \beta_i \omega_i \frac{(r_0, z_i)}{(r_0, r_{i+1})} \right)^{-1}. \tag{5}$$

The expression for α_{i+1} above is equivalent to (3) in exact arithmetic, but uses three dot-products instead of four. Since the number of global reductions remains unaffected by this operation, replacing the expression for α_{i+1} by (5) has no effect on the time spent in communication. Numerical experiments in finite precision arithmetic have shown that expression (3) results in a more robust variant of the BiCGStab algorithm, in particular when combined with a residual replacement strategy, see Section 4.

Algorithm 8 Communication-avoiding BiCGStab.

```
1: function CA-BICGSTAB($A$, $b$, $x_0$)
2:     $r_0 := b - Ax_0$; $w_0 := Ar_0$; $\alpha_0 := (r_0, r_0)/(r_0, w_0)$; $\beta_{-1} := 0$
3:     for $i = 0, 1, 2, \dots$ do
4:         $p_i := r_i + \beta_{i-1}(p_{i-1} - \omega_{i-1}s_{i-1})$
5:         $s_i := w_i + \beta_{i-1}(s_{i-1} - \omega_{i-1}z_{i-1})$
6:         computation $z_i := As_i$
7:         $q_i := r_i - \alpha_i s_i$
8:         $y_i := w_i - \alpha_i z_i$
9:         begin reduction $(q_i, y_i)$; $(y_i, y_i)$ end reduction
10:        $\omega_i := (q_i, y_i)/(y_i, y_i)$
11:        $x_{i+1} := x_i + \alpha_i p_i + \omega_i q_i$
12:        $r_{i+1} := q_i - \omega_i y_i$
13:        computation $w_{i+1} := Ar_{i+1}$
14:        begin reduction $(r_0, r_{i+1})$; $(r_0, w_{i+1})$; $(r_0, s_i)$; $(r_0, z_i)$ end reduction
15:        $\beta_i := (\alpha_i/\omega_i)(r_0, r_{i+1})/(r_0, r_i)$
16:        $\alpha_{i+1} := (r_0, r_{i+1})/((r_0, w_{i+1}) + \beta_i(r_0, s_i) - \beta_i\omega_i(r_0, z_i))$
17:    end for
18: end function
```

3.3. Step 2: Hiding global communication: towards p-BiCGStab

Following Step 1, the modified BiCGStab Algorithm 8 now features two SPMV operations (lines 6 and 13), which are each followed by a global reduction phase (lines 9 and 14). The first SPMV-global reduction pair (lines 6 and 9) is separated by intermediate AXPY operations. Starting from this algorithm, we now aim at formulating a communication-hiding version of BiCGStab by moving the SPMV operations below the global reduction phases.

The SPMV $z_i = As_i$ in Algorithm 8 (line 6) is rewritten using the recurrence (1) for s_i:

$$\begin{aligned} z_i = As_i &= A(w_i + \beta_{i-1}(s_{i-1} - \omega_{i-1}z_{i-1})) \\ &= t_i + \beta_{i-1}(z_{i-1} - \omega_{i-1}v_{i-1}). \end{aligned} \tag{6}$$

Here we use that $z_{i-1} = As_{i-1}$, and two new auxiliary variables are defined as the SPMVs $t_i = Aw_i$ and $v_i = Az_i$. In a similar way the second SPMV $w_{i+1} = Ar_{i+1}$ can be rewritten as a recurrence using the recursive definition of r_{i+1}, i.e.,

$$r_{i+1} = q_i - \omega_i y_i = r_i - \alpha_i s_i - \omega_i(w_i - \alpha_i z_i), \tag{7}$$

see lines 7, 8 and 12. From this expression we obtain the following recurrence for w_{i+1}:

$$\begin{aligned} w_{i+1} = Ar_{i+1} &= A(q_i - \omega_i(w_i - \alpha_i z_i)) \\ &= y_i - \omega_i(t_i - \alpha_i v_i), \end{aligned} \tag{8}$$

where we use the definitions $w_i = Ar_i$, $z_i = As_i$, $t_i = Aw_i$ and $v_i = Az_i$. Next, the dot-products are reformulated such that they are independent of the corresponding newly defined SPMVs. For the first global reduction on line 9 in Algorithm 8, this implies q_i and y_i should be independent of v_i, which is clearly the case, and for the second global reduction on line 14, the variables r_{i+1}, w_{i+1}, s_i and z_i are required to be independent of t_{i+1}, which is also trivially satisfied. Hence, the SPMV $v_i = Az_i$ can be computed below the first global reduction phase (line 9), and the second new SPMV $t_{i+1} = Aw_{i+1}$ is moved below the corresponding second global reduction (line 14). This results in the pipelined BiCGStab method shown in Algorithm 9.

Mathematically, i.e., in exact arithmetic, Algorithm 9 is equivalent to standard BiCGStab. Moreover, similar to Algorithm 8, the pipelined BiCGStab Algorithm 9 features only two global communication phases, yet it additionally allows for a communication-hiding strategy. After local computation of the dot-product contributions has been executed on each worker, each global reduction (lines 9 and 16) can be overlapped with the computation of the corresponding SPMV (lines 10 and 17). This overlap hides (part of) the communication latency behind the SPMV computations. In the remainder of this work Algorithm 9 shall be referred to as unpreconditioned pipelined BiCGStab.

In finite precision arithmetic, the pipelined BiCGStab method may display different convergence behavior compared to standard BiCGStab due to the different way rounding errors are handled by the algorithm. Notably, pipelined BiCGStab features 8 lines of recursive vector operations, whereas traditional BiCGStab has only 4 lines with vector recurrences. The authors refer to the extensive literature [14,26–28,31–33,39,40] and their own related work [11] for a more elaborate discussion on the relation between the number of AXPYS and the propagation of local rounding errors throughout various variants of Krylov subspace algorithms. Hence, the p-BiCGStab residuals may deviate slightly from their original BiCGStab counterparts. Moreover, the maximal attainable accuracy of the p-BiCGStab method may be affected by the reordering of the algorithm. This is illustrated in Section 4, where countermeasures to the accuracy loss are proposed.

Algorithm 9 Pipelined BiCGStab.

```
1: function P-BICGSTAB($A$, $b$, $x_0$)
2:     $r_0 := b - Ax_0$; $w_0 := Ar_0$; $t_0 := Aw_0$; $\alpha_0 := (r_0, r_0)/(r_0, w_0)$; $\beta_{-1} := 0$
3:     for $i = 0, 1, 2, \ldots$ do
4:         $p_i := r_i + \beta_{i-1}(p_{i-1} - \omega_{i-1}s_{i-1})$
5:         $s_i := w_i + \beta_{i-1}(s_{i-1} - \omega_{i-1}z_{i-1})$
6:         $z_i := t_i + \beta_{i-1}(z_{i-1} - \omega_{i-1}v_{i-1})$
7:         $q_i := r_i - \alpha_i s_i$
8:         $y_i := w_i - \alpha_i z_i$
9:         begin reduction $(q_i, y_i)$; $(y_i, y_i)$
10:        computation $v_i := Az_i$
11:        end reduction
12:        $\omega_i := (q_i, y_i)/(y_i, y_i)$
13:        $x_{i+1} := x_i + \alpha_i p_i + \omega_i q_i$
14:        $r_{i+1} := q_i - \omega_i y_i$
15:        $w_{i+1} := y_i - \omega_i(t_i - \alpha_i v_i)$
16:        begin reduction $(r_0, r_{i+1})$; $(r_0, w_{i+1})$; $(r_0, s_i)$; $(r_0, z_i)$
17:        computation $t_{i+1} := Aw_{i+1}$
18:        end reduction
19:        $\beta_i := (\alpha_i/\omega_i)(r_0, r_{i+1})/(r_0, r_i)$
20:        $\alpha_{i+1} := (r_0, r_{i+1})/((r_0, w_{i+1}) + \beta_i(r_0, s_i) - \beta_i\omega_i(r_0, z_i))$
21:    end for
22: end function
```

Table 1
Specifications of different unpreconditioned BiCGStab variations. Columns GLRED and SPMV list the number of global reduction phases and SPMVS per iteration respectively. The *-symbol indicates that SPMVS are overlapped with global reductions. Column *Flops* shows the number of flops ($\times N$) required to compute AXPYS and dot-products. The *Time* column has the time spent in global all-reduce communications (GLREDS) and SPMVS. *Memory* counts the total number of vectors that need to be kept in memory.

	GLRED	SPMV	Flops (AXPY + DOT-PROD)	Time (GLRED + SPMV)	Memory
BiCGStab	3	2	20	3 GLRED + 2 SPMV	7
IBiCGStab	1	2	30	1 GLRED + 2 SPMV	10
p-BiCGStab	2	2*	38	2 max(GLRED, SPMV)	11
s-step CA-BiCGStab	$1/s$	4	$32s + 45$	$1/s$ GLRED + 4 SPMV	$4s + 5$

3.4. Pipelined BiCGStab vs. Improved BiCGStab

The derivation of the pipelined BiCGStab method, Algorithm 9, is essentially but one of several possible ways to obtain a more efficient parallel variant of the BiCGStab algorithm. By performing the recursive substitutions and re-orderings of the algorithm in a different manner, other parallel algorithms that are mathematically equivalent to standard BiCGStab but feature improved global communication properties may be derived. In particular, it is possible to reduce the number of global reductions even further after obtaining Algorithm 8 in Section 3.2. This is achieved by merging the reduction required to compute ω_i (line 9) with the global reduction for α_{i+1} and β_i (line 14). The resulting algorithm comprises only one global reduction step per iteration. This algorithm has been proposed by Yang et al. in 2002 as the Improved BiCGStab method (IBiCGStab) [44]. Since only one global reduction step (avoiding) but no SPMV overlap (hiding) is performed, the IBiCGStab method performs well on setups with a heavy communication-to-computation ratio.

We refer to Table 1 for a comparison between the unpreconditioned IBiCGStab and p-BiCGStab algorithms. The *Flops* and *Memory* requirements for both methods are largely comparable. Only the two main time-consuming components (GLREDS and SPMVS) are taken into account for the *Time* estimates; the AXPY operations are neglected. In the ideal scenario for pipelining where the SPMV computation perfectly overlaps the communication time of one global reduction, the IBiCGStab method achieves a speed-up of approximately $5/3 = 1.67\times$ compared to standard BiCGStab, whereas the p-BiCGStab method is theoretically able to attain a speed-up factor of $5/2 = 2.5\times$.

It is possible to combine the communication-avoiding strategy of the IBiCGStab method with a communication-hiding strategy. Starting from the IBiCGStab algorithm, additional recurrences for auxiliary variables are introduced to merge all SPMVS into one block that directly follows the global reduction. This results in a hybrid pipelined IBiCGStab algorithm with only one global all-reduce communication phase, overlapped by SPMVS, per iteration. However, to reorganize the algorithm to this state, an additional SPMV is introduced, increasing the total number of SPMVS per iteration from two to three. This is undesirable; the reduction of global communication latency should not come at the expense of an increased computational cost. More importantly, a large number of auxiliary axpys is required, which threatens the stability of the algorithm and leading to very low attainable accuracy. We therefore do not expound on the details of this method.

3.5. Pipelined BiCGStab vs. *s*-step CA-BiCGStab

Algorithm 8 is denoted 'CA-BiCGStab' in this work to distinguish it from the standard BiCGStab Algorithm 7, which features more global reductions, and from the pipelined Algorithm 9, which overlaps global reductions with computational work. Despite the nominal similarity, Algorithm 8 is unrelated to the *s*-step CA-BiCGStab algorithm proposed by Carson et al. in [3,5]. The latter algorithm is not obtained by reorganizing subsequent algorithmic operations to reduce the number of global reductions like Algorithm 8. Instead, *s*-step CA-BiCGStab simultaneously orthogonalizes *s* Krylov subspace basis vectors prior to every *s* iterations, such that the time spent in global communication is roughly divided by a factor of *s*.

Table 1 lists specifications of the *s*-step CA-BiCGStab algorithm from [3], p.54, Algorithm 12. The *s*-step algorithm is very efficient for problems where the time spent by the global reduction phase significantly exceeds the time of computing an spmv, as is shown by the *Time* estimates. The p-BiCGStab method, on the other hand, focuses on overlapping communication and computations, and thus benefits most from an approximately equal execution time for global reductions and spmvs. Of particular interest in this context is the work on pipelined methods with a pipeline length of *l* iterations, presented for GMRES in [24], which allows to overlap the global reduction by multiple iterations. Currently no *l*-length version of the pipelined BiCGStab method is available. We aim to address this non-trivial question in future research. Note that for *s*-step methods, the inclusion of a preconditioner is generally challenging, whereas it is (theoretically) straightforward in the case of pipelined methods. This is illustrated for the p-BiCGStab method in Section 3.6.

Compared to other BiCGStab variants, the *s*-step CA-BiCGStab method requires a larger average number of spmvs per iteration and significantly more flops to compute the growing number of axpys and dot-products. The large number of extra operations affects the stability of the method for increasing values of *s*, see the analysis in [3]. The latter phenomenon is also observed for pipelined methods, and is treated in more detail for p-BiCGStab in Section 4 of this work.

3.6. Preconditioned pipelined BiCGStab

The derivation of the pipelined BiCGStab algorithm can easily be extended to the preconditioned BiCGStab method shown in Algorithm 10. A right-preconditioned system $AM^{-1}y = b$ with $Mx = y$ is considered, where M is the preconditioning op-

Algorithm 10 Preconditioned BiCGStab.

```
1:  function PREC-BICGSTAB($A$, $M^{-1}$, $b$, $x_0$)
2:      $r_0 := b - Ax_0;\ p_0 := r_0$
3:      for $i = 0, 1, 2, \ldots$ do
4:          computation $\hat{p}_i := M^{-1}p_i$
5:          computation $s_i := A\hat{p}_i$
6:          begin reduction $(r_0, s_i)$ end reduction
7:          $\alpha_i := (r_0, r_i)/(r_0, s_i)$
8:          $q_i := r_i - \alpha_i s_i$
9:          computation $\hat{q}_i := M^{-1}q_i$
10:         computation $y_i := A\hat{q}_i$
11:         begin reduction $(q_i, y_i);\ (y_i, y_i)$ end reduction
12:         $\omega_i := (q_i, y_i)/(y_i, y_i)$
13:         $x_{i+1} := x_i + \alpha_i\hat{p}_i + \omega_i\hat{q}_i$
14:         $r_{i+1} := q_i - \omega_i y_i$
15:         begin reduction $(r_0, r_{i+1})$ end reduction
16:         $\beta_i := (\alpha_i/\omega_i)(r_0, r_{i+1})/(r_0, r_i)$
17:         $p_{i+1} := r_{i+1} + \beta_i(p_i - \omega_i s_i)$
18:     end for
19: end function
```

erator. The preconditioned algorithm differs only slightly from Algorithm 7: lines 4 and 9 are added to implement the preconditioner, and the solution update on line 13 now uses the preconditioned direction vectors; the remaining expressions in Algorithm 10 are identical to the ones in Algorithm 7. The preconditioned variables are denoted by their respective variable with a hat-symbol, e.g. $\hat{p}_i = M^{-1}p_i$ and $\hat{q}_i = M^{-1}q_i$.

The preconditioned pipelined BiCGStab method is derived starting from Algorithm 10 following a framework that is comparable to the derivation of unpreconditioned pipelined algorithm. However, in this case the preconditioned variables $\hat{p}_i$ (line 4) and $\hat{q}_i$ (line 9) are also reformulated as recurrences where required, similar to the reformulation of the spmvs for s_i (line 5) and y_i (line 10). First, the number of global communication phases is reduced by joining the global reductions required to compute α_i and β_i together into one global reduction phase. To this aim, we rewrite the variables $\hat{p}_i$ and s_i in the first preconditioned SPMV (line 4–5) as recurrences, i.e.,

$$\begin{aligned}\hat{p}_i = M^{-1}p_i &= M^{-1}(r_i + \beta_{i-1}(p_{i-1} - \omega_{i-1}s_{i-1})) \\ &= \hat{r}_i + \beta_{i-1}(\hat{p}_{i-1} - \omega_{i-1}\hat{s}_{i-1}),\end{aligned} \tag{9}$$

where the new variables $\hat{r}_i := M^{-1}r_i$ and $\hat{s}_i := M^{-1}s_i$ are defined, and furthermore we write

$$\begin{aligned}s_i = A\hat{p}_i &= A(\hat{r}_i + \beta_{i-1}(\hat{p}_{i-1} - \omega_{i-1}\hat{s}_{i-1})) \\ &= w_i + \beta_{i-1}(s_{i-1} - \omega_{i-1}z_{i-1}),\end{aligned} \tag{10}$$

where $\hat{p}_i$ is substituted by the recurrence (9), and the variables $w_i = A\hat{r}_i = AM^{-1}r_i$ and $z_i = A\hat{s}_i = AM^{-1}s_i$ are introduced. Furthermore, using the definitions above, the variables $\hat{q}_i$ and y_i in the second preconditioned SPMV (line 9–10) are rewritten as recurrence relations, yielding for $\hat{q}_i = M^{-1}q_i$

$$\hat{q}_i = M^{-1}q_i = M^{-1}(r_i - \alpha_i s_i) = \hat{r}_i - \alpha_i\hat{s}_i, \tag{11}$$

and for the variable $y_i = A\hat{q}_i$ the recurrence

$$y_i = A\hat{q}_i = A(\hat{r}_i - \alpha_i\hat{s}_i) = w_i - \alpha_i z_i. \tag{12}$$

Observe how the recurrences (10) and (12) for s_i and y_i respectively are identical to their counterparts (1) and (4) in the unpreconditioned case. Consequently, definition (3) for α_i can again be used here. This expression does not depend on the variables p_{i+1}, $\hat{p}_i$ and s_i. The global reduction for α_i can thus be moved upward and merged with the global reduction for β_i. Hence, the number of global reductions is reduced from three to two at the cost of two additional recursive vector operations, resulting in a communication-avoiding preconditioned BiCGStab algorithm.

We then focus on hiding the global communication phases behind the preconditioned SPMVs, which will allow to overlap both the preconditioner application and the SPMV with the global reduction. The two preconditioned SPMVs that compute $\hat{s}_i = M^{-1}s_i$, $z_i = A\hat{s}_i$ and $\hat{r}_{i+1} = M^{-1}r_{i+1}$, $w_{i+1} = A\hat{r}_{i+1}$ are rewritten, such that the SPMVs can be placed below the corresponding global reduction phases. This is achieved by introducing recurrences and auxiliary variables as follows:

$$\begin{aligned}\hat{s}_i = M^{-1}s_i &= M^{-1}(w_i + \beta_{i-1}(s_{i-1} - \omega_{i-1}z_{i-1})) \\ &= \hat{w}_i + \beta_{i-1}(\hat{s}_{i-1} - \omega_{i-1}\hat{z}_{i-1}),\end{aligned} \tag{13}$$

where $\hat{w}_i = M^{-1}w_i$ and $\hat{z}_i = M^{-1}z_i$ are introduced, and

$$\begin{aligned}z_i = A\hat{s}_i &= A(\hat{w}_i + \beta_{i-1}(\hat{s}_{i-1} - \omega_{i-1}\hat{z}_{i-1})) \\ &= t_i + \beta_{i-1}(z_{i-1} - \omega_{i-1}v_{i-1}),\end{aligned} \tag{14}$$

where $t_i = A\hat{w}_i$ and $v_i = A\hat{z}_i$ are defined. Additionally, using these new definitions, the following recurrences for $\hat{r}_{i+1}$ and w_{i+1} can be derived

$$\begin{aligned}\hat{r}_{i+1} = M^{-1}r_{i+1} &= M^{-1}(q_i - \omega_i(w_i - \alpha_i z_i)) \\ &= \hat{q}_i - \omega_i(\hat{w}_i - \alpha_i\hat{z}_i),\end{aligned} \tag{15}$$

and

$$\begin{aligned}w_{i+1} = A\hat{r}_{i+1} &= A(\hat{q}_i - \omega_i(\hat{w}_i - \alpha_i\hat{z}_i)) \\ &= y_i - \omega_i(t_i - \alpha_i v_i).\end{aligned} \tag{16}$$

After moving the preconditioned SPMVs $\hat{w}_i = M^{-1}w_i$, $t_i = A\hat{w}_i$ and $\hat{z}_i = M^{-1}z_i$, $v_i = A\hat{z}_i$ below the dot-product computations of which they are independent, and following a minor reordering of operations, the above recurrences result in Algorithm 11. This algorithm is denoted as preconditioned pipelined BiCGStab.

Note that Algorithm 11 can alternatively be derived directly from the unpreconditioned pipelined BiCGStab Algorithm 9 by adding the proper preconditioned variables. This includes recursively computing $\hat{p}_i = M^{-1}p_i$ (line 5), $\hat{s}_i = M^{-1}s_i$ (line 7), $\hat{q}_i = M^{-1}q_i$ (line 10) and $\hat{r}_{i+1} = M^{-1}r_{i+1}$ (line 19), whose recurrences can be derived directly from the corresponding unpreconditioned variables, and defining the preconditioned variables $\hat{z}_i = M^{-1}z_i$ (line 13) and $\hat{w}_{i+1} = M^{-1}w_{i+1}$ (line 22), which are computed explicitly right before their respective SPMVs.

Similar to the unpreconditioned case, the preconditioned pipelined BiCGStab Algorithm 11 allows for the overlap of the two SPMV operations (lines 14 and 23) with the global reduction phases (lines 12 and 20). In addition, due to the consecutive application of the preconditioner M^{-1} (lines 13 and 22) and the operator A, the global reductions also overlap with preconditioning. Depending on the preconditioner choice, this possibly results in a much better hiding of global communication when the application of A is not sufficient to cover the global reduction time frame. For good parallel performance in practice the preconditioner application should be compute-bound, requiring only limited communication between neighboring processors.

Algorithm 11 Preconditioned pipelined BiCGStab.

```
1: function PREC-P-BICGSTAB($A$, $M^{-1}$, $b$, $x_0$)
2:     $r_0 := b - Ax_0$; $\hat{r}_0 := M^{-1}r_0$; $w_0 := A\hat{r}_0$; $\hat{w}_0 := M^{-1}w_0$
3:     $t_0 := A\hat{w}_0$; $\alpha_0 := (r_0, r_0)/(r_0, w_0)$; $\beta_{-1} := 0$
4:     for $i = 0, 1, 2, \dots$ do
5:         $\hat{p}_i := \hat{r}_i + \beta_{i-1}\left(\hat{p}_{i-1} - \omega_{i-1}\hat{s}_{i-1}\right)$
6:         $s_i := w_i + \beta_{i-1}(s_{i-1} - \omega_{i-1}z_{i-1})$
7:         $\hat{s}_i := \hat{w}_i + \beta_{i-1}\left(\hat{s}_{i-1} - \omega_{i-1}\hat{z}_{i-1}\right)$
8:         $z_i := t_i + \beta_{i-1}(z_{i-1} - \omega_{i-1}v_{i-1})$
9:         $q_i := r_i - \alpha_i s_i$
10:        $\hat{q}_i := \hat{r}_i - \alpha_i\hat{s}_i$
11:        $y_i := w_i - \alpha_i z_i$
12:        begin reduction $(q_i, y_i)$; $(y_i, y_i)$
13:        computation $\hat{z}_i := M^{-1}z_i$
14:        computation $v_i := A\hat{z}_i$
15:        end reduction
16:        $\omega_i := (q_i, y_i)/(y_i, y_i)$
17:        $x_{i+1} := x_i + \alpha_i\hat{p}_i + \omega_i\hat{q}_i$
18:        $r_{i+1} := q_i - \omega_i y_i$
19:        $\hat{r}_{i+1} := \hat{q}_i - \omega_i\left(\hat{w}_i - \alpha_i\hat{z}_i\right)$
20:        $w_{i+1} := y_i - \omega_i(t_i - \alpha_i v_i)$
21:        begin reduction $(r_0, r_{i+1})$; $(r_0, w_{i+1})$; $(r_0, s_i)$; $(r_0, z_i)$
22:        computation $\hat{w}_{i+1} := M^{-1}w_{i+1}$
23:        computation $t_{i+1} := A\hat{w}_{i+1}$
24:        end reduction
25:        $\beta_i := (\alpha_i/\omega_i)(r_0, r_{i+1})/(r_0, r_i)$
26:        $\alpha_{i+1} := (r_0, r_{i+1})/((r_0, w_{i+1}) + \beta_i(r_0, s_i) - \beta_i\omega_i(r_0, z_i))$
27:    end for
28: end function
```

4. Numerical results

In this section a variety of numerical experiments is reported to benchmark the convergence of the pipelined BiCGStab method. Results comparing BiCGStab and p-BiCGStab on a wide range of matrices from applications are presented in Section 4.1. A residual replacement strategy is presented in Section 4.2 to increase the maximal attainable accuracy and robustness of the pipelined method.

4.1. Pipelined bicgstab benchmark on matrix market problems

Numerical results on a wide range of different linear systems are presented. Table 2 lists a collection of test matrices from Matrix Market[1], with their respective condition number κ, number of rows N and total number of nonzero elements #nnz. A linear system with exact solution $\hat{x}_j = 1/\sqrt{N}$ and right-hand side $b = A\hat{x}$ is solved for each of these matrices. The system is solved using both standard BiCGStab, Algorithm 10, and the new pipelined BiCGStab variant, Algorithm 11. The initial guess is the all-zero vector $x_0 = 0$ for both methods. An Incomplete LU factorization preconditioner is included to reduce the number of Krylov iterations where applicable. The stopping criterion defined on the scaled recursive residual is $\|r_i\|_2/\|r_0\|_2 \le 10^{-6}$.

Table 2 lists the number of iterations required to reach the stopping criterion as well as the final true residual norm $\|b - Ax_i\|_2$ for both methods. For most problems the preset tolerance of 10^{-6} on the scaled residual is achievable by p-BiCGStab in a comparable number of iterations with respect to standard BiCGStab. Averaged over all matrices, 3.5% less iterations are required by the pipelined algorithm. For specific problems p-BiCGStab requires significantly more iterations (e.g. `bcsstk18`, `utm5940`), whereas for others less iterations are required (e.g. `bcsstm25`, `sstmodel`).

Fig. 1 illustrates the convergence histories corresponding to a few randomly selected Matrix Market problems listed in Table 2. The per-iteration residual norms for BiCGStab and p-BiCGStab on the `1138_bus` (top-left), `bcsstk18` (top-right), `bwm2000` (bottom-left) and `sherman3` (bottom-right) matrices are shown, all with ILU0 preconditioner except the `bwm2000` problem. The tolerance of 10^{-6} on the scaled residual used as stopping criterion in Table 2 is marked by the dashed black line. Note that a stagnation of the true residual (solid colored lines) does not necessarily imply stagnation of

[1] http://math.nist.gov/MatrixMarket/.

Table 2
Collection of matrices from Matrix Market with condition number $\kappa(A)$, number of rows/columns N and total number of nonzeros #nnz. A linear system with right-hand side $b = A\hat{x}$ where $\hat{x}_i = 1/\sqrt{N}$ is solved with each of these matrices with the presented BiCGStab and pipelined BiCGStab algorithms. The initial guess is all-zero $x_0 = 0$. An Incomplete LU preconditioner with zero fill-in (ILU0) is included where applicable. The number of iterations required to reach a tolerance of 1e-6 on the scaled residual, i.e., $\|r_i\|_2/\|r_0\|_2 \le 10^{-6}$, is shown along with the corresponding true residual norm $\|b - Ax_i\|_2$.

Matrix	Prec	$\kappa(A)$	N	#nnz	$\|r_0\|_2$	BiCGStab		p-BiCGStab	
						iter	$\|b - Ax_i\|_2$	iter	$\|b - Ax_i\|_2$
1138_bus	ILU	8.6e+06	1138	4054	4.3e+01	89	1.4 e–05	95	1.8 e–05
add32	ILU	1.4e+02	4960	19,848	8.0 e–03	19	5.9 e–09	19	5.9 e–09
bcsstk14	ILU	1.3e+10	1806	63,454	2.1e+09	315	1.6e+03	322	1.2e+03
bcsstk18	ILU	6.5e+01	11,948	149,090	2.6e+09	84	2.2e+03	102	2.0e+03
bcsstk26	ILU	1.7e+08	1922	30,336	3.5e+09	113	2.8e+03	107	2.9e+03
bcsstm25	-	6.1e+09	15,439	15,439	6.9e+07	928	6.8e+01	825	6.5e+01
bfw782a	ILU	1.7e+03	782	7514	3.2 e–01	72	1.1 e–07	65	6.2 e–08
bwm2000	-	2.4e+05	2000	7996	1.1e+03	1156	6.6 e–04	1162	9.1 e–04
cdde6	ILU	1.8e+02	961	4681	5.8 e–01	9	2.2 e–07	9	2.2 e–07
fidap014	-	3.5e+16	3251	65,747	2.7e+06	121	2.6e+00	123	2.6e+00
fs_760_3	ILU	1.0e+20	760	5816	1.6e+07	930	1.4e+01	709	1.1e+01
jagmesh9	-	6.0e+03	1349	9101	6.8e+00	1022	6.4 e–06	996	6.6 e–06
jpwh_991	ILU	1.4e+02	991	6027	3.8 e–01	9	2.9 e–07	9	2.9 e–07
orsreg_1	ILU	6.7e+03	2205	14,133	4.8e+00	25	2.7 e–06	25	2.7 e–06
pde2961	ILU	6.4e+02	2961	14,585	2.9 e–01	31	1.3 e–07	31	4.5 e–08
rdb3200l	-	1.1e+03	3200	18,880	1.0e+01	149	5.1 e–06	145	9.1 e–06
s3dkq4m2	-	1.9e+11	90,449	2,455,670	6.8e+01	3736	5.8 e–05	3500	6.2 e–05
saylr4	ILU	6.9e+06	3564	22,316	3.1 e–03	40	3.0 e–09	39	1.5 e–09
sherman3	ILU	5.5e+18	5005	20,033	1.8e+01	98	3.7 e–06	83	9.1 e–06
sstmodel	-	2.7e+18	3345	22,749	7.9e+00	6968	7.7 e–06	4399	7.5 e–06
utm5940	ILU	4.3e+08	5940	83,842	3.6 e–01	223	4.0 e–08	244	3.5 e–07
Average iter deviation wrt BiCGStab								–3.5%	

the recursive residual (dotted colored lines). The latter may continue to decrease although the solution no longer improves with additional iterations, cf [27].

Table 2 and Fig. 1 show that the residual norm history for BiCGStab and p-BiCGStab, although comparable, is not identical. This discrepancy is due to the different way rounding errors are propagated through the algorithms in finite precision arithmetic. Indeed, although BiCGStab and p-BiCGStab are mathematically equivalent algorithms, small differences in convergence may arise as a result of rounding error propagation in finite precision arithmetic. Due to the generally non-smooth convergence history of the BiCGStab method, these effects are more pronounced for BiCGStab compared to the pipelined Conjugate Gradient method [25], where the behavior of the traditional and pipelined algorithms is largely identical. We expound on the numerical stability of the algorithm in finite precision in Section 4.2.

4.2. Improving attainable accuracy: residual replacement strategy

Some applications demand a significantly higher accuracy on the solution than imposed by the relatively mild stopping criterion $\|r_i\|_2/\|r_0\|_2 \le 10^{-6}$ used in Section 4.1. Table 3 shows the maximal attainable accuracy, i.e., the minimal true residual norm $\min_i \|b - Ax_i\|_2$, and the corresponding number of iterations required to attain this accuracy for the BiCGStab and p-BiCGStab algorithms on the Matrix Market selection. The pipelined method is unable to reach the same maximal accuracy as standard BiCGStab. This is a known issue related to most communication-avoiding algorithms, see [4], and also affects pipelined Krylov subspace methods, cf [11,25].

Several orders of magnitude on maximal attainable precision are typically lost when switching to the pipelined algorithm. The accuracy loss is caused by the increased number of AXPY operations in the pipelined Algorithm 11, which features 11 vector recurrence operations, compared to only 4 recurrences in the standard preconditioned BiCGStab Algorithm 10. Additional recurrence relations are prone to introducing and amplifying local rounding errors in the algorithm, causing the residuals to level off sooner, as has been analyzed in a number of papers among which [11,27,28,31,39,40].

Although in practice the loss of maximal attainable accuracy is only problematic when a very high precision on the solution is required, from a numerical point of view it is nonetheless important to address this issue. Countermeasures in the form of a residual replacement strategy [4,27,35,36,42] are introduced to negate the effect of propagating rounding errors introduced by the additional recurrences in the pipelined method. The residual replacement technique resets the residuals r_i and $\hat{r}_i$, as well as the auxiliary variables w_i, s_i, $\hat{s}_i$ and z_i, to their true values every k iterations, i.e.,

$$r_i := b - Ax_i, \qquad \hat{r}_i := M^{-1}r_i, \qquad w_i := A\hat{r}_i,$$
$$s_i := A\hat{p}_i, \qquad \hat{s}_i := M^{-1}s_i, \qquad z_i := A\hat{s}_i.$$

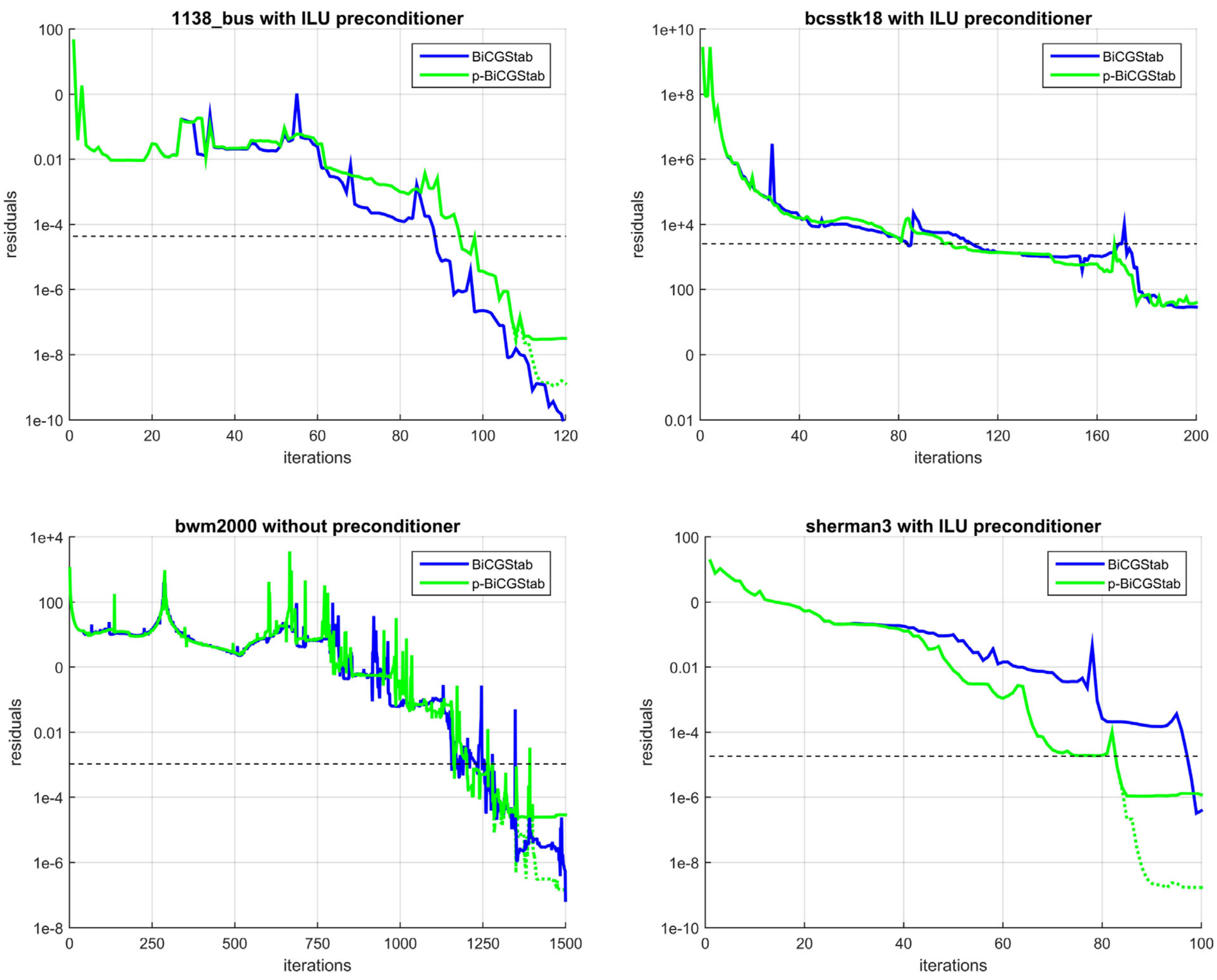


Fig. 1. Residual history of BiCGStab (blue) and p-BiCGStab (green) for different test matrices. Specifications: see Table 2. Solid color lines represent the true residual norms $\|b - Ax_i\|_2$, while the dotted color lines are the norms of the recursive residuals $\|r_i\|_2$. The dashed line marks the tolerance 1 e–6 on the scaled residual that was used as a stopping criterion. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

This induces an extra computational cost of 4 SPMVs and 2 preconditioner applications every k iterations. However, it is typically sufficient to perform only a small number of residual replacement steps to improve the maximal attainable accuracy, as indicated by the average #nrr percentage displayed in the bottom right corner of Table 3. Hence, the replacement period k is often large, limiting the computational overhead of performing the residual replacements. In Section 5 it is shown that the extra SPMV cost is negligible compared to the global cost of the method. The pipelined BiCGStab method with residual replacement is denoted as p-BiCGStab-rr for short.

By resetting the residual to its true value every k-th iteration, any build-up rounding errors are effectively eliminated. Through the periodic removal of rounding errors the algorithm is able to attain the same maximal accuracy as standard BiCGStab. This is illustrated by the p-BiCGStab-rr residual norms in Table 3. However, the introduction of the residual replacement step alters the p-BiCGStab convergence behavior, resulting in an increase in iterations required to attain the maximum precision on the solution for some problems. This delayed convergence [39] is clearly visible in Fig. 2, where the convergence histories (true residual norms) of the BiCGStab and p-BiCGStab methods on four selected matrices are shown up to a high accuracy.

Table 3 indicates an average reduction of iterations by 11.0% for p-BiCGStab compared to standard BiCGStab over all listed matrices. These iteration numbers are however related to the lower maximal attainable accuracy of p-BiCGStab, and in general do not imply p-BiCGStab converges faster than the original BiCGStab method, cf. Fig. 2. For the p-BiCGStab-rr method an average increase in iterations of 22.1% is observed. Indeed, for some problems such as `1138_bus`, `fs_760_3` and `utm5940` a significant increase in iterations and convergence irregularity is observed, see also Fig. 2. Final residuals displayed in Table 3 show that a high accuracy on the solution *can* be achieved by use of the residual replacement strategy

Table 3
Collection of matrices from Matrix Market. See Table 2 for additional specifications. A linear system with right-hand side $b = A\hat{x}$ where $\hat{x}_i = 1/\sqrt{N}$ is solved with each of these matrices with the BiCGStab, p-BiCGStab and p-BiCGStab-rr algorithms. The initial guess is all-zero $x_0 = 0$. An Incomplete LU preconditioner with zero fill-in (ILU0) is included where applicable, see Table 2. The number of iterations required to reach the maximal attainable accuracy on the residual is shown, along with the corresponding true residual norm $\|b - Ax_i\|_2$. A '-' symbol denotes failure to converge to a tolerance of 1 e–8 on the scaled residual, i.e., $\|r_i\|_2/\|r_0\|_2 \le 10^{-8}$, within 10,000 iterations. For the p-BiCGStab-rr method the table indicates the replacement period k and the total number of replacement steps #nrr.

Matrix	$\|r_0\|_2$	BiCGStab		p-BiCGStab		p-BiCGStab-rr			
		iter	$\|b - Ax_i\|_2$	iter	$\|b - Ax_i\|_2$	iter	$\|b - Ax_i\|_2$	k	#nrr
1138_bus	4.3e+01	124	1.8 e–11	130	4.0 e–09	220	7.4 e–12	35	3
add32	8.0 e–03	46	7.8 e–18	42	5.0 e–16	51	5.7 e–18	10	2
bcsstk14	2.1e+09	559	7.3 e–06	444	6.6 e–01	522	3.8 e–03	200	2
bcsstk18	2.6e+09	523	4.8 e–06	450	1.1 e–01	725	2.8 e–05	50	7
bcsstk26	3.5e+09	414	1.1 e–05	216	5.7 e–01	475	8.6 e–04	30	6
bcsstm25	6.9e+07	-	3.2e+00	-	3.8e+00	-	4.6e+00	1000	9
bfw782a	3.2 e–01	117	9.5 e–14	106	5.1 e–13	133	2.6 e–15	20	4
bwm2000	1.1e+03	1733	2.5 e–09	1621	1.4 e–05	2231	3.8 e–08	500	3
cdde6	5.8 e–01	151	8.1 e–14	147	2.0 e–11	159	2.1 e–15	10	13
fidap014	2.7e+06	-	4.3 e–03	-	9.7 e–03	-	4.3 e–03	50	3
fs_760_3	1.6e+07	1979	1.2 e–05	1039	5.1 e–02	4590	1.1 e–05	900	3
jagmesh9	6.8e+00	6230	2.4 e–14	3582	5.8 e–09	9751	1.1 e–11	500	13
jpwh_991	3.8 e–01	53	1.3 e–14	54	1.8 e–12	63	2.5 e–15	10	4
orsreg_1	4.8e+00	51	4.0 e–11	52	3.7 e–09	56	5.9 e–12	10	3
pde2961	2.9 e–01	50	4.5 e–15	48	3.4 e–13	52	1.4 e–15	10	3
rdb3200l	1.0e+01	178	3.7 e–08	167	9.9 e–08	181	3.4 e–08	100	1
s3dkq4m2	6.8e+01	-	1.0 e–05	-	1.4 e–05	-	1.3 e–05	1000	9
saylr4	3.1 e–03	52	4.0 e–12	43	7.5 e–11	46	1.8 e–12	10	4
sherman3	1.8e+01	111	2.5 e–11	100	2.8 e–07	128	6.2 e–11	20	4
sstmodel	7.9e+00	-	5.1 e–06	-	3.1 e–06	-	4.5 e–06	1000	9
utm5940	3.6 e–01	256	3.0 e–12	248	4.3 e–08	395	2.9 e–11	100	3
Average iter deviation wrt BiCGStab				-11.0%		22.1%			
Average #nrr wrt p-BiCGStab-rr iter									2.4%

when required by the application. The associated increase in iterations implies a trad e–off between speedup and accuracy that should be kept in mind when using the p-BiCGStab-rr method.

Note that the replacement period parameter is chosen manually for the matrices in Table 3, based on an ad hoc estimation of the total number of BiCGStab iterations. These values are relatively arbitrary, and a different choice of this parameter could lead to either significantly slower or faster convergence of the p-BiCGStab-rr algorithm.

Fig. 2 illustrates an additional beneficial effect of the residual replacement strategy on the pipelined BiCGStab method. The convergence history of the p-BiCGStab method is generally comparable (albeit not identical) to the BiCGStab residuals, up to the stagnation point where p-BiCGStab attains maximal accuracy. However, it is observed that after some iterations the p-BiCGStab true residuals start to increase again. Standard BiCGStab does not display this unwanted behavior. By periodically resetting the residual and auxiliary variables to their true values, the residual replacement strategy resolves these robustness issues. Fig. 2 shows a stagnation of the p-BiCGStab-rr residual norm similar to standard BiCGStab after maximal accuracy is attained.

5. Parallel performance

In this section we illustrate the parallel performance of the pipelined BiCGStab method, Algorithm 11. The scaling and accuracy experiments in this section are performed on a small cluster with 20 compute nodes, consisting of two 6-core Intel Xeon X5660 Nehalem 2.80 GHz processors each (12 cores per node), for a total of 240 cores. Nodes are connected by $4 \times$ QDR InfiniBand technology, providing 32 Gb/s of point-to-point bandwidth for message passing and I/O. Since each node consists of 12 cores, we use 12 MPI processes per node to fully exploit parallelism on the machine. The MPI library used for this experiment is MPICH-3.1.3[2]. The environment variables `MPICH_ASYNC_PROGRESS=1` and `MPICH_MAX_THREAD_SAFETY=multiple` are set to ensure optimal parallelism[3]; the first variable enables asynchronous non-blocking reductions, while the second allows for a process to have multiple threads which simultaneously call MPI functions.

Parallel test problem 1. The pipelined BiCGStab Algorithm 11 was implemented in the open-source PETSc library [1], version 3.6.2, as a direct modification of the `fbcgs` implementation found in the PETSc Krylov solvers (KSP) folder. The first

[2] http://www.mpich.org/.

[3] We point out that the need for these settings depends on the specific hard- and firmware used in practice.

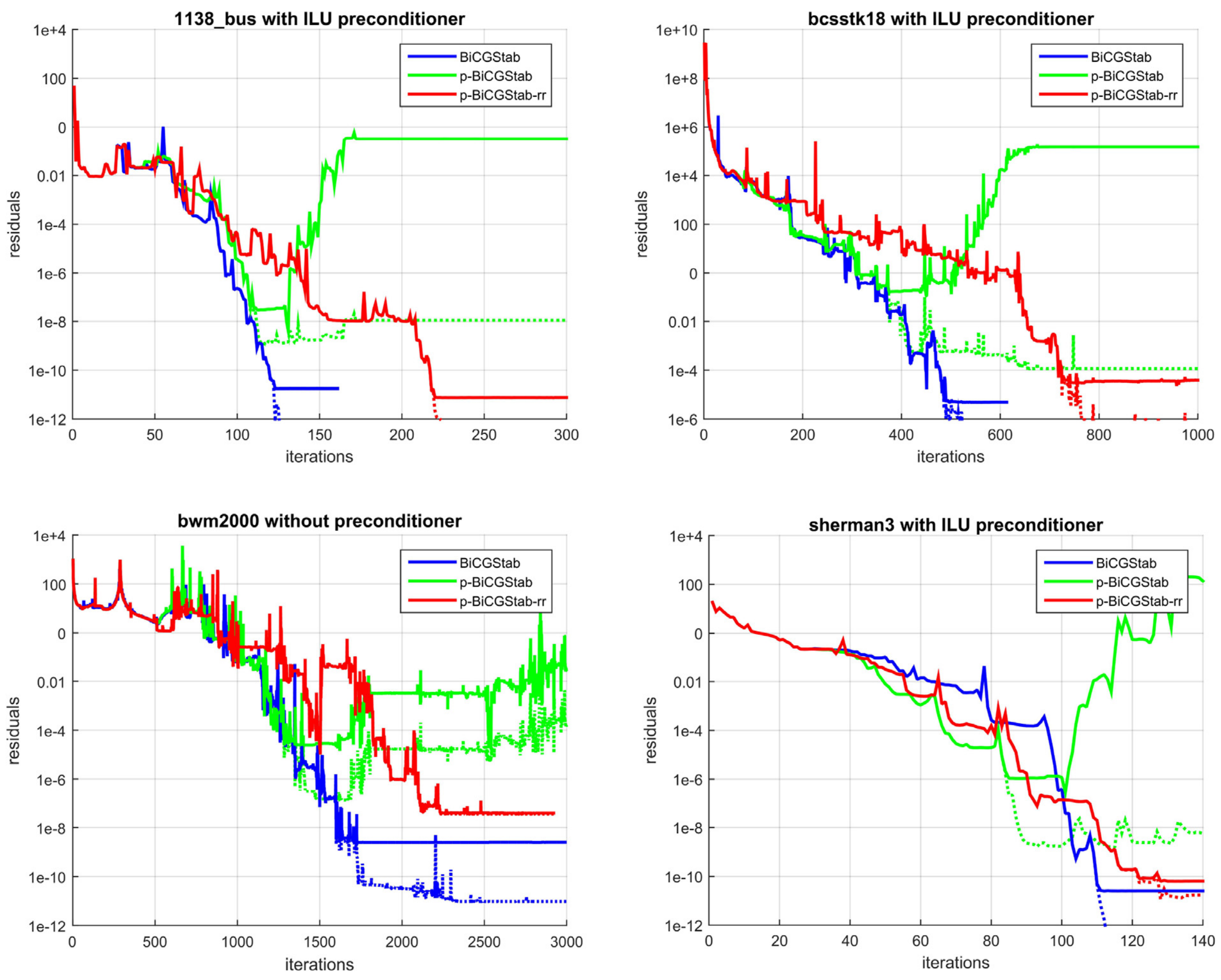


Fig. 2. Residual history of BiCGStab (blue), p-BiCGStab (green) and p-BiCGStab-rr (red) for different test matrices. Specifications: see Table 3. The dotted lines are the norms of the recursive residuals $\|r_i\|_2$, while solid lines represent the true residual norms $\|b - Ax_i\|_2$. The p-BiCGStab residuals stagnate several orders of magnitude above the standard BiCGStab residuals. The residual replacement strategy improves the attainable accuracy of the p-BiCGStab method. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

benchmark problem is a 2D PD e–type model, defined by the unsymmetric 5-point stencil

$$A_{1\text{st}} = \begin{bmatrix} & -1 & \\ -1 & 4 & -\varepsilon \\ & -\varepsilon & \end{bmatrix}, \quad \varepsilon = 1 - 0.001. \tag{PTP1}$$

The right-hand side $b = A_1\hat{x}$ is constructed using the exact solution $\hat{x} = \mathbf{1}$. The operator is a modified 2D Poisson PDE stencil, discretized using second order finite difference approximations on a uniform grid, which is available in the PETSc distribution as example 2 in the Krylov solvers folder. Although academic, the model problem (PTP1) is non-trivial from an iterative solver perspective, since a large number of the operator's eigenvalues are located close to zero. The number of grid points is set to 1.000 per spatial dimension, resulting in a total of one million unknowns.

As illustrated in Section 3.6, the inclusion of a preconditioner in pipelined algorithms is generally straightforward. However, to efficiently overlap the preconditioner application with the global communication phase, the preconditioner itself should not be bottl e–necked by communication. As such, a simple block Jacobi preconditioner is trivially well-suited for this purpose, whereas the inclusion of a more advanced preconditioning scheme like parallel ILU [6] or additive Schwarz [37] requires a more careful treatment. For simplicity no preconditioner is included in the following experiment.

Fig. 3 (top left) shows the average time per iteration required to solve the problem up to a tolerance of 10^{-6} on the scaled residual as a function of the number of nodes. For this test problem and hardware configuration, pipelined BiCGStab method (green) starts to outperform standard BiCGStab (blue) when the number of nodes exceeds four. Fig. 3 (top right) shows the same data, reformulated as speedup over standard 1-node BiCGStab. The p-BiCGStab method scales well up to 20 nodes. The bottom row in Fig. 3 shows the non-averaged total time (bottom left) and absolute speedup in function of

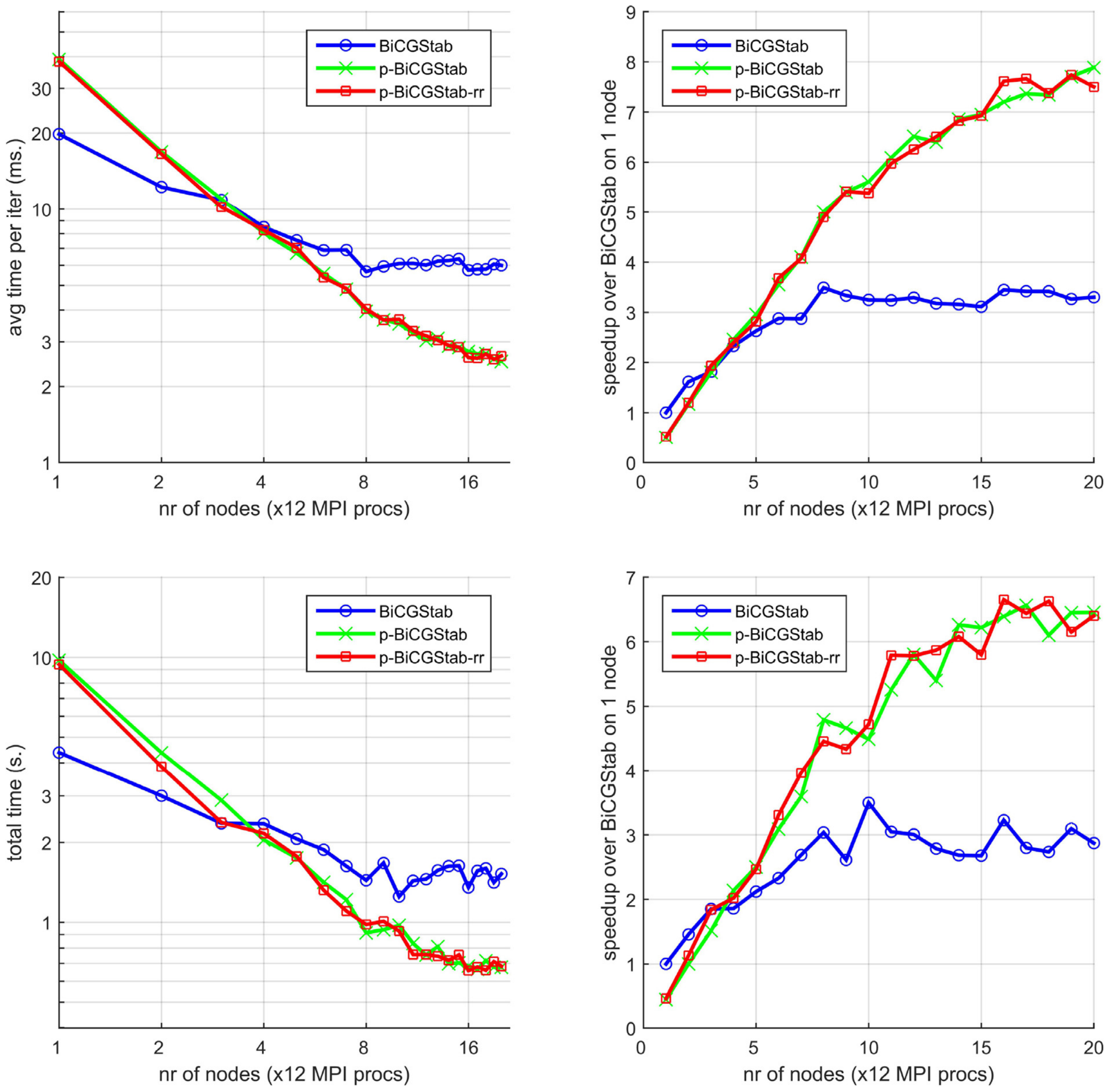
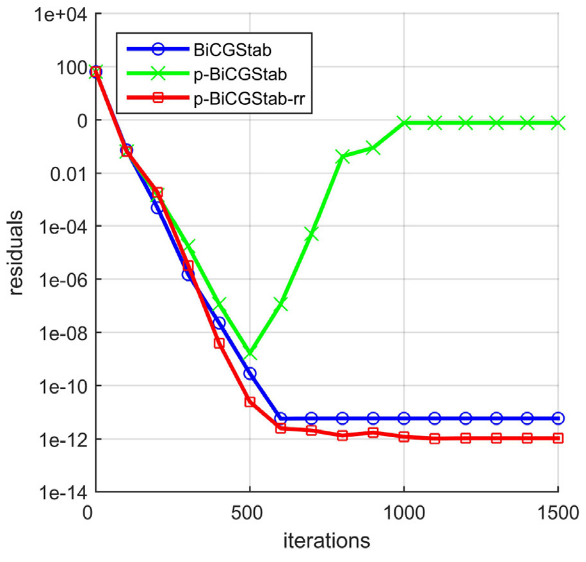


Fig. 3. **(PTP1)** Strong scaling experiment on up to 20 nodes (240 cores). Top left: Average time per iteration (`log10` scale) as function of the number of nodes (`log2` scale). Top right: Speedup (per iteration) over standard BiCGStab on a single node. Bottom left: Total CPU time as function of the number of nodes. Bottom right: Absolute speedup over standard BiCGStab on a single node. All methods were set to converge to a tolerance of 10^{-6} on the scaled residual, which was reached in 205 (min.) to 282 (max.) iterations. The p-BiCGStab-rr algorithm performs a replacement step every 100 iterations.

the number of nodes (bottom right), which display similar scaling behavior, albeit slightly less smooth due to the varying number of iterations between individual runs. The maximum speedup on 20 nodes over standard BiCGStab on a single node is 7.89× (computed on averaged timings). In contrast, for the model problem and hardware configuration in this benchmark experiment, the standard BiCGStab method stops scaling at around 8 nodes, obtaining a speedup of only 3.30 × on 20 nodes. Hence, pipelined BiCGStab attains a net speedup (per iteration) of 2.39× compared to standard CG when both are executed on 20 nodes, which approximates the theoretically optimal speedup of 2.5×, cf. Section 3.4. The absolute (non-averaged) speedup factor of p-BiCGStab (270 iterations) over standard BiCGStab (254 iterations) on 20 nodes is 2.25×. Performance results for the p-BiCGStab-rr method are similar to those of p-BiCGStab. The small computational overhead from the residual replacement steps does not affect strong scaling.

Fig. 4 shows the accuracy of the solution in function of the number of iterations (left) and in function of the total computational time (right) for the BiCGStab and p-BiCGStab algorithms on the 2D unsymmetric benchmark problem on a 20 node setup. Standard BiCGStab attains a maximal accuracy on the solution corresponding to a residual 2-norm of



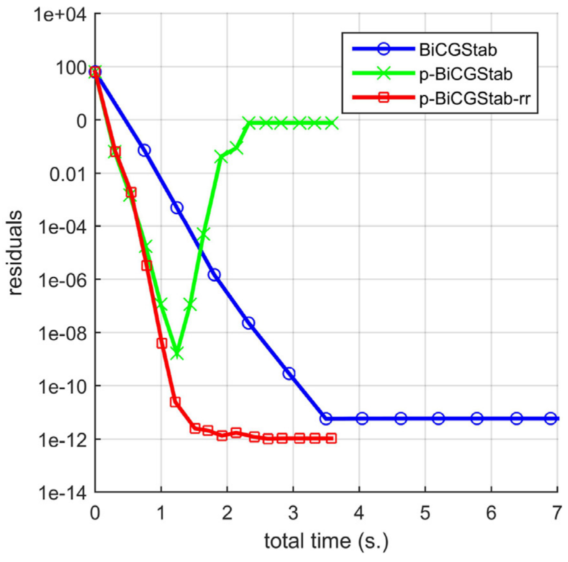


Fig. 4. **(PTP1)** Accuracy experiment on 20 nodes (240 cores). Left: True residual norm $\|b - Ax_i\|_2$ as function of iterations. Right: True residual norm as function of total time spent by the algorithm. Maximal number of iterations is 2000 for all methods. The p-BiCGStab-rr algorithm performs a replacement step every 100 iterations (max. 10 replacements).

Table 4
(PTP2) Reference table showing the iterations required to attain the scaled residual tolerance 1 e–6, corresponding to a true residual norm $\|b - Ax_i\|_2 \leq 3.0$ e–3, as a function of the number of nodes.

nodes	1	2	3	4	5	6	7	8	9	10
BiCGStab	1563	1805	1789	1875	1710	1852	1789	1555	1641	1909
p-BiCGStab	1807	1614	1779	1787	1673	1547	1668	1773	1640	1673
p-BiCGStab-rr	1857	1788	1728	1570	1677	1721	1688	1283	1884	1718
nodes	11	12	13	14	15	16	17	18	19	20
BiCGStab	1805	1715	1875	1717	1765	1722	1657	2050	1778	1670
p-BiCGStab	1936	1811	1629	1642	1849	1843	1726	1796	1713	1870
p-BiCGStab-rr	1845	1628	1562	1861	1650	1647	1889	1750	1701	2112

5.8 e–12 in 4.0 seconds. The pipelined variant is significantly faster, attaining a residual norm of 1.7 e–9 in only 1.2 seconds. However, for the pipelined method a higher accuracy is only obtainable by including the residual replacement strategy. The p-BiCGStab-rr method is able to attain a residual norm of 2.5 e–12 in 1.5 seconds, which is significantly faster than standard BiCGStab for a comparable accuracy.

Parallel test problem 2. The second benchmark problem used to asses parallel performance is a Helmholtz-type PDE model given by the 5-point stencil

$$A_{2\text{st}} = \begin{bmatrix} & -1 & \\ -1 & 1 & -1 \\ & -1 & \end{bmatrix}, \tag{PTP2}$$

and a right-hand side $b = A_2\hat{x}$, where again $\hat{x} = \mathbf{1}$. It can be considered as a 2D Poisson operator, shifted by a diagonal matrix that relates to the Helmholtz wave number. This operator is highly indefinite and notably hard to solve using iterative methods, see [22]. The one million unknowns system $A_2x = b$ is solved by unpreconditioned[4] BiCGStab, Algorithm 7, and its pipelined variant, Algorithm 9.

Fig. 5 shows timing (left) and speedup (right) results for test problem (PTP2) on one up to 20 nodes. The averaged speedup graph (top right) for p-BiCGStab is largely comparable to the results for (PTP1), showing good scaling on up to 20 nodes with a per-iteration speedup of 7.52× (non-averaged: 6.29×) compared to 1-node BiCGStab. Total timings are slightly more oscillating due to the significant differences in iterations between individual runs, see Table 4. The total time spent by the p-BiCGStab algorithm on 20 nodes is 9.8 s. (for 1670 iterations), whereas the p-BiCGStab algorithm requires only

[4] Most standard preconditioners based on e.g. incomplete LU factorization, multigrid methods or domain decomposition methods available in PETSc do not improve convergence for Parallel test problem 2. The authors are aware of the existence of specialized preconditioning techniques for Helmholtz-type problems, as proposed in e.g[19,21].. However, a discussion on the effectiveness of these preconditioners is beyond the scope of this work.

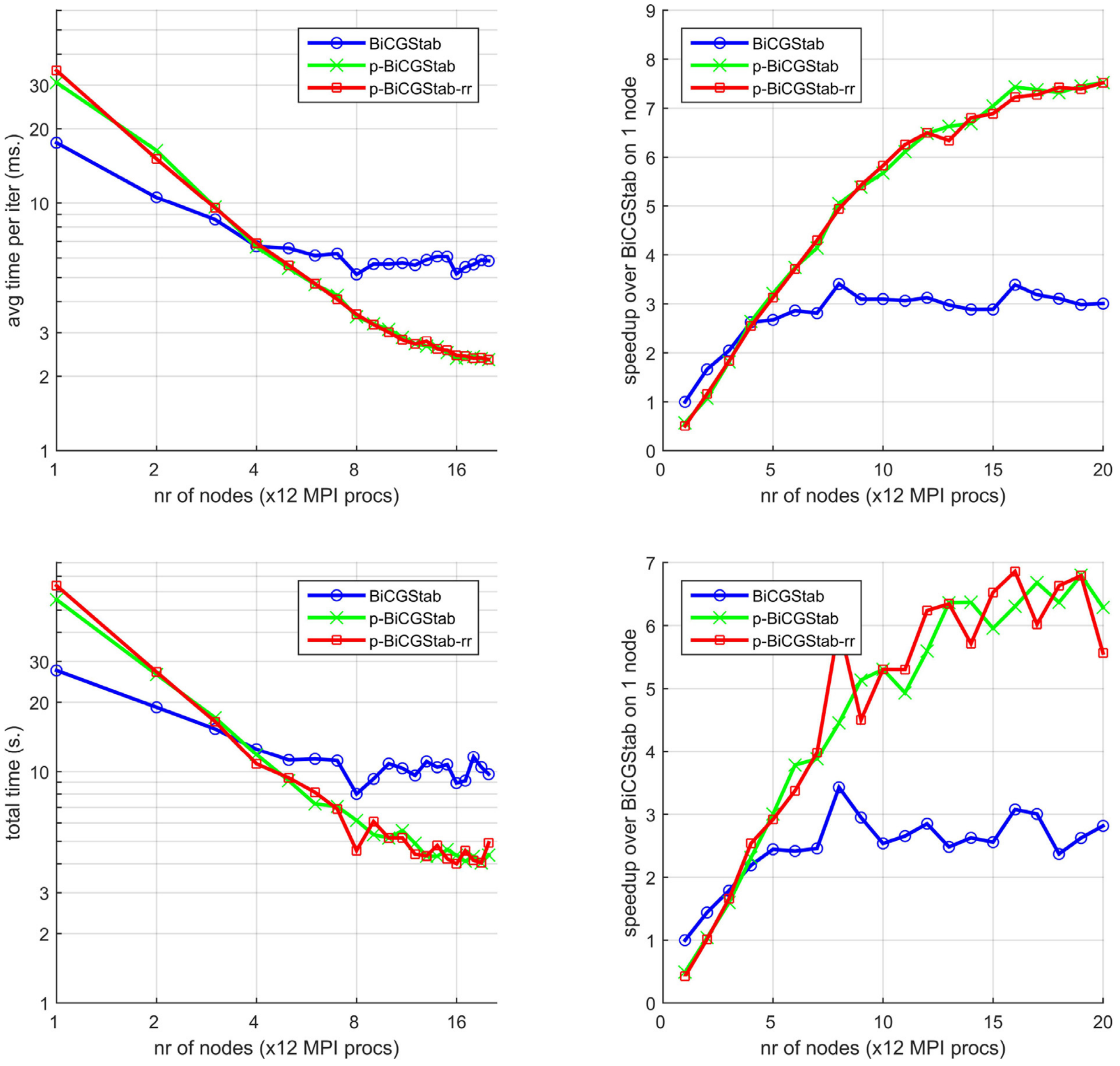


Fig. 5. **(PTP2)** Strong scaling experiment on up to 20 nodes (240 cores). Top left: Average time per iteration (`log10` scale) as function of the number of nodes (`log2` scale). Top right: Speedup (per iteration) over standard BiCGStab on a single node. Bottom left: Total CPU time as function of the number of nodes. Bottom right: Absolute speedup over standard BiCGStab on a single node. All methods converged to a scaled residual tolerance of 10^{-6}, which was reached in 1283 (min.) to 2112 (max.) iterations. The p-BiCGStab-rr algorithm performs a replacement step every 100 iterations (max. 10 replacements).

4.6 s. (for 1870 iterations) to attain the same accuracy. Hence, pipelining results in a net speedup factor of 2.23 $\times$ on 20 nodes for this model problem.

6. Conclusions

Pipelined algorithms (partially) circumvent the traditional global synchronization bottleneck in traditional Krylov subspace methods. As a consequence, they offer better scalability in the strong scaling limit for computing solutions to large and sparse linear systems on massively parallel hardware. In this work we proposed a general framework for the derivation of a pipelined variant of a given Krylov subspace method. The pipelining framework consists of two main phases. In the first step, denoted as *communication-avoiding*, the standard Krylov algorithm is rewritten into a mathematically equivalent algorithm with fewer global synchronization points. This is achieved by combining the global reduction phases of different dot-products scattered across the algorithm into one global communication phase. The second step, called *communication-hiding*, subsequently reformulates the algorithm such that the remaining global reduction phases are overlapped by the sparse matrix-vector product and preconditioner application. As such, the typical communication bottleneck is mitigated by hiding communication time behind useful computational work.

Applications of the proposed framework include the reformulation of several widely used Krylov subspace methods, such as the Conjugate Gradient method (CG) for symmetric and positive definite linear systems [25], and the Generalized Minimal Residual method (GMRES) [24] and Bi-Conjugate Gradient Stabilized method (BiCGStab) for the solution of general unsymmetric and/or indefinite systems. The proposed high-level framework can be used to derive a length-one pipelined version of any Krylov subspace method. The development of a general framework for the derivation of length-l pipelined methods is left as future work.

To illustrate the methodology, the pipelining framework is successfully applied to the BiCGStab method for the solution of large and sparse unsymmetric linear systems. The pipelined BiCGStab method (p-BiCGStab) reduces the number of global synchronization points from three to two, and overlaps the remaining global reduction phases with computational work. This induces a theoretical speed-up of up to 250% over traditional BiCGStab on a large number of nodes. Contrary to the so-called s-step methods [3–5,7–10], the combination of pipelined methods and preconditioning is straightforward, as is illustrated by the derivation of the preconditioned pipelined BiCGStab method in this work. However, to ensure optimal scaling the chosen preconditioner preferably requires only a limited amount of global communication.

Numerical experiments on a moderately sized cluster show that the p-BiCGStab method displays significantly increased parallel performance and improved strong scaling compared to standard BiCGStab on an increasing number of computational nodes. In practice a speedup of 2.0 to 2.5$\times$ over the traditional BiCGStab method can be expected when both are executed on the same number of parallel processors.

Finally, the experimental results point out two minor numerical drawbacks that originate from reordering the BiCGStab algorithm into a pipelined version. In the extremely small residual regime, a loss of maximal attainable accuracy can be expected, which is a typical phenomenon related to pipelined (and other communication-avoiding) Krylov subspace methods [11,25]. Furthermore, due to the introduction of additional AXPY operations by the pipelining framework, the p-BiCGStab algorithm is typically less robust with respect to numerical rounding errors compared to the standard algorithm. It is observed that the p-BiCGStab residuals are not guaranteed to remain at the same level after attaining the maximal accuracy, which is a highly unwanted feature. Both of these numerical issues are simultaneously resolved by including a residual replacement strategy [4,27,35,36,42] in the pipelined method. Indeed, through a periodical reset of the residual and auxiliary variables to their true values by explicitly computing the corresponding SPMVs, it is shown that both robustness and attainable accuracy can be restored to the original BiCGStab method's level at the expense of a moderate added computational cost.

Acknowledgments

This work is partially funded by the EXA2CT European Project on Exascale Algorithms and Advanced Computational Techniques, which receives funding from the EU's Seventh Framework Programme (FP7/2007-2013) under grant agreement no. 610741. Additionally, S. C. is funded by the Research Foundation Flanders (FWO). The authors are grateful to Bram Reps for his preliminary work on this topic, and to Pieter Ghysels and Patrick Sanan for their useful insights and remarks.

References

[1] S. Balay, S. Abhyankar, M.F. Adams, J. Brown, P. Brune, K. Buschelman, L. Dalcin, V. Eijkhout, W.D. Gropp, D. Kaushik, M.G. Knepley, L.C. McInnes, K. Rupp, B.F. Smith, S. Zampini, H. Zhang, PETSc Web page, 2015, (http://www.mcs.anl.gov/petsc).
[2] R. Barrett, M. Berry, T.F. Chan, J. Demmel, J. Donato, J. Dongarra, V. Eijkhout, R. Pozo, C. Romine, H.A. Van der Vorst, Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods, 2nd ed., SIAM, Philadelphia, 1994.
[3] E. Carson, Communication-avoiding Krylov Subspace Methods in Theory and Practice, U.C. Berkeley, EECS, 2015 Ph.D. thesis.
[4] E. Carson, J. Demmel, A residual replacement strategy for improving the maximum attainable accuracy of s-step krylov subspace methods, SIAM J. Matrix Anal. Appl. 35 (1) (2014) 22–43.
[5] E. Carson, N. Knight, J. Demmel, Avoiding communication in nonsymmetric Lanczos-based Krylov subspace methods, SIAM J. Sci. Comput. 35 (5) (2013) S42–S61.
[6] E. Chow, A. Patel, Fine–grained parallel incomplete LU factorization, SIAM J. Sci. Comput. 37 (2) (2015) C169–C193.
[7] A.T. Chronopoulos, s-Step iterative methods for (non)symmetric (in)definite linear systems, SIAM J. Numer. Anal. 28 (6) (1991) 1776–1789.
[8] A.T. Chronopoulos, C.W. Gear, s-Step iterative methods for symmetric linear systems, J. Comput. Appl. Math. 25 (2) (1989) 153–168.
[9] A.T. Chronopoulos, A.B. Kucherov, Block s-step Krylov iterative methods, Numerical Linear Algebra with Applications 17 (1) (2010) 3–15.
[10] A.T. Chronopoulos, C.D. Swanson, Parallel iterative s-step methods for unsymmetric linear systems, Parallel Comput. 22 (5) (1996) 623–641.
[11] S. Cools, W. Vanroose, E.F. Yetkin, E. Agullo, L. Giraud, On rounding error resilience, maximal attainable accuracy and parallel performance of the pipelined Conjugate Gradients method for larg e–scale linear systems in PETSc, in: Proceedings of the Exascale Applications and Software Conference 2016, ACM, 2016, pp. 20–29.
[12] E. De Sturler, A parallel variant of GMRES(m), in: Proceedings of the 13th IMACS World Congress on Computational and Applied Mathematics. IMACS, Criterion Press, 9, 1991.
[13] E. De Sturler, H.A. Van der Vorst, Reducing the effect of global communication in GMRES(m) and CG on parallel distributed memory computers, Appl. Numer. Math. 18 (4) (1995) 441–459.
[14] J.W. Demmel, Applied Numerical Linear Algebra, SIAM, 1997.
[15] J.W. Demmel, M.T. Heath, H.A. Van der Vorst, Parallel Numerical Linear Algebra, Acta Numerica 2 (1993) 111–197.
[16] J. Dongarra, M.A. Heroux, Toward a New Metric for Ranking High Performance Computing Systems, 312, 2013.
[17] J. Dongarra, M.A. Heroux, P. Luszczek, HPCG Benchmark: A New Metric for Ranking High Performance Computing Systems, 2015.
[18] P.R. Eller, W. Gropp, Non-blocking preconditioned conjugate gradient methods for extrem e–scale computing, in: Conference Proceedings 17th Copper Mountain Conference on Multigrid Methods, Colorado, US, 2015.
[19] B. Engquist, L. Ying, Sweeping preconditioner for the Helmholtz equation: moving perfectly matched layers, Multiscale Model. Simulation 9 (2) (2011) 686–710.

[20] J. Erhel, A parallel GMRES version for general sparse matrices, Electron. Trans. Numer. Anal. 3 (12) (1995) 160–176.
[21] Y.A. Erlangga, C. Vuik, C.W. Oosterlee, On a class of preconditioners for solving the Helmholtz equation, Appl. Numer. Math. 50 (3) (2004) 409–425.
[22] O.G. Ernst, M.J. Gander, Why it is difficult to solve Helmholtz problems with classical iterative methods, in: Numerical Analysis of Multiscale Problems, Springer, 2012, pp. 325–363.
[23] R. Fletcher, Conjugate gradient methods for indefinite systems, in: Numerical analysis, Springer, 1976, pp. 73–89.
[24] P. Ghysels, T.J. Ashby, K. Meerbergen, W. Vanroose, Hiding global communication latency in the GMRES algorithm on massively parallel machines, SIAM J. Sci. Comput. 35 (1) (2013) C48–C71.
[25] P. Ghysels, W. Vanroose, Hiding global synchronization latency in the preconditioned conjugate gradient algorithm, Parallel Comput. 40 (7) (2014) 224–238.
[26] A. Greenbaum, Behavior of slightly perturbed Lanczos and conjugat e–gradient recurrences, Linear Algebra Appl. 113 (1989) 7–63.
[27] A. Greenbaum, Estimating the attainable accuracy of recursively computed residual methods, SIAM J. Matrix Anal. Appl. 18 (3) (1997) 535–551.
[28] M.H. Gutknecht, Z. Strakoš, Accuracy of two thre e–term and three two-term recurrences for Krylov space solvers, SIAM J. Matrix Anal. Appl. 22 (1) (2000) 213–229.
[29] M.R. Hestenes, E. Stiefel, Methods of conjugate gradients for solving linear systems, 49, NBS, 1952.
[30] T. Jacques, L. Nicolas, C. Vollaire, Electromagnetic Scattering with the Boundary Integral Method on MIMD Systems, in: Parallel Numerical Computation with Applications, Springer, 1999, pp. 215–230.
[31] G. Meurant, Z. Strakoš, The Lanczos and conjugate gradient algorithms in finite precision arithmetic, Acta Numerica 15 (2006) 471–542.
[32] C.C. Paige, Error analysis of the Lanczos algorithm for tridiagonalizing a symmetric matrix, IMA J. Appl. Math. 18 (3) (1976) 341–349.
[33] C.C. Paige, Accuracy and effectiveness of the Lanczos algorithm for the symmetric eigenproblem, Linear Algebra Appl. 34 (1980) 235–258.
[34] P. Sanan, S.M. Schnepp, D.A. May, Pipelined, flexible Krylov subspace methods, SIAM J. Sci. Comput. 38 (5) (2016) C441–C470.
[35] G.L.G. Sleijpen, H.A. Van der Vorst, Reliable updated residuals in hybrid bi-CG methods, Computing 56 (2) (1996) 141–163.
[36] G.L.G. Sleijpen, H.A. Van der Vorst, J. Modersitzki, Differences in the effects of rounding errors in Krylov solvers for symmetric indefinite linear systems, SIAM J. Matrix Anal. Appl. 22 (3) (2001) 726–751.
[37] B. Smith, P. Bjorstad, W.D. Gropp, Domain Decomposition: Parallel Multilevel Methods for Elliptic Partial Differential Equations, Cambridge University Press, 2004.
[38] P. Sonneveld, CGS, a fast Lanczos-type solver for nonsymmetric linear systems, SIAM J. Sci. Stat. Comput. 10 (1) (1989) 36–52.
[39] Z. Strakoš, P. Tichỳ, On error estimation in the conjugate gradient method and why it works in finite precision computations, Electron. Trans. Numer. Anal. 13 (2002) 56–80.
[40] C. Tong, Q. Ye, Analysis of the finite precision bi-conjugate gradient algorithm for nonsymmetric linear systems, Math. Comput. 69 (232) (2000) 1559–1575.
[41] H.A. Van der Vorst, Bi-CGSTAB: a fast and smoothly converging variant of bi-CG for the solution of nonsymmetric linear systems, SIAM J. Sci. Stat. Comput. 13 (2) (1992) 631–644.
[42] H.A. Van der Vorst, Q. Ye, Residual replacement strategies for krylov subspace iterative methods for the convergence of true residuals, SIAM J. Sci. Comput. 22 (3) (2000) 835–852.
[43] L.T. Yang, The improved CGS method for large and sparse linear systems on bulk synchronous parallel architectures, in: Proceedings of the Fifth International Conference on Algorithms and Architectures for Parallel Processing, IEEE, 2002, pp. 232–237.
[44] L.T. Yang, R.P. Brent, The improved BiCGStab method for large and sparse unsymmetric linear systems on parallel distributed memory architectures, in: Proceedings of the Fifth International Conference on Algorithms and Architectures for Parallel Processing, IEEE, 2002, pp. 324–328.
[45] L.T. Yang, R.P. Brent, The improved parallel BiCG method for large and sparse unsymmetric linear systems on distributed memory architectures, in: Proceedings of the 16th International Parallel and Distributed Processing Symposium IPDPS 2002, IEEE, 2003, pp. 349–360.