

## ANALYSIS AND PRACTICAL USE OF FLEXIBLE BICGSTAB

JIE CHEN\*, LOIS CURFMAN MCINNES\*, AND HONG ZHANG\*

**Abstract.** A flexible version of the BiCGStab algorithm (FBiCGStab) for solving a linear system of equations is analyzed. We show that under flexible preconditioning, the perturbation to the outer residual norm is of the same order as that to the preconditioner. In addition, we formulate a mathematically equivalent variant, FBiCGStab-R, in order to reduce the global synchronization cost for performing inner product calculations. FBiCGStab-R is numerically stable and sometimes far outperforms FBiCGStab if high accuracy of the solution is required. Both analysis and numerical experiments show that a strong preconditioner is often favored for the practical use of flexible BiCGStab. We demonstrate in a large-scale reacting flow application (PFLOTRAN) that the use of flexible BiCGStab leads to significantly accelerated simulation time on extreme-scale computers using  $O(10^4)$ – $O(10^5)$  processor cores.

**Key words.** BiCGStab, flexible preconditioning

**AMS subject classifications.** 65F10

**1. Introduction.** Flexible iterative methods [9, 14, 15, 26, 28, 38, 42] for solving a linear system of equations are referred to as preconditioned Krylov methods where the preconditioner may change across iterations. The flexible preconditioning strategy is also known under various terms such as *inexact*, *nonlinear*, or *variable preconditioning*. A representative scenario is that the preconditioning requires a linear solve with a second iterative method, in which case “inner iterations” are used to mean preconditioning and “outer iterations” are used to mean the flexible Krylov method itself. Flexible methods are an important class of methods that offer several advantages over the use of a fixed preconditioner, one of which is the flexibility to balance the accuracy of the preconditioning solves and the convergence of the outer Krylov iterations in order to reduce the total computational cost. Furthermore, in large-scale applications, the changing landscape of both scientific needs (complex physical models and couplings) and emerging extreme-scale computing systems gives rise to practical preconditioners that are hierarchical or nested [5, 6, 10, 16, 20, 31]. Many of these emerging preconditioners benefit from inexact inner solves and thus encourage the use of flexible Krylov methods.

Among many proposed flexible methods, flexible GMRES (abbreviated as FGMRES [28]) is the most frequently used in practice. Its wide use is probably linked to the robustness that results from the long-term recurrence and the global orthogonality. Compared with standard GMRES [30], even though the traditional notion of a Krylov subspace is lost, FGMRES computes an orthonormal basis of a subspace within which an optimal residual is sought. Hence, FGMRES still enjoys the residual norm minimization property, and it often shows a satisfactory convergence behavior. On the other hand, in other flexible methods with short-term recurrences, such as inexact PCG [15], flexible QMR [38] and flexible BiCG [42], the global (bi)orthogonality is lost, and the convergence behavior is generally unpredictable unless the inner solves are sufficiently accurate that orthogonality is nearly preserved. An idea to improve the robustness is to explicitly perform the orthogonalization as proposed for a variant of the flexible CG algorithm [26]. On the other hand, several analyses of flexible

---

\*Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, IL 60439. Emails: (jiechen, mcinnes, hzhang)mcs.anl.gov

methods, using a larger Krylov subspace that includes the Arnoldi vectors, indicate that the convergence behavior can be maintained with respect to the fixed preconditioning case if the perturbation to the preconditioner grows inversely with the current residual norm [11, 32, 33].

BiCGStab [39] is a widely used Krylov method. In many applications, BiCGStab outperforms GMRES in terms of both solution time and memory usage, and it becomes the de facto method of choice for practitioners. Although BiCGStab is akin to BiCG [12], which generates two sets of biorthogonal residual vectors that naturally form two associated Krylov subspaces, the convergence behavior of BiCGStab is harder to describe because the residual vectors alone do not span the Krylov subspace that contains them. BiCGStab can be understood as a member of a family of induced dimension reduction (IDR) methods whereby the generated residuals belong to a nested sequence of subspaces that are recursively defined [34, 35, 36].

Motivated by an important reacting flow application PFLOTRAN [18, 19], for which BiCGStab has been predominantly used as the linear solver, we study in this paper the flexibly preconditioned BiCGStab algorithm (abbreviated as FBiCGStab) as an effort to improve the solution time. Detailed information about PFLOTRAN will be introduced in Section 5. FBiCGStab was initially proposed in [42], and it was recently cast under the framework of flexible variants of IDR methods [40]. However, little is known about convergence properties. The goal of this paper is to analyze the behavior of the method and to provide guidance on its practical use. We do not study the convergence guarantee of the method, but we argue that the convergence behavior is close to that of the fixed preconditioning case if the perturbation to the preconditioner is relatively small. Because of the loose connection of BiCGStab with the associated Krylov subspace, this analysis differs from that for other flexible Krylov methods (see [11, 32, 33]). Rather, we adopt a geometric argument on the iterates and show that the perturbation to the relative residual norm of the outer iterations is of the same order as that of the inner iterations. Interestingly, this result can lead to a similar conclusion as that in [11, 32, 33]; that is, the convergence behavior of the flexible method can be maintained by relaxing the accuracy requirement of the preconditioning solves as the outer residual norm decreases.

The efficiency of parallelism is of particular importance for solving large-scale linear systems on high-performance computers. Often the computation of inner products and norms constitutes the bottleneck of iterative solvers, because they require global synchronizations on distributed-memory machines [2, 7, 13, 24, 37, 41, 43]. The scaling of the solver starts to deteriorate when the number of processor cores increases beyond  $O(10,000)$ . Thus, we formulate an alternative version of FBiCGStab, namely, FBiCGStab-R, by grouping the inner product calculations in order to reduce the number of global synchronizations. This mathematically equivalent variant will be useful on emerging extreme-scale computers when the cost of computing inner products is high, for example, when a perfect load balance is difficult to achieve among processors and thus a fair amount of time is spent on the synchronization for performing collective operations. Empirical study shows that FBiCGStab-R is numerically stable and sometimes performs far better than FBiCGStab if high accuracy of the solution is sought. Similar efforts have been made on BiCGStab (the fixed preconditioning case), and numerical stability was seen within floating-point precision [8, 44], but we demonstrate that FBiCGStab-R is stable up to double precision.

The rest of the paper is organized as follows. With a brief derivation of FBiCGStab, Section 2 analyzes the behavior of the residual norm under flexible preconditioning.

Section 3 presents FBiCGStab-R and discusses its numerical behavior. Several numerical examples are shown in Section 4 to demonstrate the advantage of using a strong preconditioner. Under this guideline, Section 5 presents the successful use of FBiCGStab with multigrid preconditioners in PFLOTRAN. Concluding remarks are given in Section 6.

**2. Algorithm and analysis.** The following is the standard unpreconditioned BiCGStab algorithm for solving a linear system

$$Ax = b,$$

using  $x_0$  as the initial guess [29].

```

1:  $r_0 = b - Ax_0$ ;  $\bar{r}_0$  arbitrary
2:  $p_0 = r_0$ 
3: for  $j = 0, 1, \dots$  until convergence do
4:    $\alpha_j = (r_j, \bar{r}_0) / (Ap_j, \bar{r}_0)$ 
5:    $s_j = r_j - \alpha_j Ap_j$ 
6:    $\omega_j = (As_j, s_j) / (As_j, As_j)$ 
7:    $x_{j+1} = x_j + \alpha_j p_j + \omega_j s_j$ 
8:    $r_{j+1} = s_j - \omega_j As_j$ 
9:    $\beta_j = (r_{j+1}, \bar{r}_0) / (r_j, \bar{r}_0) \cdot \alpha_j / \omega_j$ 
10:   $p_{j+1} = r_{j+1} + \beta_j (p_j - \omega_j Ap_j)$ 
11: end for

```

The coefficient iterates  $\alpha_j$  and  $\beta_j$  are derived based on their counterparts in BiCG for updating the residual vectors and the search direction vectors. One can show that  $\alpha_j$  makes  $s_j \perp \bar{r}_0$  and  $\beta_j$  makes  $p_{j+1} \perp \bar{r}_0$  for all  $j$ . Furthermore,  $\omega_j$  is defined to minimize the 2-norm of the residual vector  $r_{j+1}$ .

When the algorithm is used with a preconditioner  $M \approx A$ , the right preconditioning is equivalent to solving the system

$$AM^{-1}y = b, \quad y = Mx.$$

One way to derive the preconditioned iteration is, in the above unpreconditioned version, to replace the symbol  $A$  by  $AM^{-1}$  and  $x_j$  by  $y_j$ , and then substitute  $y_j$  back by  $Mx_j$ . This introduces two auxiliary vectors  $\tilde{p}_j = M^{-1}p_j$  and  $\tilde{s}_j = M^{-1}s_j$ , which are the only computations that require the preconditioner. We summarize this preconditioned version in Algorithm 1. It is the same as the one presented in [42].

One would also like to consider left preconditioning, where  $M^{-1}$  is applied to the left of the system  $Ax = b$ . After a change of variables, the left preconditioned version is almost the same as Algorithm 1, except that the two inner products in line 8 are changed to the ones using the  $(MM^T)^{-1}$ -norm. In this case,  $\omega_j$  minimizes the  $(MM^T)^{-1}$ -norm of  $r_{j+1}$ . Compared with right preconditioning, left preconditioning incurs two more applications of the preconditioner in each iteration, thus increasing the computational cost. Therefore, we do not consider left preconditioning in this paper.

An iterative method can be used to compute  $\tilde{p}_j$  in line 4 and  $\tilde{s}_j$  in line 7 of Algorithm 1, but the iterations may not run to full accuracy. In this case, Algorithm 1 becomes the flexible version of BiCGStab. The computed iterates  $\tilde{p}_j$  and  $\tilde{s}_j$  under inexact preconditioning will carry on their error to subsequent iterations. To gauge

**Algorithm 1** Right preconditioned BiCGStab / Flexible BiCGStab

---

```

1:  $r_0 = b - Ax_0$ ;  $\bar{r}_0$  arbitrary
2:  $p_0 = r_0$ 
3: for  $j = 0, 1, \dots$  until convergence do
4:    $\tilde{p}_j = M^{-1}p_j$ 
5:    $\alpha_j = (r_j, \bar{r}_0)/(A\tilde{p}_j, \bar{r}_0)$ 
6:    $s_j = r_j - \alpha_j A\tilde{p}_j$ 
7:    $\tilde{s}_j = M^{-1}s_j$ 
8:    $\omega_j = (A\tilde{s}_j, s_j)/(A\tilde{s}_j, A\tilde{s}_j)$ 
9:    $x_{j+1} = x_j + \alpha_j \tilde{p}_j + \omega_j \tilde{s}_j$ 
10:   $r_{j+1} = s_j - \omega_j A\tilde{s}_j$ 
11:   $\beta_j = (r_{j+1}, \bar{r}_0)/(r_j, \bar{r}_0) \cdot \alpha_j/\omega_j$ 
12:   $p_{j+1} = r_{j+1} + \beta_j(p_j - \omega_j A\tilde{p}_j)$ 
13: end for

```

---

the amplification of error, we are interested in the situation that the relative residual of the inner solves with  $M$  is upper bounded by a small tolerance  $\epsilon$ . That is, if we use an underline to denote the actual iterates with errors, we assume that

$$\|\underline{p}_j - M\underline{\tilde{p}}_j\| \leq \epsilon \|\underline{p}_j\| \quad \text{and} \quad \|\underline{s}_j - M\underline{\tilde{s}}_j\| \leq \epsilon \|\underline{s}_j\|. \quad (2.1)$$

In the following, we characterize the relative difference between  $r_{j+1}$  and  $\underline{r}_{j+1}$  under Condition (2.1).

The analysis is based on the fact that the coefficient iterates  $\alpha_j$ ,  $\beta_j$ , and  $\omega_j$  are computed such that the inaccuracy incurred in the preconditioning solves is not ‘‘adversely’’ accumulated to affect outer iterations. For this, we need the following observations. They are trivially correct in the fixed preconditioned case but are also true when a flexible preconditioner is used. The proof is simple and thus omitted.

**PROPOSITION 2.1.** *The iterates in Algorithm 1 have the following properties:*

- (i) *The vector  $r_{j+1}$  is the residual; that is,  $r_{j+1} = b - Ax_{j+1}$ .*
- (ii) *Consider that  $s_j$  is a function of  $\alpha_j$ ; then the definition of  $\alpha_j$  in line 5 makes  $s_j \perp \bar{r}_0$ .*
- (iii) *Consider that  $p_{j+1}$  is a function of  $\beta_j$ ; then the definition of  $\beta_j$  in line 11 makes  $p_{j+1} \perp \bar{r}_0$ .*
- (iv) *Consider that  $r_{j+1}$  is a function of  $\omega_j$ ; then the definition of  $\omega_j$  in line 8 minimizes  $\|r_{j+1}\|_2$ .*

In light of these observations, we have the following two lemmas. They imply that the perturbation to the vector  $x - \alpha y$ , where the scalar  $\alpha$  is used to satisfy some orthogonality or minimization property, is of the same order as the perturbations to the vectors  $x$  and  $y$ . We use  $\angle(x, y)$  to denote the *acute* angle between the two vectors; that is,  $|(x, y)| = \|x\|\|y\|\cos\angle(x, y)$ . Hence,  $\cos\angle(x, y)$  is always nonnegative.

**LEMMA 2.2.** *Given a vector  $r$ , let  $z = x - \alpha y$  and  $\underline{z} = \underline{x} - \underline{\alpha}y$ , where  $\alpha = (x, r)/(y, r)$  and  $\underline{\alpha} = (\underline{x}, r)/(\underline{y}, r)$ . If there exist  $\epsilon_x, \epsilon_y$  such that  $\epsilon_y < \cos\angle(y, r)$  and that*

$$\|x - \underline{x}\| \leq \epsilon_x \|x\| \quad \text{and} \quad \|y - \underline{y}\| \leq \epsilon_y \|y\|,$$

then  $\|z - \underline{z}\| \leq \epsilon_z \|z\|$  with

$$\epsilon_z = \frac{\epsilon_x + \sqrt{\epsilon_x^2 + \left[ \epsilon_x \frac{\sin^2(\theta_y + \angle(y, r))}{\cos(\theta_y + \angle(y, r))} + \epsilon_y \frac{\cos \angle(x, r)}{\cos \angle(y, r) [\sqrt{1 - \epsilon_y^2} \cos \angle(y, r) - \epsilon_y \sin \angle(y, r)]} \right]^2}}{\sin \angle(x, y)}, \quad (2.2)$$

where  $\theta_y = \arcsin \epsilon_y$ . Furthermore, denote  $\epsilon = \max\{\epsilon_x, \epsilon_y\}$ . If  $\angle(x, y)$  and  $\angle(x, r)$  are lower bounded by  $\theta$  and  $\phi$ , respectively, and  $\angle(y, r)$  is upper bounded by  $\psi$ , then

$$\epsilon_z \leq C(\theta, \phi, \psi, \epsilon) \cdot \epsilon \quad \text{with} \quad \lim_{\epsilon \rightarrow 0} C(\theta, \phi, \psi, \epsilon) = \frac{1 + \sqrt{1 + \left[ \frac{\sin^2 \psi}{\cos \psi} + \frac{\cos \phi}{\cos^2 \psi} \right]^2}}{\sin \theta}, \quad (2.3)$$

where  $C$  is defined based on the expression of  $\epsilon_z$  in (2.2), with  $\angle(x, y)$ ,  $\angle(x, r)$ ,  $\angle(y, r)$ ,  $\epsilon_x$ , and  $\epsilon_y$  being replaced by  $\theta$ ,  $\phi$ ,  $\psi$ ,  $\epsilon$ , and  $\epsilon$ , respectively, and then being divided by  $\epsilon$ .

*Proof.* First consider the triangle defined by  $z$ ,  $x$  and  $\alpha y$ . Clearly,

$$\|z\| \geq \|x\| \sin \angle(x, y). \quad (2.4)$$

Then consider

$$\|z - \underline{z}\| \leq \|x - \underline{x}\| + \|\alpha y - \underline{\alpha y}\| \quad \text{with} \quad \|x - \underline{x}\| \leq \epsilon_x \|x\|. \quad (2.5)$$

We proceed to bound the norm of  $w = \alpha y - \underline{\alpha y}$ . Let  $w$  be split into two orthogonal components,  $w_{\parallel r}$  and  $w_{\perp r}$ , where the former is parallel to  $r$  and the latter orthogonal to  $r$ . Then

$$\|\alpha y - \underline{\alpha y}\|^2 = \|w\|^2 = \|w_{\parallel r}\|^2 + \|w_{\perp r}\|^2. \quad (2.6)$$

Since both  $z$  and  $\underline{z}$  are orthogonal to  $r$ , performing an inner product with  $r$  on both sides of  $w = x - \underline{x} + z - \underline{z}$  yields  $(w, r) = (x - \underline{x}, r)$ . Then,

$$\|w_{\parallel r}\| = \frac{|(w, r)|}{\|r\|} = \frac{|(x - \underline{x}, r)|}{\|r\|} \leq \|x - \underline{x}\| \leq \epsilon_x \|x\|. \quad (2.7)$$

To find an upper bound for  $\|w_{\perp r}\|$ , we use a geometric argument; see Figure 2.1. Although the figure shows 3D geometry, the argument applies to vectors of any dimension. The plane  $P$ , passing through the origin, represents a space containing all vectors orthogonal to  $r$ . The markers  $E$ ,  $G'$ ,  $D$ ,  $B$ ,  $C$ , and  $G$  all lie on  $P$ . The line  $AE$  is orthogonal to  $P$ , thus parallel to  $r$ . Imagine that the vector  $x$  starts from the origin and ends at  $A$ . The vector  $y$  then starts from  $A$  and points to the direction represented by  $\overrightarrow{AB}$ . Therefore, the vector  $-\alpha y$  ends at  $B$  because  $z = x - \alpha y$  lie on  $P$ . Hence,  $\overrightarrow{EB}$  represents  $-(\alpha y)_{\perp r}$ . Because  $\|x - \underline{x}\| \leq \epsilon_x \|x\|$ , the vector  $\underline{x}$  also starts from the origin but can end anywhere within  $\epsilon_x \|x\|$  distance to  $A$ , indicated by the dashed circle. Because  $\|y - \underline{y}\| \leq \epsilon_y \|y\|$ , the vector  $\underline{y}$ , if translated to start from  $A$ , can point to any direction within the cone  $\angle CAD$  that has a half angle  $\theta_y = \arcsin \epsilon_y$ . The lines  $FG$  and  $F'G'$  are tangent to the circle and parallel to  $AC$  and  $AD$ , respectively. Thus,  $\overrightarrow{FG}$  and  $\overrightarrow{F'G'}$  are two examples of the vector  $-\underline{\alpha y}$ . Hence, the maximum length



and

$$\epsilon_z = (\epsilon_x + \epsilon_y)(1 + \epsilon_x) + \frac{\epsilon_x + \epsilon_y \cos \angle(x, y) + (\epsilon_x + \epsilon_y) \cos \angle(x, y) \left[ 1 + \frac{(1 + \epsilon_x)(\epsilon_x + \epsilon_y)}{2\sqrt{1 - (\epsilon_x + \epsilon_y)^2}} \right]}{\sin \angle(x, y)}. \quad (2.10)$$

Furthermore, denote  $\epsilon = \max\{\epsilon_x, \epsilon_y\}$ . If  $\angle(x, y)$  is lower bounded by  $\theta_l$  and upper bounded by  $\theta_u$ , then

$$\epsilon_\alpha \leq C_\alpha(\theta_u, \epsilon) \cdot \epsilon \quad \text{with} \quad \lim_{\epsilon \rightarrow 0} C_\alpha(\theta, \epsilon) = 2 + 2 \tan \theta \quad (2.11)$$

and

$$\epsilon_z \leq C_z(\theta_l, \epsilon) \cdot \epsilon \quad \text{with} \quad \lim_{\epsilon \rightarrow 0} C_z(\theta, \epsilon) = \left( 2 + \frac{1 + 3 \cos \theta}{\sin \theta} \right), \quad (2.12)$$

where  $C_\alpha$  is defined based on the expression of  $\epsilon_\alpha$  in (2.9), with  $\angle(x, y)$ ,  $\epsilon_x$ , and  $\epsilon_y$  being replaced by  $\theta$ ,  $\epsilon$ , and  $\epsilon$ , respectively, and then being divided by  $\epsilon$ . The same is used to define  $C_z$  based on (2.10).

*Proof.* First we have

$$\begin{aligned} \|z - \underline{z}\| &\leq \|x - \underline{x}\| + \|\alpha y - \underline{\alpha} y\| \\ &\leq \|x - \underline{x}\| + \|\alpha(y - \underline{y})\| + \|(\alpha - \underline{\alpha})\underline{y}\| \\ &\leq \epsilon_x \|x\| + \epsilon_y \|\alpha y\| + \|(\alpha - \underline{\alpha})\underline{y}\|. \end{aligned} \quad (2.13)$$

Because  $\alpha$  and  $\underline{\alpha}$  have the same sign, we obtain

$$\frac{\|(\alpha - \underline{\alpha})\underline{y}\|}{\|\alpha y\|} = \left| \frac{\|\underline{y}\|}{\|y\|} - \frac{\|\underline{\alpha} y\|}{\|\alpha y\|} \right|.$$

Then, using the fact that  $|\|\underline{y}\|/\|y\| - 1| \leq \epsilon_y$ , we get

$$\|(\alpha - \underline{\alpha})\underline{y}\| \leq \|\alpha y\| \left( \epsilon_y + \left| 1 - \frac{\|\underline{\alpha} y\|}{\|\alpha y\|} \right| \right) \quad (2.14)$$

and

$$\frac{|\alpha - \underline{\alpha}|}{|\alpha|} \leq \frac{\|\underline{y}\|}{\|y\|} \left( \epsilon_y + \left| 1 - \frac{\|\underline{\alpha} y\|}{\|\alpha y\|} \right| \right) \leq (1 + \epsilon_y) \left( \epsilon_y + \left| 1 - \frac{\|\underline{\alpha} y\|}{\|\alpha y\|} \right| \right). \quad (2.15)$$

Because  $z \perp y$  and  $\underline{z} \perp \underline{y}$ , the following equalities are useful:

$$\|z\| = \|x\| \sin \angle(x, y), \quad \|\alpha y\| = \|x\| \cos \angle(x, y), \quad (2.16)$$

and similarly for the underlined counterparts. Hence, we proceed to bound

$$\left| 1 - \frac{\|\underline{\alpha} y\|}{\|\alpha y\|} \right| = \left| 1 - \frac{\|\underline{x}\| \cos \angle(x, y)}{\|x\| \cos \angle(x, y)} \right|.$$

To simplify notation, let  $\angle(x, y) = \beta$  and  $\angle(\underline{x}, \underline{y}) = \beta + \delta$  for some  $\delta$ . Then

$$\begin{aligned} \left| 1 - \frac{\cos(\beta + \delta)}{\cos \beta} \right| &= |(1 - \cos \delta) + \sin \delta \tan \beta| \\ &= \left| (\sin \delta) \left( \tan \frac{\delta}{2} + \tan \beta \right) \right| \leq |\sin \delta| \left( \frac{1}{2} |\tan \delta| + \tan \beta \right). \end{aligned}$$

Let  $\theta_x$  be the angle between  $x$  and  $\underline{x}$ , and similarly for  $\theta_y$ . Because  $\|x - \underline{x}\| \leq \epsilon_x \|x\|$  with  $\epsilon_x < 1$ ,  $\theta_x$  is acute. Similarly, so is  $\theta_y$ . Note that  $\|x\| \sin \theta_x \leq \|x - \underline{x}\| \leq \epsilon_x \|x\|$ ; therefore,  $\sin \theta_x \leq \epsilon_x$ , and similarly  $\sin \theta_y \leq \epsilon_y$ . Then,  $\sin \theta_x \leq \epsilon_x < \sqrt{1 - \epsilon_y^2} \leq \cos \theta_y$ , which indicates that  $\theta_x + \theta_y$  is also acute. Thus, the fact that  $|\delta| \leq \theta_x + \theta_y$  leads to  $\sin |\delta| \leq \sin \theta_x + \sin \theta_y \leq \epsilon_x + \epsilon_y$ . Therefore

$$\left| 1 - \frac{\cos(\beta + \delta)}{\cos \beta} \right| \leq (\epsilon_x + \epsilon_y) \left[ \frac{\epsilon_x + \epsilon_y}{2\sqrt{1 - (\epsilon_x + \epsilon_y)^2}} + \tan \angle(x, y) \right] =: A.$$

By noting that

$$\left| 1 - \frac{\|\underline{x}\|}{\|x\|} \right| \leq \epsilon_x =: B,$$

we obtain

$$\left| 1 - \frac{\|\underline{x}\| \cos \angle(\underline{x}, \underline{y})}{\|x\| \cos \angle(x, y)} \right| \leq B + A + AB,$$

which gives

$$\left| 1 - \frac{\|\underline{\alpha y}\|}{\|\alpha y\|} \right| \leq \epsilon_x + (1 + \epsilon_x)(\epsilon_x + \epsilon_y) \left[ \frac{\epsilon_x + \epsilon_y}{2\sqrt{1 - (\epsilon_x + \epsilon_y)^2}} + \tan \angle(x, y) \right]. \quad (2.17)$$

Combining (2.15) and (2.17) gives (2.9); combining (2.13), (2.14), (2.16), and (2.17) gives (2.10). The results (2.11) and (2.12) follow from the fact that  $\epsilon_\alpha$  is an increasing function of  $\angle(x, y)$  and  $\epsilon_z$  is a decreasing function of  $\angle(x, y)$ .  $\square$

Using the above two lemmas, we have the following result. It states that the relative perturbation to the outer residual norm is of the same order as the relative residual norm in the inner solves.

**THEOREM 2.4.** *If for all  $j$  in any finite number of outer iterations where breakdown does not occur,  $\angle(r_j, \bar{r}_0) \neq 0$ ,  $\angle(r_j, A\tilde{p}_j) \neq 0$ ,  $\angle(r_{j+1}, p_j - \omega_j A\tilde{p}_j) \neq 0$  and  $\angle(s_j, A\tilde{s}_j) \neq 0$ , then under Condition (2.1),*

$$\frac{\|r_j - \underline{r}_j\|}{\|r_j\|} = O(\epsilon).$$

*Proof.* To facilitate presentation, we define  $\text{err}(a) := \|a - \underline{a}\|/\|a\|$  for any vector or scalar  $a$ . We first observe that

$$\begin{aligned} \frac{\|A\tilde{p}_j - A\underline{\tilde{p}}_j\|}{\|A\tilde{p}_j\|} &= \frac{\|AM^{-1}(p_j - M\underline{\tilde{p}}_j)\|}{\|AM^{-1}p_j\|} \leq \kappa \frac{\|p_j - M\underline{\tilde{p}}_j\|}{\|p_j\|} \\ &\leq \kappa \left( \frac{\|p_j - \underline{p}_j\|}{\|p_j\|} + \frac{\|\underline{p}_j - M\underline{\tilde{p}}_j\|}{\|p_j\|} \right) \leq \kappa \left( \frac{\|p_j - \underline{p}_j\|}{\|p_j\|} + \epsilon \frac{\|\underline{p}_j\|}{\|p_j\|} \right), \end{aligned}$$

where  $\kappa$  denotes the condition number of  $AM^{-1}$ . Since the big-O notation is used for sufficiently small  $\epsilon$ , the above observation means that if  $\text{err}(p_j) = O(\epsilon)$ , then  $\text{err}(A\tilde{p}_j) = O(\epsilon)$ . Similarly, if  $\text{err}(s_j) = O(\epsilon)$ , then  $\text{err}(A\tilde{s}_j) = O(\epsilon)$ .

We now show the theorem by induction on  $\text{err}(r_j)$  and  $\text{err}(p_j)$ , based on the two preceding lemmas. The conditions in the theorem are used to ensure the applicability

of the lemmas. At  $j = 0$ ,  $r_0$  and  $p_0$  is unchanged under variable preconditioning. If  $\text{err}(r_j) = O(\epsilon)$  and  $\text{err}(p_j) = O(\epsilon)$ , then because  $\text{err}(A\tilde{p}_j) = O(\epsilon)$ , we have  $\text{err}(s_j) = O(\epsilon)$  by Lemma 2.2. Consequently,  $\text{err}(A\tilde{s}_j) = O(\epsilon)$ , and thus  $\text{err}(r_{j+1}) = O(\epsilon)$  by Lemma 2.3.

Now consider  $p_{j+1} = r_{j+1} + \beta_j z_j$ , where  $z_j = p_j - \omega_j A\tilde{p}_j$ . Both  $\text{err}(p_j)$  and  $\text{err}(A\tilde{p}_j)$  are  $O(\epsilon)$ . On the other hand,  $\text{err}(\omega_j)$  is also  $O(\epsilon)$  according to Lemma 2.3, because we have shown that both  $\text{err}(s_j)$  and  $\text{err}(A\tilde{s}_j)$  are  $O(\epsilon)$ . Hence,  $\text{err}(z_j) = O(\epsilon)$ . Then by invoking Lemma 2.2 again we have  $\text{err}(p_{j+1}) = O(\epsilon)$ , which completes the induction.  $\square$

Theorem 2.4 considers the perturbation of the residual in the relative sense. As a corollary, a result for the absolute perturbation is given next. Instead of a fixed tolerance  $\epsilon$  for all the inner solves, we allow the tolerance, denoted by  $\epsilon_j$ , to vary in each outer iteration  $j$ . The result indicates a reciprocal relationship between the residual norm  $\|r_j\|$  and  $\epsilon_j$ .

**COROLLARY 2.5.** *Let the relative inner tolerance  $\epsilon$  depend on the outer iterations indexed by  $j$ , that is,*

$$\|p_j - M\tilde{p}_j\| \leq \epsilon_j \|p_j\| \quad \text{and} \quad \|s_j - M\tilde{s}_j\| \leq \epsilon_j \|s_j\|.$$

*Under the condition of Theorem 2.4, if the residual norm  $\|r_j\|$  is monotonically decreasing, then for any  $\delta$  there exists a constant  $C$  such that if*

$$\epsilon_j = \frac{C\delta}{\|r_j\|},$$

*then  $\|r_j - r_j\| \leq \delta$ .*

*Proof.* Note that Theorem 2.4 is proved by induction on  $j$ . When  $\epsilon_j$  is monotonically increasing and when  $\epsilon_j$  is sufficiently small, a stronger conclusion is that there exists a  $C'$  that is independent of  $j$  such that

$$\text{err}(r_j) \leq C'\epsilon_j, \tag{2.18}$$

because in the right-hand side  $\epsilon_j$  can always be relaxed later by changing it to  $\epsilon_{j+1}$ . Rewriting (2.18), we obtain  $\|r_j - r_j\| \leq C'\epsilon_j \|r_j\|$ . Therefore, if we let  $\epsilon_j = C\delta/\|r_j\|$  by using some  $C$  such that  $C'C \leq 1$  and that  $C\delta/\|r_j\|$  is sufficiently small to trigger the validity of (2.18), we immediately have  $\|r_j - r_j\| \leq \delta$ .  $\square$

**3. A mathematically equivalent variant.** Each iteration of Algorithm 1 requires five inner product calculations:  $(A\tilde{p}_j, \bar{r}_0)$ ,  $(A\tilde{s}_j, \tilde{s}_j)$ ,  $(A\tilde{s}_j, A\tilde{s}_j)$ ,  $(r_{j+1}, \bar{r}_0)$ , and the hidden  $\|r_{j+1}\|$  for convergence tests. A straightforward parallel implementation (using MPI [25], for example) will incur four calls of `MPI_Allreduce` to sum the local inner products (the two inner products  $(A\tilde{s}_j, \tilde{s}_j)$  and  $(A\tilde{s}_j, A\tilde{s}_j)$  are obviously computed together and thus require only one `MPI_Allreduce`). Because inner product calculations are expensive on distributed-memory machines with a large number of processors, it is desirable that the calculations be grouped together while the numerical behavior of the iterations is maintained.

Following this guideline, we rearrange the computation of  $\beta_j$  and  $\|r_{j+1}\|$  so that they can be computed together with  $\omega_j$ . Because  $s_j \perp \bar{r}_0$ , line 10 of Algorithm 1 gives

$$(r_{j+1}, \bar{r}_0) = -\omega_j (A\tilde{s}_j, \bar{r}_0). \tag{3.1}$$

Then, together with the definition of  $\alpha_j$  in line 5, we obtain

$$\beta_j = -(A\tilde{s}_j, \bar{r}_0)/(A\tilde{p}_j, \bar{r}_0).$$

This makes the computation of  $\beta_j$  independent of  $r_{j+1}$ , and thus it can be moved ahead before  $r_{j+1}$  is available. Next, consider the residual norm  $\rho_{j+1} = \|r_{j+1}\|$ . Because

$$(r_{j+1}, r_{j+1}) = (s_j, s_j) - 2\omega_j(A\tilde{s}_j, s_j) + \omega_j^2(A\tilde{s}_j, A\tilde{s}_j) = (s_j, s_j) - \omega_j(A\tilde{s}_j, s_j),$$

the reliance of the computation of  $\rho_{j+1}$  on  $r_{j+1}$  can also be eliminated. Thus,  $\rho_{j+1}$  can be computed immediately after  $\beta_j$ .

To obtain cleaner notation, we define  $t_j = A\tilde{s}_j$ , and  $v_j = A\tilde{p}_j$ . Then, the discussed modifications lead to Algorithm 2. For clarity, we insert the convergence test in line 14 to show the use of  $\rho_{j+1}$ . Because of the simple but equivalent rearrangement of the iterate updates, this alternative version is equivalent to Algorithm 1 in exact arithmetic, as the following proposition states.

PROPOSITION 3.1. *Algorithm 2 is mathematically equivalent to Algorithm 1.*

---

**Algorithm 2** FBiCGStab-R: Flexible BiCGStab with reduced synchronizations

---

```

1:  $r_0 = b - Ax_0$ ;  $\bar{r}_0$  arbitrary
2:  $p_0 = r_0$ 
3: for  $j = 0, 1, \dots$  until convergence do
4:    $\tilde{p}_j = K^{-1}p_j$ ,  $v_j = A\tilde{p}_j$ 
5:   compute  $(r_j, \bar{r}_0)$ ,  $(v_j, \bar{r}_0)$ 
6:     then  $\alpha_j = (r_j, \bar{r}_0)/(v_j, \bar{r}_0)$ 
7:    $s_j = r_j - \alpha_j v_j$ 
8:    $\tilde{s}_j = K^{-1}s_j$ ,  $t_j = A\tilde{s}_j$ 
9:   compute  $(s_j, s_j)$ ,  $(t_j, s_j)$ ,  $(t_j, t_j)$ ,  $(t_j, \bar{r}_0)$ 
10:    then  $\omega_j = (t_j, s_j)/(t_j, t_j)$ 
11:       $\beta_j = -(t_j, \bar{r}_0)/(v_j, \bar{r}_0)$ 
12:       $\rho_{j+1} = [(s_j, s_j) - \omega_j(t_j, s_j)]^{1/2}$ 
13:    $x_{j+1} = x_j + \alpha_j \tilde{p}_j + \omega_j \tilde{s}_j$ 
14:   if  $\rho_{j+1} < tol$  return
15:    $r_{j+1} = s_j - \omega_j t_j$ 
16:    $p_{j+1} = r_{j+1} + \beta_j(p_j - \omega_j v_j)$ 
17: end for

```

---

Compared with Algorithm 1, Algorithm 2 requires the same number of matrix-vector multiplications and preconditioning steps. The difference is that the number of calls to `MPI_Allreduce` is reduced to two per iteration (lines 5 and 9), at the expense of computing one more inner product. On distributed-memory machines with a large number of processors, the cost of the extra inner product calculation is likely to be compensated by the saving in the reduced number of global synchronizations. Thus, the alternative version can be useful when the latency cost of synchronization is high. For example, a perfect load balance is difficult to achieve for unstructured meshes. Then, the matrix-vector multiplications can have a large variance in finishing time, and this contributes to the nonnegligible latency in `MPI_Allreduce` for inner product calculations. Another situation is the solution of multiphysics problems where each subproblem is solved independently but simultaneously. The concurrency of the processors may be affected by the imbalanced time cost of handling each subproblem. Reducing the number of synchronizations is thus helpful in these scenarios.

As is well known, when the iterations are rearranged, the numerical behavior of an iterative algorithm may change even if the rearrangement is done in a mathematically

equivalent way. Based on Algorithm 2, one may want to further reduce the number of inner product calculations and the calls to `MPI_Allreduce`, but it is easy to result in an algorithm that is unstable. We consider one seemingly natural, but in fact dangerous, modification here. From the relation (3.1), one can eliminate the inner product  $(r_j, \bar{r}_0)$  for computing  $\alpha_j$ :

$$\alpha_j = -\omega_{j-1}(t_{j-1}, \bar{r}_0)/(v_j, \bar{r}_0), \quad (3.2)$$

because both  $\omega_{j-1}$  and  $(t_{j-1}, \bar{r}_0)$  have been computed in the previous iteration. However, this modification is numerically unstable. A later experiment shows that in Algorithm 2 the residual norm  $\|r_{j+1}\|$  is able to decrease to machine precision, but using (3.2) makes  $\|r_{j+1}\|$  stagnate at a certain point. We speculate that the iterations are sensitive to the numerical orthogonality between  $s_j$  and  $\bar{r}_0$ . In both Algorithms 1 and 2, orthogonality is ensured because of the straightforward computation of  $\alpha_j$ ; however, when computing  $\alpha_j$  using (3.2) the loss of orthogonality is accumulated because the computation of  $(r_{j+1}, \bar{r}_0)$  uses only the part  $(-\omega_j t_j, \bar{r}_0)$  but misses the part  $(s_j, \bar{r}_0)$ . When the loss of orthogonality reaches the level of the current residual norm, the residual is unable to further decrease.

Algorithms 1 and 2 (and even the unstable version using (3.2)) behave almost the same at the beginning, because the differences of the residuals across different versions are orders of magnitude smaller than the residual itself. Only when the residual decreases to a certain level do their behaviors become distinguishable. Nevertheless, the difference usually does not affect the convergence trend. Sometimes, however, Algorithm 2 works surprisingly much better than Algorithm 1, as shown in a later experiment. The reason is unclear, but this behavior implies that even outside the regime of parallel computing, Algorithm 2 can be a useful alternative to Algorithm 1 when one seeks a solver of optimal performance.

**4. Numerical examples.** To empirically study the effectiveness of flexible preconditioning, we consider the linear system arising from a discretization of the PDE (adopted from [28]):

$$-\Delta u + \gamma \mathbf{x} \cdot \nabla u + \beta u = f \quad (4.1)$$

with zero Dirichlet boundary condition. The linear system can be made indefinite and/or unsymmetric by changing the parameters  $\gamma$  and  $\beta$ . In this study we discretized the domain into a regular grid of size  $n_1 \times n_2 \times n_3$  with spacing  $h = 1/n_1$  and set  $\gamma = 4/h$  to make the problem unsymmetric. We varied the parameter  $\beta$  to include indefinite cases, and thus we experimented with problems of different levels of difficulty. The right-hand side was chosen to be the vector of all ones, with the initial guess being zero. As a common practice in parallel solvers, if no specific preconditioner is mentioned, an iterative method (including the case of being used for inner iterations) was always preconditioned by block Jacobi/ILU(0), where each block was handled by one processor. The experiments were conducted on the supercomputer Jaguar (introduced in the next section).

We tested with several  $\beta$  values; Table 4.1 shows the results of three representative cases. As  $\beta$  decreases, the problem becomes harder and harder to solve. In the first case,  $\beta = 0.01/h^2$ , the system is positive definite; but in the next two cases,  $\beta = -0.4/h^2$  and  $-0.6/h^2$ , the system is indefinite. Figure 4.1 shows the convergence history of BiCGStab and GMRES. These results were obtained on a  $256 \times 256 \times 256$  grid using 16,384 processors. The restart cycle for GMRES was 30. The residual tolerance and the maximum number of iterations were  $1\mathbf{e-8}$  and 200, respectively.

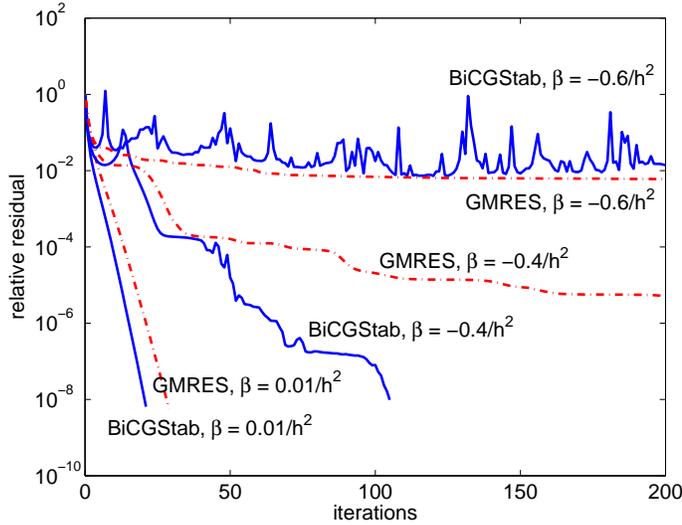


FIG. 4.1. Convergence of BiCGStab and GMRES for three  $\beta$ 's (no inner iterations).

To be compared are BiCGStab, FBiCGStab/BiCGStab, GMRES, and FGMRES/GMRES, where FBiCGStab/BiCGStab means FBiCGStab is preconditioned by BiCGStab, and similarly for FGMRES/GMRES. The implementation of FBiCGStab was according to Algorithm 1. In both of the flexible methods, the stopping criterion for the inner iterations was either a residual tolerance or a fixed number of iterations. The two major comparison metrics are the number of matrix-vector multiplications (MatMult, column 4) and the number of calls to `MPI_Allreduce` (Allreduce, column 5). The number of floating-point operations (Flops, column 6) for computing the inner products is listed for reference but should not be used for comparison because (i) the synchronization cost is much higher than that of floating-point operations and (ii) GMRES requires long term recurrence (orthogonalization) whereas BiCGStab does not. The wall-clock time is also listed, but note that the fluctuations in the use of a supercomputer and in the communication latency are factors that affect the actual running time.

Several important observations are made based on the data in Table 4.1 and other experiments with intermediate  $\beta$  values that are not shown in the table. First, for this set of test cases, the GMRES family in general performs better when the system is relatively easy to solve; but as the difficulty level increases, the BiCGStab family significantly outperforms the GMRES family. One sees that in the first case GMRES is the best solver, whereas in the second case it failed to converge to the required tolerance but BiCGStab did converge. In the second case, however, the fastest solver is still in the GMRES family. Nevertheless, when moving to the third case, FBiCGStab with an inner tolerance stopping criterion clearly wins over all other solvers. This interesting phenomenon indicates that even for the same problem, different solvers may have a significantly different performance profile as the problem parameters vary.

Second, for a flexible method, using a fixed number of inner iterations can sometimes achieve excellent performance; but as the problem becomes harder and harder, it is difficult to specify an appropriate number a priori to ensure the convergence of the outer iterations. One sees that in the first case, using a fixed number of inner it-

TABLE 4.1

*Solution summary of (4.1) with three choices of  $\beta$ . Arrows point to the fastest run.*

$\beta = 0.01/h^2$	Outer	Time	MatMult	Inner Product		
	Iter.	(sec $\times 10^{-1}$ )		Allreduce	Flops $\times 10^9$	
BiCGStab	21	1.132	42	84	3.010	
FBiCGStab, inner rtol = 1e-3	2	1.397	58	120	4.246	
FBiCGStab, inner rtol = 1e-2	2	1.335	44	92	3.233	
FBiCGStab, inner rtol = 1e-1	4	1.364	54	116	4.037	
FBiCGStab, inner max.it = 5	2	1.131	44	92	3.233	
FBiCGStab, inner max.it = 10	1	1.147	40	82	2.910	
FBiCGStab, inner max.it = 20	1	1.226	46	94	3.334	
GMRES	29	0.874	29	58	9.494	←
FGMRES, inner rtol = 1e-3	3	1.122	40	83	6.081	
FGMRES, inner rtol = 1e-2	4	7.200	37	78	4.240	
FGMRES, inner rtol = 1e-1	7	1.153	38	83	3.292	
FGMRES, inner max.it = 5	6	1.072	36	78	3.130	
FGMRES, inner max.it = 10	3	1.066	33	69	4.237	
FGMRES, inner max.it = 20	2	1.065	40	82	8.720	
$\beta = -0.4/h^2$		$\times 10^0$			$\times 10^6$	
BiCGStab	105	0.356	210	420	1.505	
FBiCGStab, inner rtol = 1e-3	2	0.668	438	880	3.144	
FBiCGStab, inner rtol = 1e-2	2	0.499	270	544	1.948	
FBiCGStab, inner rtol = 1e-1	6	0.832	586	1184	4.225	
FBiCGStab, inner max.it = 5	fail	-	-	-	-	
FBiCGStab, inner max.it = 10	fail	-	-	-	-	
FBiCGStab, inner max.it = 20	25	2.517	2050	4150	14.800	
GMRES	fail	-	-	-	-	
FGMRES, inner rtol = 1e-3	3	1.556	1165	2298	37.890	
FGMRES, inner rtol = 1e-2	4	1.062	803	1586	25.550	
FGMRES, inner rtol = 1e-1	7	0.791	560	1113	17.160	
FGMRES, inner max.it = 5	26	0.282	156	338	1.893	
FGMRES, inner max.it = 10	14	0.278	154	322	2.134	←
FGMRES, inner max.it = 20	12	0.373	252	516	5.861	
$\beta = -0.6/h^2$		$\times 10^1$			$\times 10^7$	
BiCGStab	fail	-	-	-	-	
FBiCGStab, inner rtol = 1e-3	2	0.951	8314	16632	5.962	
FBiCGStab, inner rtol = 1e-2	2	0.572	5024	10052	3.605	←
FBiCGStab, inner rtol = 1e-1	15	3.564	30120	60270	21.560	
FBiCGStab, inner max.it = 5	fail	-	-	-	-	
FBiCGStab, inner max.it = 10	fail	-	-	-	-	
FBiCGStab, inner max.it = 20	fail	-	-	-	-	
GMRES	fail	-	-	-	-	
FGMRES, inner rtol = 1e-3	5	5.291	51670	101680	169.600	
FGMRES, inner rtol = 1e-2	5	4.589	41800	82259	136.560	
FGMRES, inner rtol = 1e-1	7	3.903	37757	74305	123.730	
FGMRES, inner max.it = 5	fail	-	-	-	-	
FGMRES, inner max.it = 10	fail	-	-	-	-	
FGMRES, inner max.it = 20	fail	-	-	-	-	

erations as the stopping criterion is in general preferable to using a residual tolerance. This is also true in the second case for the FGMRES solvers. For the FBiCGStab solvers, however, the situation is completely opposite. Moving to the third case, none of the solvers using a fixed number of inner iterations converged. In this sense, setting an inner tolerance is a more robust practice.

Third, it is possible to choose an “optimal” inner tolerance for a flexible method. One sees that for FBiCGStab the inner tolerance  $1e-2$  yields the best results in all the cases, whereas for FGMRES the tolerance is  $1e-1$ . This is consistent with the observation made in the experiments of flexible QMR [38], which states that the total solver cost first decreases, then increases as the inner solves are more and more exact. The “optimal” inner tolerance may be related to the convergence behavior of the inner iterations. One sees that in Figure 4.1 the relative residual norm of BiCGStab has a step decrease at the beginning, until between  $1e-1$  and  $1e-2$ . This may be the stopping point when BiCGStab becomes the most effective as an inner solve.

Fourth, the inner-outer iterations (that is, FBiCGStab/BiCGStab and FGMRES/GMRES) are often a better alternative to the standard iterations (that is, BiCGStab and GMRES). In harder problems the standard iterations did not converge but inner-outer iterations did. In fact, the outer iterations converge extremely fast when setting an appropriate inner tolerance. This behavior means that the preconditioner is strong even though it is inexact.

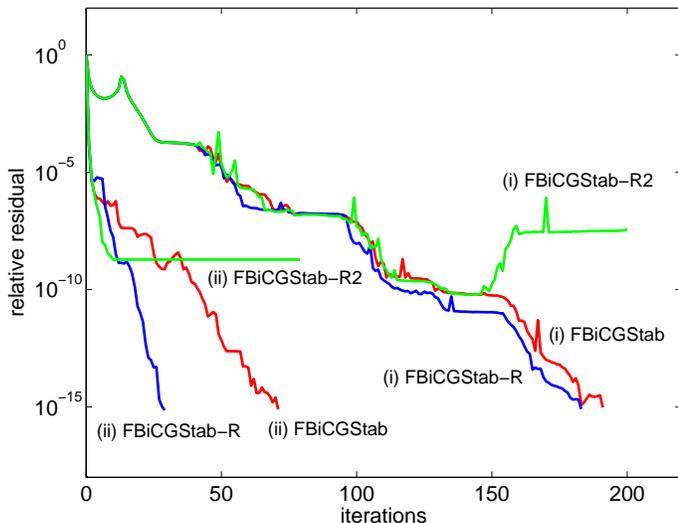


FIG. 4.2. Convergence history of FBiCGStab (Algorithm 1), FBiCGStab-R (Algorithm 2), and FBiCGStab-R2 (a modification of FBiCGStab-R using (3.2) to update  $\alpha_j$ ).

We next study the numerical behavior of Algorithm 2. Figure 4.2 shows an example with  $\beta = -0.4/h^2$ . We experimented with two preconditioners: (i) fixed: the default block Jacobi/ILU(0), and (ii) flexible: 20 BiCGStab iterations. The plot includes the convergence history of FBiCGStab (Algorithm 1), FBiCGStab-R (Algorithm 2), and a modification of FBiCGStab-R (using (3.2) to update  $\alpha_j$ ). To facilitate presentation, we call the one with modification FBiCGStab-R2. One sees that the residuals for the three versions were the same at the beginning, until they dropped to approximately  $1e-4$  to  $1e-5$ . Afterwards, the trends started to vary. FBiCGStab and

FBiCGStab-R were able to converge, but FBiCGStab-R2 stagnated. Furthermore, in case (i) the convergence behaviors for FBiCGStab and FBiCGStab-R were similar, whereas in case (ii) FBiCGStab-R converged significantly faster. Though not shown, we verified that the residual norm  $\rho_{j+1}$  computed using line 12 of Algorithm 2 was the same as that computed straightforwardly as  $(r_{j+1}, r_{j+1})^{1/2}$ . Thus, the plotted histories are reliable. The reason for the surprisingly different convergence behaviors in case (ii) is unclear.

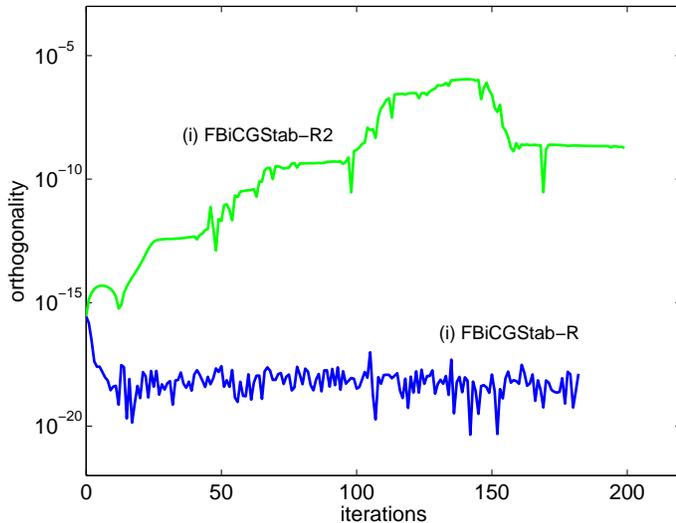


FIG. 4.3. Orthogonality of  $s_j$  and  $\bar{r}_0$ .

To further understand the numerical instability caused by (3.2), we plot in Figure 4.3 the absolute value of the inner product  $(s_j, \bar{r}_0)$  normalized by  $\|s_j\|\|\bar{r}_0\|$ . Only the result of using preconditioner (i) is shown. One sees that in FBiCGStab-R the normalized inner product is always below machine precision, whereas in FBiCGStab-R the normalized inner product shows a trend of increase. In this trend, the loss of orthogonality was accumulated and prevented the further decrease of  $r_{j+1}$  at some point.

**5. Application.** In this section we demonstrate the use of FBiCGStab in the application PFLOTRAN [18, 19, 21, 22, 23]. PFLOTRAN is a state-of-the-art code for simulating multiscale, multiphase, multicomponent flow and reactive transport in geologic media. PFLOTRAN solves a coupled system of mass and energy conservation equations for a number of phases, including air, water, and supercritical  $\text{CO}_2$  and a number of chemical components. The code utilizes finite-volume or mimetic finite-difference spatial discretizations combined with backward-Euler (fully implicit) timestepping for the flow and reactive transport solves or, optionally, operator splitting for the reactive transport. PFLOTRAN is built on the PETSc library [3, 4] and makes extensive use of PETSc iterative nonlinear and linear solvers.

We present numerical results for two benchmark problems used in [20]. The governing equations are described by Richards' equation:

$$\frac{\partial}{\partial t}(\varphi s \rho) + \nabla \cdot \rho \mathbf{u} = \mathcal{S},$$

where  $\varphi$  denotes the porosity of the geologic medium,  $s$  the saturation (fraction of pore volume filled with liquid water),  $\rho$  the fluid density,  $\mathcal{S}$  a source/sink term representing water injection/extraction, and  $\mathbf{u}$  the Darcy velocity defined as

$$\mathbf{u} = -\frac{\kappa\kappa_r}{\mu}\nabla(P - \rho gz),$$

where  $P$  denotes fluid pressure,  $\mu$  viscosity,  $\kappa$  the absolute permeability of the medium,  $\kappa_r$  the relative permeability of water to air,  $g$  the acceleration of gravity, and  $z$  the vertical distance from a datum. The two benchmark problems are as follows.

**Case 1:** Cubic domain with a central injection well; 6 time steps. This case models a  $100\text{m} \times 100\text{m} \times 100\text{m}$  domain with a uniform effective permeability of 1 darcy and an injection well at the exact center.

**Case 2:** Regional flow without well near river; 2 time steps. This case models a  $5000\text{m} \times 2500\text{m} \times 100\text{m}$  region with a river at the eastern boundary.

In our numerical experiments, we ran each test case for a minimum number of time steps required to obtain a reasonable picture of the basic physics.

BiCGStab has been the preferred linear solver for PFLOTRAN because of its small memory consumption (compared with that of GMRES) and the empirically fast convergence. Similar to other Krylov methods, BiCGStab encounters a well-known scaling difficulty for over 10,000 processor cores because of the bottleneck in the synchronization of vector inner product calculations. In order to overcome this difficulty, an improved variant of BiCGStab, namely, IBiCGStab [44], reduces the five global synchronization points per iteration to one through algorithmic reorganization of the BiCGStab iterations. Recently, in [20], the number of global inner products was significantly reduced by using Chebyshev iterations as the preconditioner, because Chebyshev requires no inner product calculations. As a convention, we use IBiCGStab/Chebyshev to denote this combination of the outer iterations and the preconditioner. An objective of this work is to further improve PFLOTRAN performance when using an extremely large number of processor cores. FBiCGStab enables us to explore a wide range of preconditioners, in particular the flexible ones, to improve over the results obtained in [20].

The analysis and discussion in preceding sections suggest that in order to maintain the convergence of FBiCGStab, using a strong preconditioner is preferable; that is, each application of the preconditioner achieves a good inner-residual reduction. This observation directed us to examine a set of strong preconditioners commonly used in PFLOTRAN.

Among all preconditioners in our experiments, the multigrid (MG) preconditioner with carefully selected smoother and coarse-level solver is found to perform best. Denoted by FBiCGStab/MG, the computational time of this combination improves over that of IBiCGStab/Chebyshev (reported in [20]) by approximately 30% to 60%.

For steady-state or close to steady-state problems, such as the two cases of PFLOTRAN we consider here, the multigrid preconditioner is known to work well on a small number of processors, but the performance starts tailing off at around 1,000 processor cores [17]. The difficulty in scaling up the number of processors is that in the coarsest level the problem is so small that the communication cost outweighs the computational cost. Here, we cope with this difficulty by (1) limiting the number of multigrid levels such that each processor core maintains sufficient workload at the coarsest level, and (2) using an iterative method that employs no or only a small number of inner product calculations in the coarsest level. Two such coarse-grid solvers we used are Chebyshev and IBiCGStab/Chebyshev. In the former the iteration matrix does not

change; thus we call it a fixed MG preconditioner. We call the latter a variable MG preconditioner.

Experiments were conducted on two computer systems: Intrepid, an IBM Blue Gene/P supercomputer located at the Argonne Leadership Computing Facility [1], and Jaguar, a Cray XK6 system located at the Oak Ridge Leadership Computing Facility [27]. Intrepid has a highly scalable torus network, as well as a high-performance collective network that minimizes the bottlenecks common in simulations on large, parallel computers. The current system has 40,960 nodes, each consisting of one 850 MHz quad-core processor and 2GB RAM, resulting in a total of 163,840 cores, 80TB of memory, and a peak performance of 557 TFlops. Jaguar has 18,688 compute nodes, each consisting of one AMD 16-core Opteron 6274 processor running at 2.2GHz and 32GB of memory, giving a total of 299,008 cores and a peak performance of 2,628 TFlops.

TABLE 5.1  
*BiCGStab and FBiCGStab for PFLOTRAN on Intrepid (IBM Blue Gene/P), Case 1.*

Number of Cores (Mesh Size)	IBiCGS/Cheby		BiCGStab/MG Smoother: Cheby CSolve: Cheby		FBiCGStab/MG Smoother: Cheby CSolve: IBiCGS/Cheby	
	Iter.	Time	Iter.	Time	Iter.	Time (% Reduction)
512 (256x256x256)	547	212.8	29	97.2	23	86.4 (11%)
4,096 (512x512x512)	1006	365.1	43	121.3	33	106.1 (12%)
32,768 (1024x1024x1024)	1886	654.3	62	153.7	37	119.1 (23%)
163,840 (1600x1600x640)	2843	308.3	88	81.8	53	65.7 (20%)

TABLE 5.2  
*BiCGStab and FBiCGStab for PFLOTRAN on Intrepid (IBM Blue Gene/P), Case 2.*

Number of Cores (Mesh Size)	IBiCGS/Cheby		BiCGStab/MG Smoother: Cheby CSolve: Cheby		FBiCGStab/MG Smoother: Cheby CSolve: IBiCGS/Cheby	
	Iter.	Time	Iter.	Time	Iter.	Time (% Reduction)
16,384 (1600x816x320)	844	231.1	33	64.3	22	52.6 (18%)
98,304 (1600x1632x640)	1520	270.5	61	70.0	39	54.7 (22%)
163,840 (1600x1632x640)	1499	169.3	62	52.0	36	40.2 (23%)

Tables 5.1 through 5.4 compare the performance of (1) IBiCGStab/Chebyshev, (2) BiCGStab with the fixed MG preconditioner, and (3) FBiCGStab with the variable MG preconditioner on the two benchmark cases and the two supercomputing systems. The results of IBiCGStab/Chebyshev (columns 2–3) have been reported in [20] and are used here as the baseline of comparison. For MG preconditioners (columns 4–8), we used V-cycles and set the maximum levels at 3. All the smoothers (before restriction and after interpolation) were 2 steps of Chebyshev iterations. The difference between

TABLE 5.3  
*BiCGStab and FBiCGStab for PFLOTRAN on Jaguar (Cray XK6), Case 1.*

Number of Cores (Mesh Size)	IBiCGS/Cheby		BiCGStab/MG Smoother: Cheby CSolve: Cheby		FBiCGStab/MG Smoother: Cheby CSolve: IBiCGS/Cheby	
	Iter.	Time	Iter.	Time	Iter.	Time (% Reduction)
512 (256x256x256)	546	24.2	29	16.8	23	12.7 (24%)
4,096 (512x512x512)	1033	44.1	43	17.4	33	16.0 (8%)
32,768 (1024x1024x1024)	2073	89.0	62	32.8	37	30.3 (8%)
160,000 (1600x1600x640)	2407	52.0	91	24.9	55	22.5 (10%)

TABLE 5.4  
*BiCGStab and FBiCGStab for PFLOTRAN on Jaguar (Cray XK6), Case 2.*

Number of Cores (Mesh Size)	IBiCGS/Cheby		BiCGStab/MG Smoother: Cheby CSolve: Cheby		FBiCGStab/MG Smoother: Cheby CSolve: IBiCGS/Cheby	
	Iter.	Time	Iter.	Time	Iter.	Time (% Reduction)
1,600 (800x408x160)	411	18.0	20	8.8	20	8.8 (0%)
16,000 (1600x816x320)	829	29.6	30	11.3	22	10.9 (4%)
80,000 (1600x1632x640)	1578	53.1	60	16.7	38	15.0 (10%)
224,000 (1600x1632x640)	1501	20.9	66	16.6	37	14.8 (11%)

the fixed MG and variable MG preconditioners is the coarse-grid solver: 100 Chebyshev iterations (CSolve: Cheby, columns 4–5) versus 5 IBiCGStab iterations, each preconditioned by 20 Chebyshev iterations (CSolve: IBiCGS/Cheby, columns 6–8). As usual, whenever a Chebyshev iteration is employed, block Jacobi/ILU(0) is applied as the innermost preconditioner. In our experiments, Chebyshev outperformed other commonly used smoothers (such as SOR). The reason may be that Chebyshev uses a small number of GMRES iterations to approximate extreme eigenvalues during the solver setup phase, which makes Chebyshev more effective in damping error modes at various grid levels and results in a more effective MG preconditioner.

Overall, on both machines, (F)BiCGStab with MG preconditioners is shown to be two to three times faster than IBiCGStab/Chebyshev on a large number of processor cores. Thus, the focus of our comparison here is how much reduction in execution time one can achieve by using a flexible preconditioner compared with using a fixed one. The percentage of reduction in execution time relative to the fixed MG preconditioner is listed in column 8 (% Reduction). As the size of the coarsest grid increases, the fixed number of iterations used for the coarsest-level solver weakens the MG preconditioner, resulting in an increased number of outer iterations. Such an increase is less significant for the variable MG preconditioner because each of its applications employs 5 IBiCGStab iterations at the coarsest level, giving a more effective but slightly more costly preconditioner than using Chebyshev alone. For example,

when the number of cores becomes larger than 30,000, the number of outer iterations when using the variable MG preconditioner is almost half that of using the fixed MG preconditioner, leading to a reduction of 10% and 20% in overall execution time on Jaguar and Intrepid, respectively.

Comparing the results obtained on the two machines, one sees a consistent iteration number (for some grid sizes, a slightly different number of processor cores was used across machines; this affected the innermost block Jacobi/ILU(0) preconditioner, thus making the outer iteration numbers slightly different). However, the time improvement of using a variable MG preconditioner is significantly different across the two machines. Because the clock rate of Intrepid is much lower than that of Jaguar, the solution time on Intrepid is longer. Because the communication network of Intrepid has lower latency, however, the global synchronization cost of `MPI_Allreduce` is significantly smaller. We thus achieve better performance improvement on Intrepid because the variable MG preconditioner requires inner product calculations in the coarsest grid solves. This phenomenon is not rare in practice and showcases that for a solver, not only the theoretical convergence matters, but also the machine architecture plays an important role.

The results of using FBiCGStab-R in place of FBiCGStab were almost the same in this application. With a stopping criterion of  $1e-5$  tolerance for the outer linear solves, the difference in residual norm decrease is unnoticeable. The gain in reducing synchronization cost is not significant enough because the outer iterations constituted a very small portion of the total run time. The advantage of FBiCGStab-R will show when a much smaller tolerance is needed, where it converges faster than does FBiCGStab.

**6. Concluding remarks.** BiCGStab has been the de facto method of choice in many application domains for solving linear systems. Motivated by the challenges in large-scale scientific applications and extreme-scale architectures that encourage the use of flexible preconditioners, we analyzed flexible BiCGStab and showed that the change of the convergence behavior with respect to standard BiCGStab is in accordance with the inaccuracy in the preconditioning solves. Thus, often a strong preconditioner is favored in order to successfully apply the method. To this end, we demonstrated through numerical experiments (including the PFLOTRAN reacting flow application) that two preconditioners are effective. One is BiCGStab itself, with a suitably chosen inner tolerance; the other is multigrid, using an efficient iterative method with a fixed number of inner iterations in the coarsest grid level. These examples are an effort toward the practical use of FBiCGStab in large-scale applications.

We also derived an alternative but mathematically equivalent version, FBiCGStab-R, in order to reduce the synchronization cost of the inner product calculations in FBiCGStab. Experiments showed that FBiCGStab-R is stable, and the stability is closely tied to the preserved numerical orthogonality in the iterate updates. In general, both FBiCGStab and FBiCGStab-R yield similar convergence behavior; however, sometimes FBiCGStab-R performs surprisingly far better than does FBiCGStab. The cause of this superior performance is unclear, but it shows that FBiCGStab-R can be a useful alternative to FBiCGStab even when the synchronization cost is not a bottleneck of the computations.

**Acknowledgments.** We thank Satish Balay and Jed Brown for insightful discussions and assistance with experiments. The authors were supported by the Office of Advanced Scientific Computing Research, Office of Science, U.S. Department of Energy, under Contract DE-AC02-06CH11357.

## REFERENCES

- [1] ALCF, *Intrepid Supercomputer*. <http://www.alcf.anl.gov/intrepid>.
- [2] J. ANG, K. EVANS, A. GEIST, M. HEROUX, P. HOVLAND, O. MARQUES, L. MCINNES, E. NG, AND S. WILD, *Report on the workshop on extreme-scale solvers: Transitions to future architectures*. Office of Advanced Scientific Computing Research, U.S. Department of Energy, 2012. Washington, DC, March 8-9, 2012.
- [3] S. BALAY, J. BROWN, K. BUSCHELMAN, V. EIJKHOUT, W. D. GROPP, D. KAUSHIK, M. G. KNEPLEY, L. C. MCINNES, B. F. SMITH, AND H. ZHANG, *PETSc users manual*, Tech. Rep. ANL-95/11 - Revision 3.3, Argonne National Laboratory, 2012.
- [4] S. BALAY, W. D. GROPP, L. C. MCINNES, AND B. F. SMITH, *Efficient management of parallelism in object oriented numerical software libraries*, in *Modern Software Tools in Scientific Computing*, E. Arge, A. M. Bruaset, and H. P. Langtangen, eds., Birkhauser Press, 1997, pp. 163–202.
- [5] P. G. BRIDGES, K. B. FERREIRA, M. A. HEROUX, AND M. HOEMMEN, *Fault-tolerant linear solvers via selective reliability*, CoRR, abs/1206.1390 (2012).
- [6] J. BROWN, M. G. KNEPLEY, D. A. MAY, L. C. MCINNES, AND B. F. SMITH, *Composable linear solvers for multiphysics*, in *Proceedings of the 11th International Symposium on Parallel and Distributed Computing (ISPDC 2012)*, IEEE Computer Society, 2012, pp. 55–62. Also available as Argonne National Laboratory preprint ANL/MCS-P2017-0112.
- [7] A. CHRONOPOULOS AND C. W. GEAR, *S-step iterative methods for symmetric linear systems*, *Journal of Computational and Applied Mathematics*, 25 (1989), pp. 153–168.
- [8] T. COLLIGNON AND M. VAN GIJZEN, *Minimizing synchronization in IDR(s)*, *Numerical Linear Algebra with Applications*, 18 (2011), pp. 805–825.
- [9] H. A. V. DER VORST AND C. VUIK, *GMRESR: a family of nested GMRES methods*, *Numerical Linear Algebra with Applications*, 1 (1994), pp. 369–386.
- [10] A. EL MALIKI, R. GUENETTE, AND M. FORTIN, *An efficient hierarchical preconditioner for quadratic discretizations of finite element problems*, *Numerical Linear Algebra with Applications*, 18 (2011), pp. 789–803.
- [11] J. V. ESHOF AND G. L. G. SLEIJPEN, *Inexact Krylov subspace methods for linear systems*, *SIAM Journal on Matrix Analysis and Applications*, 26 (2004), pp. 125–153.
- [12] R. FLETCHER, *Conjugate Gradient methods for indefinite systems*, *Lecture Notes in Mathematics*, 506 (1976), pp. 73–89.
- [13] P. GHYSELS, T. ASHBY, K. MEERBERGEN, AND W. VANROOSE, *Hiding global communication latency in the GMRES algorithm on massively parallel machines*, Tech. report 04.2012.1, Intel Exascale Lab, Leuven, Belgium, 2012.
- [14] E. GILADI, G. H. GOLUB, AND J. B. KELLER, *Inner and outer iterations for the Chebyshev algorithm*, *SIAM J. Numer. Anal.*, 35 (1995), pp. 300–319.
- [15] G. H. GOLUB AND Q. YE, *Inexact preconditioned Conjugate Gradient method with inner-outer iteration*, *SIAM Journal on Scientific Computing*, 21 (1999), pp. 1305–1320.
- [16] D. E. KEYES, L. C. MCINNES, C. WOODWARD, W. GROPP, E. MYRA, M. PERNICE, J. BELL, J. BROWN, A. CLO, J. CONNORS, E. CONSTANTINESCU, D. ESTEP, K. EVANS, C. FARHAT, A. HAKIM, G. HAMMOND, G. HANSEN, J. HILL, T. ISAAC, X. JIAO, K. JORDAN, D. KAUSHIK, E. KAXIRAS, A. KONIGES, K. LEE, A. LOTT, Q. LU, J. MAGERLEIN, R. MAXWELL, M. MCCOURT, M. MEHL, R. PAWLOWSKI, A. P. RANGLES, D. REYNOLDS, B. RIVIÈRE, U. RÜDE, T. SCHEIBE, J. SHADID, B. SHEEHAN, M. SHEPHARD, A. SIEGEL, B. SMITH, X. TANG, C. WILSON, AND B. WOHLMUTH, *Multiphysics Simulations: Challenges and Opportunities*, Tech. Rep. ANL/MCS-TM-321, Revision 1.1, Argonne National Laboratory, Oct 2012. To appear as a special issue of the *International Journal of High Performance Computing Applications*.
- [17] B. LEE AND G. HAMMOND, *Parallel performance of preconditioned Krylov solvers for the Richards equation*. manuscript, 2010.
- [18] P. LICHTNER ET AL., *PFLOTRAN project*. <http://ees.lanl.gov/pflotran/>.
- [19] C. LU AND P. C. LICHTNER, *PFLOTRAN: Massively parallel 3-D simulator for CO2 sequestration in geologic media*, in *Fourth Annual Conference on Carbon Capture and Sequestration DOE/NETL*, 2005.
- [20] L. C. MCINNES, B. SMITH, H. ZHANG, AND R. T. MILLS, *Hierarchical and nested Krylov methods for extreme-scale computing*, Preprint ANL/MCS-P2097-0512, Argonne National Laboratory, 2012. submitted to *Parallel Computing*.
- [21] R. MILLS, G. HAMMOND, P. LICHTNER, V. SRIPATHI, G. MAHINTHAKMUAR, AND B. SMITH, *Modeling subsurface reactive flows using leadership-class computing*, in *Journal of Physics: Conference Series*, vol. 180, IOP Publishing, 2009, p. 102062.

- [22] R. MILLS, C. LU, P. LICHTNER, AND G. HAMMOND, *Simulating subsurface flow and transport on ultrascale computers using PFLOTRAN*, in Journal of Physics: Conference Series, vol. 78, IOP Publishing, 2007, p. 012051.
- [23] R. T. MILLS, V. SRIPATHI, G. MAHINTHAKUMAR, G. HAMMOND, P. C. LICHTNER, AND B. F. SMITH, *Engineering PFLOTRAN for scalable performance on Cray XT and IBM BlueGene architectures*, in Proceedings of SciDAC 2010 Annual Meeting, 2010.
- [24] M. MOHIYUDDIN, M. HOEMMEN, J. DEMMEL, AND K. YELICK, *Minimizing communication in sparse matrix solvers*, in Proceedings of SC09, ACM, 2009.
- [25] *MPI: A message-passing interface standard*, International J. Supercomputing Applications, 8 (1994).
- [26] Y. NOTAY, *Flexible Conjugate Gradients*, SIAM Journal on Scientific Computing, 22 (2000), pp. 1444–1460.
- [27] OLCF, *Jaguar Supercomputer*. <https://www.olcf.ornl.gov/computing-resources/jaguar/>.
- [28] Y. SAAD, *A flexible inner-outer preconditioned GMRES algorithm*, SIAM Journal on Scientific Computing, 14 (1993), pp. 461–469.
- [29] Y. SAAD, *Iterative Methods for Sparse Linear Systems, 2nd edition*, SIAM, Philadelphia, PA, 2003.
- [30] Y. SAAD AND M. H. SCHULTZ, *GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems*, SIAM J. Sci. Stat. Comput., 7 (1986), pp. 856–869.
- [31] Y. SAAD AND M. SOSONKINA, *pARMS: A package for the parallel iterative solution of general large sparse linear systems user’s guide*, Tech. Rep. UMSI2004-8, Minnesota Supercomputer Institute, University of Minnesota, 2004.
- [32] V. SIMONCINI AND D. SZYLD, *Flexible inner-outer Krylov subspace methods*, SIAM Journal on Numerical Analysis, (2003), pp. 2219–2239.
- [33] V. SIMONCINI AND D. B. SZYLD, *Theory of inexact Krylov subspace methods and applications to scientific computing*, SIAM Journal on Scientific Computing, 25 (2003), pp. 454–477.
- [34] G. L. SLEIJPEN, P. SONNEVELD, AND M. B. VAN GIJZEN, *Bi-CGSTAB as an induced dimension reduction method*, Applied Numerical Mathematics, 60 (2010), pp. 1100–1114.
- [35] G. L. SLEIJPEN AND M. B. VAN GIJZEN, *Exploiting BiCGstab( $\ell$ ) strategies to induce dimension reduction*, SIAM J. Sci. Comput., 32 (2010), pp. 2687–2709.
- [36] P. SONNEVELD AND M. B. VAN GIJZEN, *IDR(s): A family of simple and fast algorithms for solving large nonsymmetric systems of linear equations*, SIAM Journal on Scientific Computing, 31 (2008), pp. 1035–1062.
- [37] E. D. STURLER AND H. A. VAN DER VORST, *Reducing the effect of global communication in GMRES(m) and CG on parallel distributed memory computers*, Applied Numerical Mathematics, 18 (1995), pp. 441–459.
- [38] D. B. SZYLD AND J. A. VOGEL, *FQMR: A flexible Quasi-Minimal Residual method with inexact preconditioning*, SIAM Journal on Scientific Computing, 23 (2001), pp. 363–380.
- [39] H. VAN DER VORST, *BiCGSTAB: A fast and smoothly converging variant of BiCG for the solution of nonsymmetric linear systems*, SIAM J. Sci. Stat. Comput., 13 (1992), pp. 631–644.
- [40] M. B. VAN GIJZEN, G. L. SLEIJPEN, AND J.-P. M. ZEMKE, *Flexible and multi-shift induced dimension reduction algorithms for solving large sparse linear systems*, Tech. Rep. 11-06, Delft University of Technology, 2011.
- [41] J. VAN ROSENDALE, *Minimizing inner product data dependencies in conjugate gradient iteration*, in Proceedings of the IEEE International Conference on Parallel Processing, IEEE Computer Society, 1983.
- [42] J. A. VOGEL, *Flexible BiCG and flexible Bi-CGSTAB for nonsymmetric linear systems*, Applied Mathematics and Computation, 188 (2007), pp. 226–233.
- [43] R. VUDUC, *Quantitative performance modeling of scientific computations and creating locality in numerical algorithms*, PhD thesis, Massachusetts Institute of Technology, 1995.
- [44] L. T. YANG AND R. 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.

**Government License.** The submitted manuscript has been created by UChicago Argonne, LLC, Operator of Argonne National Laboratory (“Argonne”). Argonne, a U.S. Department of Energy Office of Science laboratory, is operated under Contract No. DE-AC02-06CH11357. The U.S. Government retains for itself, and others acting on its behalf, a paid-up nonexclusive, irrevocable worldwide license in said article to reproduce, prepare derivative works, distribute copies to the public, and perform publicly and display publicly, by or on behalf of the Government.