Oil & Gas Science and Technology – Rev. IFP Energies nouvelles (2016) **71**, 66

DOI: 10.2516/ogst/2016021

Dossier

SimRace 2015: Numerical Methods and High Performance Computing for Industrial Fluid Flows
SimRace 2015 : Méthodes numériques et calcul haute performance pour la simulation d'écoulements complexes

S-Step BiCGStab Algorithms for Geoscience Dynamic Simulations

Ani Anciaux-Sedrakian[1], Laura Grigori[2,3], Sophie Moufawad[1]* and Soleiman Yousef[1]

[1] *IFP Energies nouvelles, 1-4 avenue de Bois-Préau, 92852 Rueil-Malmaison Cedex - France*
[2] *INRIA Paris, Alpines, 2 Rue Simone IFF, 75012 Paris - France*
[3] *UPMC - Univ Paris 6, CNRS UMR 7598, Laboratoire Jacques-Louis Lions, 4 Place Jussieu, 75005 Paris - France*
e-mail: ani.anciaux-sedrakian@ifpen.fr - laura.grigori@inria.fr - sm101@aub.edu.lb - soleiman.yousef@ifpen.fr

* Corresponding author

Abstract *— In basin and reservoir simulations, the most expensive and time consuming phase is solving systems of linear equations using Krylov subspace methods such as BiCGStab. For this reason, we explore the possibility of using communication avoiding Krylov subspace methods (*s*-step BiCGStab), that speedup of the convergence time on modern-day architectures, by restructuring the algorithms to reduce communication. We introduce some variants of* s*-step BiCGStab with better numerical stability for the targeted systems.*

Résumé — **Méthodes *s*-step BiCGStab appliquées en Géosciences** — Dans les simulateurs d'écoulement en milieu poreux, comme les simulateurs de réservoir et de bassin, la résolution de système linéaire constitue l'étape la plus consommatrice en temps de calcul et peut même représenter jusqu'à 80 % du temps de la simulation. Ceci montre que la performance de ces simulateurs dépend fortement de l'efficacité des solveurs linéaires. En même temps, les machines parallèles modernes disposent d'un grand nombre de processeurs et d'unités de calcul massivement parallèle. Dans cet article, nous proposons de nouveaux algorithmes BiCGStab, basés sur l'algorithme à moindre communication nommé *s*-step, permettant d'éviter un certain nombre de communication afin d'exploiter pleinement les architectures hautement parallèles.

INTRODUCTION

Many scientific problems require the solution of systems of linear equations of the form $Ax = b$, where the input matrix A is very large and sparse. These systems arise mainly from the discretization of Partial Differential Equations (PDE), and are usually solved using Krylov subspace methods, such as Generalized Minimal RESidual (GMRES) [1], Conjugate Gradient (CG) [2] and Bi-Conjugated Gradient Stabilized (BiCGStab) [3].

In the case of basin modeling or reservoir simulations with highly heterogeneous data and complex geometries, complex non-linear systems of PDE are solved. These PDE are discretized with a cell-centered finite volume scheme in space, leading to a non-linear system which is solved with an iterative Newton solver. At each Newton step, the system is linearized. Then, the generated large, sparse and unstructured linear system is solved using preconditioned GMRES, BiCG-Stab, CG, Orthomin or other preconditioned iterative methods. Some of the most commonly used preconditioners are ILU(k), ILUT, AMG and CPR-AMG. This resolution phase constitutes the most expensive part of the simulation. Thus we focus on linear solvers, since their efficiency is a key point for the simulator's performance.

Furthermore, modern parallel computing resources are based on complex hardware architecture. They are composed of several multi-core processors and massively parallel processing units such as many-cores or General-Purpose GPU (GPGPU) cards. Most of the current algorithms are not able to fully exploit the highly parallel architectures. In fact, a severe degradation of performance is detected when the number of processing units is increased. This is due to the difference between the required time to perform floating point operations (flops) by processing units and the time to communicate the obtained results, where flops have become much cheaper than data communication.

Thus, recent research has focused on reformulating dense and sparse linear algebra algorithms with the aim of reducing and avoiding communication. These methods are referred to as communication avoiding methods, whereby communication refers to data movement between different processing units in parallel, and different levels of memory hierarchy. In the case of Krylov subspace methods, the introduced communication avoiding Krylov subspace methods [4-8] are based on *s*-step methods [9-11]. The goal is to restructure the algorithms to perform *s* iterations at a time by using kernels that avoid or reduce communication, such as the matrix powers kernel [12], Tall and Skinny QR (TSQR) [13], and Block Gram Schmidt (BGS) methods.

Our aim is to reduce the overall cost of the linear solver resolution phase in geoscience simulations, specifically basin and reservoir simulations, using a parallel implementation of BiCGStab that avoids communication on multi-core hardware (CA-BiCGStab) [5] and has a similar convergence behavior as the classical BiCGStab method. Communication Avoiding BiCGStab (CA-BiCGStab), which was introduced in [5], is a reformulation of BiCGStab into *s*-step BiCGStab that avoids communication. Thus, in this paper we study the convergence behavior of a sequential version of the unpreconditioned *s*-step BiCGStab [5], on matrices obtained from reservoir simulations, with different *s* values. The obtained results show that, for most of the tested matrices, *s*-step BiCGStab requires more iterations to converge than BiCGStab. Thus, we design new variants of *s*-step BiCGStab, that have the same convergence rate as BiCGStab for *s* values between 2 and 6, and reduce communication similarly to *s*-step BiCGStab.

In Section 1, we introduce BiCGStab, its reformulation to *s*-step BiCGStab [5], and we discuss the performance of *s*-step BiCGStab in geoscience applications, specifically reservoir simulations. Then, in Section 2, we introduce the new *s*-step BiCGStab variants that we call orthonormalized *s*-step BiCGStab, split orthonormalized *s*-step BiCGStab, and modified split orthonormalized *s*-step BiCGStab. In Section 3, we present the convergence results of the newly introduced *s*-step BiCGStab variants and compare them to that of *s*-step BiCGStab. Finally, we conclude.

1 FROM BICGSTAB TO *S*-STEP BICGSTAB

In this section we briefly introduce BiCGStab (Sect. 1.1) and *s*-step BiCGStab (Sect. 1.2). We show the relation between both methods and their convergence in reservoir simulations (Sect. 1.3).

1.1 BiCGStab

The Bi-Conjugate Gradient Stabilized method (BiCGStab), introduced by van der Vorst in 1992 [3], is an iterative Krylov subspace method that solves the general systems $Ax = b$. It is a variant of the Bi-Conjugate Gradient (BiCG) method that aims at smoothing BiCG's erratic convergence. At each iteration $m \geq 0$, $r_{m+1} = P_{m+1}(A)r_0$ is replaced by $r_{m+1} = Q_{m+1}(A)P_{m+1}(A)r_0$ where $Q_{m+1}(z) \in \mathcal{P}_{m+1}$ and $P_{m+1}(z) \in \mathcal{P}_{m+1}$ are polynomials of degree $m+1$. $Q_{m+1}(z)$ is chosen to be

$$Q_{m+1}(z) = \prod_{j=1}^{m+1} (1 - \omega_{j-1}z) = (1 - \omega_m z)Q_m(z)$$

where ω_m minimizes the norm of r_{m+1}.

BiCGStab, being a variant of BiCG, has a similar form. But the recurrence relations of x_{m+1}, r_{m+1}, p_{m+1}, α_m and β_m are different.

$$x_{m+1} = x_m + \alpha_m p_m + \omega_m[r_m - \alpha_m A p_m] \quad (1)$$

$$r_{m+1} = (I - \omega_m A)[r_m - \alpha_m A p_m] \quad (2)$$

$$p_{m+1} = r_{m+1} + \beta_m(I - \omega_m A)p_m \quad (3)$$

where $r_0 = b - Ax_0$, and $p_0 = r_0$.

The scalars α_m and β_m are defined as follows:

$$\alpha_m = \frac{\langle \tilde{r}_0, r_m \rangle}{\langle \tilde{r}_0, Ap_m \rangle}, \ \beta_m = \frac{\alpha_m}{\omega_m} \frac{\langle \tilde{r}_0, r_{m+1} \rangle}{\langle \tilde{r}_0, r_m \rangle} \quad (4)$$

where $\tilde{r}_0$ is chosen such that $< \tilde{r}_0, r_0 > \neq 0$ as shown in Algorithm 1. In general $\tilde{r}_0$ is set equal to r_0. As for ω_m, it is defined by minimizing the norm of the residual r_{m+1}, *i.e.* $||(I-\omega_m A)(r_m-\alpha_m A p_m)|| = \min_{\omega\in\mathbb{R}}||(I-\omega A)(r_m-\alpha_m A p_m)||$, where

$$\omega_m = \frac{\langle Ar_m - \alpha_m A^2 p_m, r_m - \alpha_m A p_m \rangle}{\langle Ar_m - \alpha_m A^2 p_m, Ar_m - \alpha_m A^2 p_m \rangle} \quad (5)$$

In addition, we have that for $m > 0$

$$\begin{cases} p_m, r_m \in \mathcal{K}_{2m+1}(A, p_0) + \mathcal{K}_{2m}(A, r_0) \\ x_m - x_0 \in \mathcal{K}_{2m}(A, p_0) + \mathcal{K}_{2m-1}(A, r_0) \end{cases} \quad (6)$$

and more generally for $m \geq 0$ and $j > 0$

$$\begin{cases} p_{m+j}, r_{m+j} \in \mathcal{K}_{2j+1}(A, p_m) + \mathcal{K}_{2j}(A, r_m) \\ x_{m+j} - x_m \in \mathcal{K}_{2j}(A, p_m) + \mathcal{K}_{2j-1}(A, r_m) \end{cases} \quad (7)$$

Algorithm 1: BiCGStab

Input: A, b, x_0, $m_{\max}$: the maximum allowed iterations

Output: x_m: the mth approximate solution satisfying the stopping criteria

Let $r_0 = b - Ax_0$, $p_0 = r_0$, $\rho_0 = \langle r_0, r_0 \rangle$, and $m = 0$
Choose $\tilde{r}_0$ such that $\delta_0 = \langle \tilde{r}_0, r_0 \rangle \neq 0$.
While ($\sqrt{\rho_m} > \epsilon ||b||_2$ and $m < m_{\max}$) **Do**
  $\alpha_m = \delta_m / \langle \tilde{r}_0, Ap_m \rangle$
  $s = r_m - \alpha_m A p_m$
  $t = As$
  $\omega_m = \langle t, s \rangle / \langle t, t \rangle$
  $x_{m+1} = x_m + \alpha_m p_m + \omega_m s$
  $r_{m+1} = (I - \omega_m A)s$
  $\delta_{m+1} = \langle \tilde{r}_0, r_{m+1} \rangle$
  $\beta_m = (\delta_{m+1}/\delta_m)(\alpha_m/\omega_m)$
  $p_{m+1} = r_{m+1} + \beta_m (I - \omega_m A) p_m$
  $\rho_{m+1} = \langle r_{m+1}, r_{m+1} \rangle$, $m = m + 1$
end for

At each iteration of Algorithm 1, two sparse matrix-vector multiplications, six saxpy's, and five dot products are computed. Given that each processor has the scalars and its corresponding part of the vectors, then the saxpy's can be parallelized without communication. However, this is not the case for the sparse matrix-vector multiplications and the dot products, which require communication to obtain the desired results. Such operations cause a severe performance degradation, especially when using modern computing resources.

1.2 S-Step BiCGStab

To reduce the communication in parallel and sequential implementations of BiCGStab, Carson *et al.* [5] introduced the s-step version of BiCGStab. The reformulation is based on the computation of s BiCGStab iterations at once, and on the fact that for $m \geq 0$ and $1 \leq j \leq s$

$$\begin{cases} p_{m+j}, r_{m+j} \in \mathcal{K}_{2s+1}(A, p_m) + \mathcal{K}_{2s}(A, r_m) \\ x_{m+j} - x_m \in \mathcal{K}_{2s}(A, p_m) + \mathcal{K}_{2s-1}(A, r_m) \end{cases} \quad (8)$$

since $\mathcal{K}_{2j+1}(A, z) \subseteq \mathcal{K}_{2s+1}(A, z)$ for any $z \neq 0$.

The goal is to perform more flops per communication, by computing $2s$ matrix-vector products at the beginning of each iteration of the s-step BiCGStab. This would reduce the communication cost, specifically the number of messages, by $O(s)$ times in parallel [5]. However, this is not possible using the same formulation as BiCGStab. Therefore, at the beginning of each s-step iteration, one computes P_{2s+1} and R_{2s}, the Krylov matrices corresponding to the $\mathcal{K}_{2s+1}(A, p_m)$ and $\mathcal{K}_{2s}(A, r_m)$ bases respectively, where $m = 0, s, 2s, 3s, \ldots$. Then, by Equation (8), p_{m+j}, r_{m+j}, and $x_{m+j} - x_m$ can be defined as the product of the basis vectors and a span vector, for $j = 0, \ldots, s$,

$$p_{m+j} = [P_{2s+1}, R_{2s}]a_j \quad (9)$$

$$r_{m+j} = [P_{2s+1}, R_{2s}]c_j \quad (10)$$

$$x_{m+j} = x_m + [P_{2s+1}, R_{2s}]e_j \quad (11)$$

where $[P_{2s+1}, R_{2s}]$ is an $n \times (4s+1)$ matrix containing the basis vectors of $\mathcal{K}_{2s+1}(A, p_m)$ and $\mathcal{K}_{2s}(A, r_m)$, and a_j, c_j, and e_j are span vectors of size $4s + 1$. Note that $e_0 = 0$. As for a_0 and c_0, their definition depends on the type of computed basis.

One can compute a basis defined by a recurrence relation with three or less terms, such as a monomial, scaled monomial, Newton, or Chebyshev basis. Then, we have that

$$AP_{2s} = P_{2s+1}T_{2s+1} \quad (12)$$

$$AR_{2s-1} = R_{2s}T_{2s} \quad (13)$$

$$A[P_{2s}, 0, R_{2s-1}, 0] = [P_{2s+1}, R_{2s}]T' \quad (14)$$

where T_{2s+1} and T_{2s} are change of basis matrices of size $(2s + 1) \times (2s)$ and $(2s) \times (2s - 1)$ respectively, and

$$T' = \begin{bmatrix} [T_{2s+1} & 0] & \\ & [T_{2s} & 0] \end{bmatrix}$$

is a $(4s + 1) \times (4s + 1)$ matrix.

The definition of T_{2s+1}, T_{2s} and eventually T'depends on the chosen type of basis

$$[P_{2s+1}, R_{2s}] = [\overline{p}_m, \overline{p}_{m+1}, \ldots, \overline{p}_{m+2s}, \overline{r}_m, \overline{r}_{m+1}, \ldots, \overline{r}_{m+2s-1}]$$

For example, in the monomial basis case, where $\overline{p}_m = p_m$, $\overline{r}_m = r_m$, $\overline{p}_{m+i} = A\overline{p}_{m+i-1}$, and $\overline{r}_{m+i} = A\overline{r}_{m+i-1}$ for $i > 0$, the matrices T_{2s+1} and T_{2s} are all zeros except the lower diagonal which is ones, *i.e.* $T_{2s+1}(i + 1, i) = 1$ for $i = 1, \ldots, 2s$. In the case of the scaled monomial basis, where $\overline{p}_m = \frac{p_m}{||p_m||}$ and $\overline{p}_{m+i} = \frac{A\overline{p}_{m+i-1}}{||A\overline{p}_{m+i-1}||}$, the matrices are defined as $T_{2s+1}(i + 1, i) = ||A\overline{p}_{m+i-1}||$ for $i = 1, \ldots, 2s$, and $T_{2s}(i + 1, i) = ||A\overline{r}_{m+i-1}||$ for $i = 1, \ldots, 2s - 1$, and zero elsewhere.

The reformulation of BiCGStab into s-step BiCGStab starts by replacing the definitions (9)-(11) in Equations (1)-(3), and taking into consideration that for $j = 0$ to $s - 1$.

$$Ap_{m+j} = A[P_{2s+1}, R_{2s}]a_j = A[P_{2s}, 0, R_{2s-1}, 0]a_j \quad (15)$$

$$= [P_{2s+1}, R_{2s}]T'a_j \quad (16)$$

$$Ar_{m+j} = [P_{2s+1}, R_{2s}]T'c_j \quad (17)$$

$$A^2 p_{m+j} = [P_{2s+1}, R_{2s}](T')^2 a_j \quad (18)$$

Then we get the following,

$$e_{j+1} = e_j + \alpha_{m+j} a_j + \omega_{m+j} c_j - \omega_{m+j}\alpha_{m+j} T' a_j \quad (19)$$

$$a_{j+1} = c_{j+1} + \beta_{m+j} a_j - \beta_{m+j}\omega_{m+j} T' a_j \quad (20)$$

$$c_{j+1} = c_j - \alpha_{m+j} T' a_j - \omega_{m+j} T'(c_j - \alpha_{m+j} T' a_j) \quad (21)$$

with $e_0 = 0$, $a_0 = [1, 0_{4s}]^T$, and $c_0 = [0_{2s+1}, 1, 0_{2s-1}]$ in the case of monomial and Newton basis. Then, the span vectors, a_j, c_j, and e_j are updated for $j = 1, \ldots, s$, rather than p_{m+j}, r_{m+j}, and x_{m+j} which are of size $n \gg 4s + 1$.

As for α_{m+j}, δ_{m+j+1}, and ω_{m+j}, it is sufficient to replace r_{m+j} and p_{m+j} by definitions (10) and (9) to obtain

$$\delta_{m+j+1} = \langle \tilde{r}_0, r_{m+j+1} \rangle = \langle g, c_{j+1} \rangle$$

$$\alpha_{m+j} = \delta_{m+j} / \langle \tilde{r}_0, A p_{m+j} \rangle = \delta_{m+j} / \langle g, T' a_{j+1} \rangle$$

$$\omega_{m+j} = \frac{\langle T' c_j - \alpha_{m+j}(T')^2 a_j, G c_j - \alpha_{m+j} G T' a_j \rangle}{\langle T' c_j - \alpha_{m+j}(T')^2 a_j, G T' c_j - \alpha_{m+j} G (T')^2 a_j \rangle}$$

where $G = [P_{2s+1}, R_{2s}]^T [P_{2s+1}, R_{2s}]$ is a Gram-like matrix, $g = [P_{2s+1}, R_{2s}]^T \tilde{r}_0$. Then, $\beta_{m+j} = \frac{\delta_{m+j+1}}{\delta_{m+j}} \frac{\alpha_{m+j}}{\omega_{m+j}}$.

Algorithm 2: *s*-step BiCGStab

Input: A, b, x_0, m_{max}, s, *Type of Basis*
Output: x_m: the mth approximate solution satisfying the stopping criteria

Let $r_0 = b - Ax_0$, $p_0 = r_0$, $\rho_0 = \langle r_0, r_0 \rangle$, and $k = 0$
Choose $\tilde{r}_0$ such that $\delta_0 = \langle \tilde{r}_0, r_0 \rangle \neq 0$.
While $\left(\sqrt{\rho_k} > \epsilon ||b||_2 \text{ and } k < \lfloor \frac{m_{\max}}{s} \rfloor\right)$ **Do**
  Compute P_{2s+1} and R_{2s} depending on Type of Basis, and output the diagonals of T'
  $G = [P_{2s+1}, R_{2s}]^T [P_{2s+1}, R_{2s}]$ and $g = [P_{2s+1}, R_{2s}]^T \tilde{r}_0$
  Initialize a_0, e_0, c_0 and set $m = k * s$
  for ($j = 0$ to $s - 1$) **Do**
    $t_a = T' a_j$
    $\alpha_{m+j} = \delta_{m+j} / \langle g, t_a \rangle$
    $d = c_j - \alpha_{m+j} t_a$
    $t_d = T' d$
    $g_d = G d$
    $g_t = G t_d$
    $\omega_{m+j} = \langle t_d, g_d \rangle / \langle t_d, g_t \rangle$
    $e_{j+1} = e_j + \alpha_{m+j} a_j + \omega_{m+j} d$
    $c_{j+1} = c_j - \omega_{m+j} t_d - \alpha_{m+j} t_a$
    $\delta_{m+j+1} = \langle g, c_{j+1} \rangle$
    $\beta_{m+j} = (\delta_{m+j+1}/\delta_{m+j})(\alpha_{m+j}/\omega_{m+j})$
    $a_{j+1} = c_{j+1} + \beta_{m+j} a_j - \beta_{m+j}\omega_{m+j} t_a$
  end for
  $p_{m+s} = [P_{2s+1}, R_{2s}] a_s$, $r_{m+s} = [P_{2s+1}, R_{2s}] z_s$
  $x_{m+s} = x_m + [P_{2s+1}, R_{2s}] e_s$
  $\rho_{m+s} = \langle r_{m+s}, r_{m+s} \rangle$, $k = k + 1$
end While

Algorithm 2 is a reformulation of BiCGStab that reduces communication where the matrix-vector multiplications are grouped at the beginning of the outer iteration, and the Gram-like matrix G is computed once per outer iteration. Then, in the inner iterations, the vector operations of size n are replaced by vector operations of size $4s + 1$, where $4s + 1 \ll n$. However, this reformulation alone is not sufficient to reduce communication.

For example, in the sequential case the basis computation should be done using the matrix powers kernel [12], where $\overline{p}_{m+j+1}$ is computed by parts that fit into cache memory, for $j = 0, \ldots, 2s - 1$. This reduces communication in the memory hierarchy of the processor and increases cache hits. In the parallel case, each processor fetches, at the beginning, the needed data from neighboring processors to compute its assigned part of the $2s$ vectors $\overline{p}_{m+j+1}$ without any communication, for $j = 0, \ldots, 2s - 1$. Similarly, we can compute the $2s - 1$ vectors $\overline{r}_{m+j+1}$ for $j = 0, \ldots, 2s - 2$. Note that it is possible to compute the two bases simultaneously using a block version of the matrix powers kernel that computes a block of vectors without communication.

1.3 S-Step BiCGStab for Geoscience Applications

In geoscience applications, specifically in reservoir simulations, at each time step a new linear system of the form $Ax = b$ has to be solved. The difficulty and the ill-conditioning of the systems may vary throughout the simulation. However, in most cases, an iterative method and a preconditioner are chosen at the beginning of the simulation and are used for solving all the obtained linear systems. Since the obtained systems are not symmetric, Krylov subspace methods such as BiCGStab are used. Our aim is to implement a numerically stable version of *s*-step BiCGStab that has a similar convergence rate as BiCGStab for the reservoir simulations systems. The stability of *s*-step BiCGStab is related to the chosen s value and to the type of the basis.

Thus, we study the convergence of the *s*-step BiCGStab (Algorithm 2) method for different s values, using the monomial and Newton basis [7, 14, 15] and compare it to BiCGStab's convergence. We do not consider the scaled versions of the monomial and Newton basis, since this requires the computation of the norm of each vector at a time, which annihilates the possibility of avoiding communication in the matrix powers kernel. The test matrices, described in Section 3.1, are obtained from different reservoir simulations.

A sample of the obtained results is described in Section 3.2 (Tab. 3). For the well-conditioned matrices, the s-step BiCGStab with the monomial basis converges in fewer s-step iterations as s increases from 2 to 6. But for the ill-conditioned matrices, the convergence of the s-step BiCGStab with monomial basis is chaotic with respect to s. Note that, we focus on the consistency of the convergence behavior as s increases for the following reasons. First, as s increases, the communication cost per s-step BiCGStab iteration decreases, since more flops are performed per communication. Moreover, if the convergence of s-step BiCGStab improves as s increases, then the overall communication cost is decreased which should lead to a speedup in parallel implementations. Second, although we test the convergence of s-step BiCGStab on matrices obtained from reservoir simulations, our goal is the speedup obtained in the full simulation. As mentioned earlier, the obtained linear systems could be well-conditioned or ill-conditioned. However, at the beginning of the simulation a single s value is chosen without having any information on the systems to be solved. Thus, we would like to implement an s-step BiCGStab version for which the number of iterations needed for convergence decreases as s increases to some upper limit.

An alternative to the monomial basis is the Newton basis, which is known to be more stable [7] in the case of GMRES. However, in the case of s-step BiCGStab, computing a Newton basis is expensive. First, the $2s$ largest eigenvalues have to be computed once per matrix using some library such as ARPACK [16]. In general, the computed eigenvalues could be almost equal, which does not improve the stability of the basis. Thus, the eigenvalues are reordered in the Leja ordering as discussed in [7]. For the well-conditioned matrices obtained in geoscience simulations, using the Newton basis did not improve the convergence of s-step BiCGStab. Moreover, for the ill-conditioned matrices, finding the desired number of eigenvalues is time consuming. In addition, in geoscience simulations, we seek a relatively "cheap" and stable basis, since several linear systems are solved during the simulation (at least one per time step). For all these reasons, we will use the monomial basis and improve its numerical stability, as discussed in the next section.

2 ORTHONORMALIZED S-STEP BICGSTAB

The s-step BiCGStab method with the monomial basis, has an irregular convergence with respect to the s values, and converges slower than BiCGStab for some of the tested systems. This irregular and slow convergence might be due to the fact that the estimated residual used for stopping criterion is not equal to the exact residual [4]. However, in our case, the slow convergence is caused by the basis vectors which become numerically linearly dependent. One way to improve the stability of the basis is by orthonormalizing it. We propose a new variant of s-step BiCGStab that orthonormalizes the basis vectors. This new version is referred to as orthonormalized s-step BiCGStab (Algorithm 3).

There are several ways of constructing the basis and orthonormalizing it. However, we derive the algorithm irrespective of the method used. We replace the $4s + 1$ basis vectors $[P_{2s+1}, R_{2s}]$ by an orthonormal basis Q_{4s+1}. Then, the vectors p_{m+j}, r_{m+j}, and x_{m+j} can be defined as follows,

$$p_{m+j} = Q_{4s+1} a_j \tag{22}$$

$$r_{m+j} = Q_{4s+1} c_j \tag{23}$$

$$x_{m+j} - x_m = Q_{4s+1} e_j \tag{24}$$

The $n \times (4s + 1)$ orthonormal matrix Q_{4s+1} should satisfy $AQ_{4s+1}v = Q_{4s+1}H_{4s+1}v$, where $Q_{4s+1}v \in \mathcal{K}_{2s}(A, p_m) + \mathcal{K}_{2s-1}(A, r_m)$ and H_{4s+1} is a $(4s + 1) \times (4s + 1)$ upper Hessenberg matrix. Then, for $j = 0$ to $s - 1$ we get

$$Ap_{m+j} = Q_{4s+1} H_{4s+1} a_j \tag{25}$$

$$Ar_{m+j} = Q_{4s+1} H_{4s+1} c_j \tag{26}$$

$$A^2 p_{m+j} = Q_{4s+1} (H_{4s+1})^2 a_j \tag{27}$$

By replacing the definitions (22)-(27) in Equations (1)-(3), we get that

$$\begin{aligned} e_{j+1} &= e_j + \alpha_{m+j} a_j + \omega_{m+j} c_j - \omega_{m+j}\alpha_{m+j} H_{4s+1} a_j \\ a_{j+1} &= c_{j+1} + \beta_{m+j} a_j - \beta_{m+j}\omega_{m+j} H_{4s+1} a_j \\ c_{j+1} &= c_j - \alpha_{m+j} H_{4s+1} a_j - \omega_{m+j} H_{4s+1}\left(c_j - \alpha_{m+j} H_{4s+1} a_j\right) \end{aligned}$$

with $e_0 = 0$. As for a_0 and c_0, their definitions depend on the orthonormalization technique used. We will discuss this in Section 2.1.

As for α_{m+j} and δ_{m+j+1}, it is sufficient to replace r_{m+j} and p_{m+j} by definitions (22) and (23) to obtain

$$\begin{aligned} \delta_{m+j+1} &= \left\langle \tilde{r}_0, r_{m+j+1} \right\rangle = \left\langle g, c_{j+1} \right\rangle \\ \alpha_{m+j} &= \frac{\delta_{m+j}}{\left\langle \tilde{r}_0, Ap_{m+j} \right\rangle} = \frac{\delta_{m+j}}{\left\langle g, H_{4s+1} a_{j+1} \right\rangle} \end{aligned}$$

where $g = Q_{4s+1}^T \tilde{r}_0$. Similarly for ω_{m+j}, we get

$$\omega_{m+j} = \frac{\left\langle c_j - \alpha_{m+j} H_{4s+1} a_j, H_{4s+1} c_j - \alpha_{m+j} (H_{4s+1})^2 a_j \right\rangle}{\left\langle H_{4s+1} c_j - \alpha_{m+j} (H_{4s+1})^2 a_j, H_{4s+1} c_j - \alpha_{m+j} (H_{4s+1})^2 a_j \right\rangle} \tag{28}$$

since $Q_{4s+1}^T Q_{4s+1} = I$. Finally, $\beta_{m+j} = \frac{\delta_{m+j+1}}{\delta_{m+j}} \frac{\alpha_{m+j}}{\omega_{m+j}}$.

Algorithm 3 describes the orthonormalized s-step BiCGStab method except for the orthonormal basis construction phase, which we discuss in Section 2.1.

Algorithm 3: Orthonormalized s-step BiCGStab

Input: A, b, x_0, m_{max}, s, Type of Basis
Output: x_m: the mth approximate solution satisfying the stopping criteria

Let $r_0 = b - Ax_0$, $p_0 = r_0$, $\rho_0 = \langle r_0, r_0 \rangle$, and $k = 0$
Choose $\tilde{r}_0$ such that $\delta_0 = \langle \tilde{r}_0, r_0 \rangle \neq 0$.
While $\left(\sqrt{\rho_k} > \epsilon ||b||_2 \text{ and } k < \left\lfloor \frac{m_{\max}}{s} \right\rfloor\right)$ **Do**
  Compute the orthonormal basis Q_{4s+1} and
    the upper Hessenberg matrix H_{4s+1}
  Compute $g = Q_{4s+1}^T \tilde{r}$
  Initialize a_0, e_0, c_0 and set $m = k * s$
  for ($j = 0$ to $s - 1$) **Do**
    $h_a = H_{4s+1} a_j$
    $\alpha_{m+j} = \delta_{m+j} / \langle g, h_a \rangle$
    $d = c_j - \alpha_{m+j} h_a$
    $h_d = H_{4s+1} d$
    $\omega_{m+j} = \langle d, h_d \rangle / \langle h_d, h_d \rangle$
    $e_{j+1} = e_j + \alpha_{m+j} a_j + \omega_{m+j} d$
    $c_{j+1} = c_j - \omega_{m+j} h_d - \alpha_{m+j} h_a$
    $\delta_{m+j+1} = \langle g, c_{j+1} \rangle$
    $\beta_{m+j} = (\delta_{m+j+1}/\delta_{m+j})(\alpha_{m+j}/\omega_{m+j})$
    $a_{j+1} = c_{j+1} + \beta_{m+j} a_j - \beta_{m+j}\omega_{m+j} h_a$
  end for
  $p_{m+s} = Q_{4s+1} a_s$, $r_{m+s} = Q_{4s+1} z_s$,
  $x_{m+s} = x_m + Q_{4s+1} e_s$
  $\rho_{m+s} = \langle r_{m+s}, r_{m+s} \rangle$, $k = k + 1$
end While

2.1 Construction of the Orthonormal Basis

The simplest parallelizable way to compute the orthonormal basis Q_{4s+1} is to compute first $[P_{2s+1}, R_{2s}]$ using the matrix powers kernel. Then, orthonormalize it using a QR algorithm, such as the Tall and Skinny QR (TSQR) algorithm [13] that requires log (p) messages in parallel, where p is the number of processors. In this case,

$$a_0 = U_{4s+1} \times [1, 0_{4s}]^T$$

and

$$c_0 = U_{4s+1} \times [0_{2s+1}, 1, 0_{2s-1}]^T$$

where U_{4s+1} is the $(4s + 1) \times (4s + 1)$ upper triangular matrix obtained from the QR factorization of $[P_{2s+1}, R_{2s}]$. In addition,

$$A[P_{2s}, 0, R_{2s-1}, 0] = [P_{2s+1}, R_{2s}]T'$$

where T' is the change of basis matrix. By replacing $[P_{2s+1}, R_{2s}]$ by $Q_{4s+1}U_{4s+1}$, obtained from the QR factorization, and by assuming that U_{4s+1} is invertible, we get:

$$\begin{aligned} AQ_{4s+1}v &= A[P_{2s+1}, R_{2s}]U_{4s+1}^{-1}v \\ &= A[P_{2s}, 0, R_{2s-1}, 0]U_{4s+1}^{-1}v \\ &= [P_{2s+1}, R_{2s}]T'U_{4s+1}^{-1}v \\ &= Q_{4s+1}U_{4s+1}T'U_{4s+1}^{-1}v \\ &= Q_{4s+1}H_{4s+1}v \end{aligned}$$

Q_{4s+1} is an $n \times (4s + 1)$ orthonormal matrix, $H_{4s+1} = U_{4s+1}T'U_{4s+1}^{-1}$ is a $(4s + 1) \times (4s + 1)$ upper Hessenberg matrix, and

$$Q_{4s+1}v \in \mathcal{K}_{2s}(A, p_m) + \mathcal{K}_{2s-1}(A, r_m)$$

Note that the matrix H_{4s+1} is never constructed and multiplying H_{4s+1} by a vector is equivalent to solving an upper triangular system and multiplying T' and U_{4s+1} by a vector.

However, there are two issues to take into consideration in the construction of the orthonormal basis. First, at iteration $k \geq 0$, we compute two bases P_{2s+1} and R_{2s}, for two different subspaces $\mathcal{K}_{2s+1}(A, p_k)$ and $\mathcal{K}_{2s}(A, r_k)$. There is no guarantee that the two bases are linearly independent with respect to each others. In other words, the $4s + 1$ vectors obtained are not necessarily linearly independent. Moreover, the upper triangular matrix obtained from the orthonormalization of a linearly dependent set of vectors, is not invertible. Second, at iteration $k = 0$, we compute two bases of $\mathcal{K}_{2s+1}(A, r_0)$ and $\mathcal{K}_{2s}(A, r_0)$ subspaces, since p_0 is initialized to r_0, as in the BiCGStab and s-step BiCGStab algorithms.

A solution to the first problem, is to perform a split orthonormalization, where $P_{2s+1} = Q_{2s+1}U_{2s+1}$ and $R_{2s} = Q_{2s}U_{2s}$ are orthonormalized separately. Then, $Q_{4s+1} = \left[Q_{2s+1}, Q_{2s}\right]$ and $U_{4s+1} = \begin{pmatrix} U_{2s+1} & 0 \\ 0 & U_{2s} \end{pmatrix}$ still satisfy the relation

$$AQ_{4s+1}v = Q_{4s+1}H_{4s+1}v$$

where $H_{4s+1} = U_{4s+1}T'U_{4s+1}^{-1}$. But Q_{4s+1} is not orthonormal, only Q_{2s+1} and Q_{2s} are orthonormal. Note that in the derivation of the orthonormalized s-step BiCGStab in Section 2, we only need that Q_{4s+1} is orthonormal for the definition of ω_{m+j}. Moreover, ω_{m+j} is obtained by minimizing the $L2$ norm of r_{m+j+1}. If instead we minimize the B norm of r_{m+j+1}, where B is an $n \times n$ matrix that satisfies $Q_{4s+1}^T B Q_{4s+1} = I_{4s+1}$, then ω_{m+j} would be defined as in Equation (28). We call this version split orthonormalized s-step BiCGStab (Algorithm 3).

Algorithm 4: Split Orthonormalized s-step BiCGStab

Input: A, b, x_0, m_{max}, s, Type of Basis
Output: x_m: the mth approximate solution satisfying the stopping criteria

Let $r_0 = b - Ax_0$, $p_0 = r_0$, $\rho_0 = <r_0, r_0>$, and $k = 0$
Choose $\tilde{r}_0$ such that $\delta_0 =< \tilde{r}_0, r_0 > \neq 0$.
While $\left(\sqrt{\rho_k} > \epsilon ||b||_2 \text{ and } k < \left\lfloor \frac{m_{max}}{s} \right\rfloor\right)$ **Do**
  Compute P_{2s+1} and R_{2s} depending on Type of Basis, and output the diagonals of T'
  Perform the QR factorization of $P_{2s+1} = Q_{2s+1}U_{2s+1}$ and $R_{2s} = Q_{2s}U_{2s}$.
  Let $Q_{4s+1} = [Q_{2s+1}, Q_{2s}]$, $U_{4s+1} = \begin{pmatrix} R_{2s+1} & 0 \\ 0 & R_{2s} \end{pmatrix}$, and $H_{4s+1} = U_{4s+1}T'U_{4s+1}^{-1}$
  Compute $g = Q_{4s+1}^T \tilde{r}$
  Initialize a_0, e_0, c_0 and set $m = k * s$
  for ($j = 0$ to $s - 1$) **Do**
    $h_a = H_{4s+1}a_j$
    $\alpha_{m+j} = \delta_{m+j} / \langle g, h_a \rangle$
    $d = c_j - \alpha_{m+j}h_a$
    $h_d = H_{4s+1}d$
    $\omega_{m+j} = \frac{\langle d, h_d \rangle}{\langle h_d, h_d \rangle}$
    $e_{j+1} = e_j + \alpha_{m+j}a_j + \omega_{m+j}d$
    $c_{j+1} = c_j - \omega_{m+j}h_d - \alpha_{m+j}h_a$
    $\delta_{m+j+1} = \langle g, c_{j+1} \rangle$
    $\beta_{m+j} = (\delta_{m+j+1}/\delta_{m+j})(\alpha_{m+j}/\omega_{m+j})$
    $a_{j+1} = c_{j+1} + \beta_{m+j}a_j - \beta_{m+j}\omega_{m+j}h_a$
  end for
  $p_{m+s} = Q_{4s+1}a_s$, $r_{m+s} = Q_{4s+1}z_s$, $x_{m+s} = x_m + Q_{4s+1}e_s$
  $\rho_{m+s} = \langle r_{m+s}, r_{m+s} \rangle$, $k = k + 1$
end While

For the second problem, starting with a $p_0 \neq r_0$, might improve the convergence of all the previously discussed s-step BiCGStab versions. One might pick a random p_0. However, its effect on the convergence of the method is unknown. Thus, to be consistent with the previously introduced BiCGStab versions, we choose to perform one iteration of BiCGStab before constructing the first $4s + 1$ basis vectors. The advantage is that all the information obtained in the BiCGStab iteration are used afterwards. Algorithm 5 could be considered as a "preprocessing" step, for all the s-step algorithms, where it is performed before the while loop and replacing the first two lines in Algorithm 2, Algorithm 3, or Algorithm 4. We refer to these versions as modified s-step BiCGStab, modified orthonormalized s-step BiCGStab, and modified split orthonormalized s-step BiCGStab.

Algorithm 5: Choosing p_0 not equal to r_0

Let $r_0 = b - Ax_0$, $p_0 = r_0$, $\rho_0 = \langle r_0, r_0 \rangle$, and $k = 0$
Choose $\tilde{r}_0$ such that $\delta_0 = \langle \tilde{r}_0, r_0 \rangle \neq 0$.
$\alpha_0 = \delta_0 / \langle \tilde{r} A p_0 \rangle$4: $s = r_0 - \alpha_0 A p_0$, $t = As$
$\omega_0 = \langle t, s \rangle / \langle t, t \rangle$
$x_0 = x_0 + \alpha_0 p_0 + \omega_0 s$
$r_0 = (I - \omega_0 A)s$
$\beta_0 = \alpha_0/(\omega_0 \delta_0)$
$\delta_0 = \langle \tilde{r}_0, r_0 \rangle$, $\beta_0 = \beta_0 * \delta_0$
$p_0 = r_0 + \beta_0(I - \omega_0 A)p_0$
$\rho_0 = \langle r_0, r_0 \rangle$

3. RESULTS AND EXPECTED PERFORMANCE

In this Section, we show the convergence behavior of the newly introduced split orthonormalized s-step BiCGStab and modified split orthonormalized s-step BiCGStab, on the matrices defined in Section 3.1, and compare them to that of BiCGStab, s-step BiCGStab, and modified s-step BiCGStab in Section 3.2. Then we discuss the split orthonormalized s-step BiCGStab's computation and communication cost in Section 3.3 and compare it to that of s-step BiCGStab and BiCGStab.

3.1 Test Matrices

The study cases presented in this paper are obtained from different representative models for reservoir simulations at different time steps. Table 1 illustrates the characteristics of the models from which the square test matrices, GCS2K, CantaF3, SPE10 [17], and HIS, are generated.

Consequently, the obtained matrices have different profiles and varying degrees of difficulty. Table 2 describes the test matrices.

3.2 Convergence Results

We have implemented the s-step BiCGStab, and the split orthonormalized s-step BiCGStab (Algorithm 4) whereby the monomial bases P_{2s+1} and R_{2s} are first built and then orthonormalized separately using MKL's (Math Kernel Library) QR factorization. We have also implemented the corresponding modified versions, whereby one iteration of BiCGStab is performed before building the first $4s+1$ bases vectors. These algorithms are developed in MCG Solver [18, 19], a C++ based software package developed by *IFPEN* to provide linear solver algorithms for its industrial simulators in reservoir simulation, basin simulation or in engine combustion simulation.

Table 3 shows the convergence results for the s-step BiCGStab versions on the matrices introduced previously with tolerance $tol = 10^{-8}$ except for the SPE10 matrix ($tol = 10^{-4}$). The number of iterations needed for the convergence of s-step BiCGStab versions, referred to as s-Step Iterations (SI), is shown. For comparison reasons, we also show the Total number of Iterations (TI) of the s-step

TABLE 1
The test matrices.

	Block CSR format			
Matrix	Size	Block size	Nonzero blocks	Remark
GCS2K	370 982	3	2 928 696	Well-conditioned
CantaF3	8016	3	65 202	Ill-conditioned
SPE10	1 094 421	2	6 421 171	Ill-conditioned
HIS	34 761	3	189 661	Ill-conditioned

TABLE 2
The reservoir models.

Matrix	Reservoir model	PVT-type	Scheme	Time step
GCS2K	Fractured reservoir (dual-medium)	Bi component	Fully implicit	1
CantaF3	Fractured reservoir (dual-medium)	Multi component	IMPEX	11
SPE10	Classical 2 phase flow model	Black oil	Fully implicit	81
HIS	Classical 3 phase flow model	Black oil	Fully implicit	23

BiCGStab versions, which is equal to the *s*-step iterations times *s*. Note that in exact arithmetics, each *s*-step BiCGStab iteration is equivalent to *s* iterations of the classical BiCGStab method. However, in terms of computation and communication cost, they are not equivalent, as discussed in the next section.

For the well-conditioned matrices such as GCS2K, *s*-step BiCGStab with monomial basis converges faster as *s* increases from 2 to 6, and the corresponding total iterations are in the same range as that of BiCGStab. However, this is not the case for the other ill-conditioned matrices. The *s*-step BiCGStab's convergence is chaotic with respect to *s* and for some *s* values it requires more total iterations to converge than BiCGStab. As mentioned earlier, using a more stable basis such as the Newton basis could improve the convergence of the *s*-step BiCGStab for the ill-conditioned matrices. However, for the ill-conditioned matrices ARPACK was not able to find the requested number of eigenvalues in 3000 iterations, which is time consuming. Thus, we orthonormalize the bases for better numerical stability.

In the case of well-conditioned matrices such GCS2K, the computed monomial basis is already numerically stable. Thus, it is expected that the convergence will not improve much by orthonormalizing the basis or by starting with a $p_0 \neq r_0$. This is clear in Table 3, where the convergence of split orthonormalized *s*-step BiCGStab, modified *s*-step BiCGStab, and modified split orthonormalized *s*-step BiCGStab is in the same range as that of *s*-step BiCGStab.

On the other hand, the convergence behavior of the *s*-step BiCGStab methods for the ill-conditioned matrices varies. For SPE10, *s*-step BiCGStab converges in fewer *s*-step iterations, as *s* increases from 2 to 6. Then, orthonormalizing the basis and/or starting with $p_0 \neq r_0$ improves the numerical stability of the basis, leading to a better convergence. Note that orthonormalizing the basis (split ortho *s*-step BiCGStab) has a larger effect on convergence than starting with a $p_0 \neq r_0$ (modified *s*-step BiCGStab).

For the HIS matrix, *s*-step BiCGStab's convergence fluctuates as *s* increases. Whereas, the other *s*-step BiCGStab methods have a strictly decreasing convergence with respect to *s*. Starting with $p_0 \neq r_0$ (modified *s*-step BiCGStab) improved the convergence of *s*-step BiCGStab (except for $s = 3$). Moreover, the convergence results of modified *s*-step BiCGStab and modified split orthonormalized *s*-step BiCGStab are very similar for $s \geq 4$.

CantaF3 is a special case where the modified split orthonormalized *s*-step BiCGStab method was the only method that had a stable convergence as *s* increased from 2 to 5. All the other *s*-step BiCGStab methods had a chaotic convergence with respect to *s*.

Note that in some cases the total iterations of the *s*-step BiCGStab variants is more than that of BiCGStab. However, the communication cost of each *s*-step iteration is much less than that of *s* iterations of BiCGStab. Thus the *s*-step variants should still converge faster, in terms of runtime, in parallel implementations.

In general, we can say that for well-conditioned matrices, the *s*-step BiCGStab methods have a similar rate of convergence. However, for the ill-conditioned matrices, orthonormalizing the bases separately and starting with a $p_0 \neq r_0$ have positive effects on the stability of the basis which speeds up and stabilizes the convergence with respect to *s*.

As mentioned earlier, several linear system with different degrees of difficulty are solved throughout a basin or reservoir simulation using the same method (BiCGStab). Our goal is to speedup the convergence of BiCGStab by replacing it with an *s*-step version that reduces communication. Based on the presented convergence results, the modified split orthonormalized *s*-step BiCGStab method seems to be the most stable version with respect to *s* for both, well-conditioned and ill-conditioned matrices. The reason we focus on the convergence behavior as *s* increases, rather

TABLE 3

The convergence of the *s*-step BiCGStab, split orthonormalized *s*-step BiCGStab and their modified versions with monomial basis, for different *s* values, compared to BiCGStab. The *s*-Step Iterations (SI) and the Total Iterations (TI) are shown. The number of total iterations is equal to the number of *s*-step iterations times *s*.

Matrix	BiCGStab	*s*	*s*-Step		Split ortho *s*-step		Modified *s*-step		Modified split ortho *s*-step	
	TI		SI	TI	SI	TI	SI	TI	SI	TI
GCS2K	193	**2**	99	198	91	182	89	178	90	180
		3	61	183	62	186	59	177	61	183
		4	45	180	47	188	46	184	50	200
		5	41	205	41	205	38	190	38	190
		6	34	204	33	198	36	216	35	210
CantaF3	4472	**2**	2813	5626	1266	2532	1980	3960	1566	3132
		3	1065	3195	1240	3720	1924	5772	1013	3039
		4	5510	22 040	801	3204	4824	19 296	947	3788
		5	1228	6140	3612	18 060	2658	13 290	774	3870
		6	662	3972	1079	6474	623	3738	1072	6432
SPE10	4744	**2**	2438	4876	1682	3364	2144	4288	1829	3658
		3	1756	5268	1345	4035	1712	5136	1200	3600
		4	1215	4860	752	3008	1137	4548	862	3448
		5	1032	5160	752	3760	1021	5105	567	2835
		6	758	4548	506	3036	856	5136	548	3288
HIS	4721	**2**	1867	3734	1778	3556	1721	3442	2571	5142
		3	1249	3747	1288	3864	1568	4704	1158	3474
		4	1631	6524	1213	4852	1107	4428	1100	4400
		5	1013	5065	1007	5035	808	4040	869	4345
		6	699	4194	624	3744	685	4110	676	4056

TABLE 4
The number of flops performed in one iteration of the sequential (modified) split orthonormalized *s*-step BiCGStab and *s*-step BiCGStab with monomial basis, and the corresponding flops for computing s iterations of BiCGStab.

	BiCGStab	*s*-Step BiCGStab	Split ortho *s*-step BiCGStab
Flops	$4\ \mathrm{nnz}s + 20\ ns$	$(8\ s-2)\ \mathrm{nnz} + 32\ ns^2 + 44\ sn + 11\ n + 64\ s^3 + 96\ s^2 + 16\ s - 2$	$(8\ \mathrm{s}-2)\ \mathrm{nnz} + 16\ ns^2 + 32\ sn + 11\ n + \frac{32}{3}s^3 + \frac{112}{3}s^2 + \frac{308}{3}s - \frac{64}{3}$

than the convergence behavior for a given *s* value, is that the *s* value is fixed throughout the simulation. And the convergence effect of the chosen *s* value on the different linear systems is not known beforehand. Thus, we seek a robust method that will converge faster as *s* increases to some upper limit (5 or 6).

3.3 Computation and Communication Cost

In Table 4, the number of flops performed in one *s*-step iteration of orthonormalized *s*-step BiCGStab and *s*-step BiCGStab is presented, along with the number of flops performed in *s* iterations of BiCGStab.

In the (modified) split orthonormalized *s*-step BiCGStab there is a need for performing two QR factorizations of the matrices P_{2s+1} and R_{2s}, however Gram-like matrix *G* is not computed. Thus, the computed flops in the (modified) split orthonormalized *s*-step BiCGStab is slightly less than the computed flops in the *s*-step BiCGStab as shown in Table 4. As discussed in [5], the only communication that occurs in the parallel implementation of *s*-step BiCGStab is in the construction of the basis and the matrix *G*. Similarly for the parallel implementation of the (modified) split orthonormalized *s*-step BiCGStab, only the construction of the basis and its orthonormalization using TSQR require communication. Note that both TSQR and the computation of *G* require $\log(p)$ messages and sending $O((4s+1)^2 \log(p))$ words where *p* is the number of processors. Thus in terms of computed flops, sent messages and sent words, the (modified) split orthonormalized *s*-step BiCGStab and the *s*-step BiCGStab are equivalent. Note that the only difference between the modified split orthonormalized *s*-step BiCGStab and the split orthonormalized *s*-step BiCGStab is that Algorithm 5 is called once before the while loop. Hence, we may assume that the communication and computation cost of both methods is bounded by the same value.

On the other hand, the performed flops in one iteration of *s*-step BiCGStab and orthonormalized *s*-step BiCGStab is at least twice the flops performed in *s* iterations of BiCGStab. However, the number of sent messages in the *s*-step versions is reduced by a factor of $O(s)$, at the expense of increasing the number of sent words. Therefore, it is expected to obtain speedup in the parallel implementations of the introduced *s*-step BiCGStab variants.

CONCLUSION

In this paper, we have introduced the split orthonormalized *s*-step BiCGStab and the modified split orthonormalized *s*-step BiCGStab, variants of *s*-step BiCGStab where the basis vectors are orthonormalized. In addition, in the modified split orthonormalized *s*-step BiCGStab, we perform one iteration of BiCGStab to define a p_0 not equal to r_0.

We have studied the convergence behavior of the introduced methods with monomial basis, and compared it to that of the *s*-step BiCGStab for the matrices obtained from reservoir simulations. For almost all the tested matrices, the modified split orthonormalized *s*-step BiCGStab with monomial basis, converged faster than the *s*-step BiCGStab for $s = 2, \ldots, 6$. Moreover, for ill-conditioned matrices, the modified split orthonormalized *s*-step BiCGStab has a similar convergence behavior as the BiCGStab method for $s = 2, \ldots, 6$, unlike the *s*-step BiCGStab.

All the *s*-step BiCGStab versions send $O(s)$ times less messages than *s* iterations of BiCGStab. Moreover, the computation cost of the introduced variants is slightly less than that of the *s*-step BiCGStab. Hence, it is expected that the introduced *s*-step BiCGStab methods, specifically modified split orthonormalized *s*-step BiCGStab, will perform well in parallel on multi-core architectures.

As a future work, we would like to implement the split orthonormalized *s*-step BiCGStab and the modified split orthonormalized *s*-step BiCGStab, in parallel and compare its runtime to that of the parallel BiCGStab and *s*-step BiCGStab.

REFERENCES

1 Saad Y., Schultz M.H. (1986) Gmres: a generalized minimal residual algorithm for solving nonsymmetric linear systems, *SIAM J. Sci. Stat. Comput.* **7**, 3, 856-869.

2 Hestenes M.R., Stiefel E. (1952) Methods of conjugate gradients for solving linear systems, *J. Res. Natl. Bur. Stand.* **49**, 409-436.

3 van der Vorst H.A. (1992) Bi-CGSTAB: a fast and smoothly converging variant of Bi-CG for the solution of nonsymmetric linear systems, *SIAM J. Sci. Statist. Comput.* **13**, 2, 631-644.

4 Carson E., Knight N., Demmel J. (2011) Avoiding communication in two-sided Krylov subspace methods. *Technical Report UCB/EECS-2011-93*, EECS Department, University of California, Berkeley, August.

5 Carson E., Knight N., Demmel J. (2013) Avoiding communication in nonsymmetric Lanczos-based Krylov subspace methods, *SIAM J. Sci. Comput.* **35**, 5, S42-S61.

6 Grigori L, Moufawad S. (2013) Communication avoiding ILU0 preconditioner, *Technical Report*, ALPINES - INRIA Paris-Rocquencourt, March.

7 Hoemmen M. (2010) Communication-avoiding Krylov subspace methods, *PhD Thesis*, EECS Department, University of California, Berkeley.

8 Mohiyuddin M., Hoemmen M., Demmel J., Yelick K. (2009) Minimizing communication in sparse matrix solvers, *In Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis, SC'09*, New York, NY, USA, ACM, pp.1-12.

9 Chronopoulos A.T., Gear W. (1989) *s*-Step iterative methods for symmetric linear systems, *J. Comput. Appl. Math.* **25**, 2, 153-168.

10 Erhel J. (1995) A parallel GMRES version for general sparse matrices, *Electron. Trans. Numer. Anal.* **3**, 160-176.

11 Walker H.F. (1988) Implementation of the GMRES method using householder transformations, *SIAM J. Sci. Statist. Comput.* **9**, 1, 152-163.

12 Demmel J., Hoemmen M., Mohiyuddin M., Yelick K. (2008) Avoiding communication in sparse matrix computations, In *Parallel and Distributed Processing, 2008. IPDPS 2008. IEEE International Symposium*, 14–18 April 2008, Held in Hyatt Regency Hotel in Miami, Florida, USA, pp. 1-12.

13 Demmel J., Grigori L., Hoemmen M., Langou J. (2012) Communication-avoiding parallel and sequential QR factorizations, *SIAM J. Sci. Comput.* **34**, 206-239.

14 Bai Z., Hu D., Reichel L. (1994) A Newton basis GMRES implementation, *IMA J. Numer. Anal.* **14**, 563-581.

15 Reichel L. (1990) Newton interpolation at Leja points, *BIT Numerical Mathematics* **30**, 332-346.

16 Lehoucq R., Sorensen D., Yang C. (1998) *ARPACK Users' Guide*, Society for Industrial and Applied Mathematics, Philadelphia, PA.

17 The 10th SPE Comparative Solution Project (2000) Retrieved from http://www.spe.org/web/csp/datasets/set02.htm.

18 Anciaux-Sedrakian A., Eaton J., Gratien J., Guignon T., Havé P., Preux C., Ricois O. (2015) Will GPGPUs be finally a credible solution for industrial reservoir simulators, *SPE Reservoir Simulation Symposium*, 23-25 February, Houston, Texas, USA, SPE-173223-MS. DOI: 10.2118/173223-MS.

19 Anciaux-Sedrakian A., Gottschling P., Gratien J., Guignon T. (2014) Survey on efficient linear solvers for porous media flow models on recent hardware architectures, *Oil Gas Sci. Technol. - Rev. IFP* **69**, 4, 753-766.

Manuscript submitted in December 2015
Manuscript accepted in October 2016
Published online in December 2016

Cite this article as: A. Anciaux-Sedrakian, L. Grigori, S. Moufawad and S. Yousef (2016). *S*-Step BiCGStab Algorithms for Geoscience Dynamic Simulations, *Oil Gas Sci. Technol* **71**, 66.