SIAM J. Sci. Comput.
Vol. 42, No. 1, pp. A187–A219

s-STEP ENLARGED KRYLOV SUBSPACE CONJUGATE GRADIENT METHODS[*]

SOPHIE M. MOUFAWAD[†]

Abstract. Recently, enlarged Krylov subspace methods, which consist of enlarging the Krylov subspace by a maximum of t vectors per iteration based on the domain decomposition of the graph of A, were introduced with the aim of reducing communication when solving systems of linear equations $Ax = b$. In this paper, the s-step enlarged Krylov subspace conjugate gradient methods are introduced, whereby s iterations of the enlarged conjugate gradient methods are merged in one iteration. The numerical stability of these s-step methods is studied, and several numerically stable versions are proposed. Similarly to the enlarged Krylov subspace methods, the s-step enlarged Krylov subspace methods have a faster convergence than Krylov methods, in terms of iterations. Moreover, by computing st basis vectors of the enlarged Krylov subspace $\mathcal{K}_{k,t}(A, r_0)$ at the beginning of each s-step iteration, communication is further reduced. It is shown in this paper that the introduced methods are parallelizable with less communication, with respect to their corresponding enlarged versions and to conjugate gradient.

Key words. linear algebra, iterative methods, Krylov subspace methods, high performance computing, minimizing communication

AMS subject classifications. 65F10, 68W10

DOI. 10.1137/18M1182528

1. Introduction.

Recently, enlarged Krylov subspace methods [11, 22] were introduced with the aim of obtaining methods that converge faster than classical Krylov methods and are parallelizable with less communication. "Communication" here means data movement between different levels of memory hierarchy and different processors over a network. Different methods and techniques have been previously introduced for reducing communication in Krylov subspace methods, such as conjugate gradient (CG) [16], generalized minimal residual (GMRES) [26], bi-conjugate gradient [19, 8], and bi-conjugate gradient stabilized [27]. The interest in such methods is due to the communication bottleneck on modern-day computers, where performing arithmetic operations is rather fast but the gap between communication and computation is expected to keep increasing due to technological reasons. Moreover, Krylov subspace methods are governed by BLAS 1 and BLAS 2 operations that are communication-bound.

These methods and techniques can be categorized depending on how the communication reduction is achieved. It is possible to hide the cost of communication by overlapping it with other computations and communication, like pipelined CG [5, 13] and pipelined GMRES [9]. However, most of the algorithms replace BLAS1 and BLAS2 operations by denser operations that may be parallelized more efficiently with less communication. This can be achieved by introducing new methods based on different Krylov subspaces, such as augmented Krylov methods [3, 25], and the block Krylov methods [23] that are based on augmented and block Krylov subspaces, respectively. The recently introduced enlarged Krylov subspace methods [11, 22] fall into this category since the methods search for the approximate solution in the

[*]Submitted to the journal's Methods and Algorithms for Scientific Computing section April 23, 2018; accepted for publication (in revised form) October 7, 2019; published electronically January 9, 2020.
https://doi.org/10.1137/18M1182528

[†]Department of Mathematics, American University of Beirut (AUB), Beirut, Lebanon (sm101@aub.edu.lb).

enlarged Krylov subspace. Another way is to restructure the Krylov methods algorithms themselves to reduce communication, such as the s-step and communication avoiding methods. The s-step methods [28, 4, 29, 6, 2] compute s basis vectors per iteration and use them to update the next approximate solution. Moreover, the communication avoiding methods [21, 17, 10], which are based on s-step methods, introduce communication avoiding kernels that further reduce communication, even if at the expense of performing redundant computations.

In this paper, we introduce the s-step enlarged Krylov subspace methods, whereby s iterations of the enlarged Krylov subspace methods are merged into one iteration. The idea of s-step methods is not new, as mentioned previously. However, the aim of this work is the introduction of methods that further reduce communication with respect to the enlarged Krylov methods and eventually the classical Krylov methods. In terms of iterations, the s-step enlarged Krylov subspace methods have a faster convergence than classical Krylov methods, similarly to the enlarged Krylov subspace methods. In addition, computing st basis vectors of the enlarged Krylov subspace $\mathcal{K}_{k,t}(A, r_0)$ at the beginning of each s-step enlarged Krylov subspace iteration reduces the number of sent messages in a distributed memory architecture, compared to s iterations of the corresponding enlarged Krylov method. This reduces the global communication latency in a distributed memory architecture, even if at the expense of slightly increasing the arithmetic operations.

We introduce several s-step enlarged CG versions, based on the short recurrence enlarged CG methods (SRE-CG and SRE-CG2) and MSDO-CG presented in [11]. After briefly introducing CG, a Krylov projection method for symmetric (Hermitian) positive definite matrices, and the enlarged CG methods in section 2, we discuss the new s-step enlarged CG versions (section 3) in terms of numerical stability (section 4), preconditioning (section 5), and communication reduction in parallel (section 6). We only consider in this article a distributed memory system, where we reduce the number of messages sent between different processors. However, the introduced methods reduce communication even in shared memory systems, i.e., reduce the data movement between main memory and cache. Finally we conclude in section 7.

2. From conjugate gradient to enlarged conjugate gradient methods.

The CG method of Hestenes and Stiefel [16] was introduced in 1952. Since then, different CG versions have been introduced for different purposes. In 1980, O'Leary introduced the block CG method [23] for solving a symmetric positive definite system with multiple right-hand sides. Block CG, when solving multiple linear systems at the same time, performs less work than solving each system separately with CG. In addition, it may converge faster in terms of iterations and time in some cases discussed in [23]. In 1989, Chronopoulos and Gear introduced the s-step CG method [4], which performs s CG iterations simultaneously with the goal of reducing communication by performing more flops using the data in fast memory. Several CG versions were introduced for solving successive linear systems with different right-hand sides, by recycling the computed Krylov subspace, such as [7]. Moreover, several preconditioned and parallelizable CG versions were introduced, such as deflated CA-CG [1], MSD-CG [14], and augmented CG [3, 25]. Recently, enlarged CG methods such as SRE-CG, SRE-CG2, and MSDO-CG were introduced [11]. In this section, we briefly discuss CG, s-step CG, block CG, and enlarged CG versions. For a brief overview of other related CG versions, such as coop-CG and MSD-CG, refer to [22].

The CG method is a Krylov projection method that finds a sequence of approximate solutions $x_k \in x_0 + \mathcal{K}_k(A, r_0)$ $(k > 0)$ of the system $Ax = b$, by imposing the Petrov–Galerkin condition, $r_k \perp \mathcal{K}_k$, where $\mathcal{K}_k(A, r_0) = \text{span}\{r_0, Ar_0, A^2r_0, \ldots, A^{k-1}r_0\}$ is the Krylov subspace of dimension k, x_0 is the initial iterate, and r_0 is the initial residual. At the kth iteration, CG computes the new approximate solution $x_k = x_{k-1} + \alpha_k p_k$ that minimizes $\phi(x) = \frac{1}{2}x^tAx - b^tx$ over the corresponding space $x_0 + \mathcal{K}_k(A, r_0)$, where $k > 0$, $p_k = r_{k-1} + \beta_k p_{k-1} \in \mathcal{K}_k(A, r_0)$ is the kth search direction, $p_1 = r_0$, and $\alpha_k = \frac{(p_k)^t r_{k-1}}{(p_k)^t A p_k} = \frac{||r_{k-1}||_2^2}{||p_k||_A^2}$ is the step along the search direction. As for $\beta_k = -\frac{(r_{k-1})^t A p_{k-1}}{(p_{k-1})^t A p_{k-1}} = \frac{||r_{k-1}||_2^2}{||r_{k-2}||_2^2}$, it is defined so that the search directions are A-orthogonal ($p_k^t A p_i = 0$ for all $i \neq k$), since otherwise the Petrov–Galerkin condition is not guaranteed.

The s-step CG method [4] introduced by Chronopoulos and Gear in 1989 is also a Krylov projection method that solves the system $Ax = b$ by imposing the Petrov–Galerkin condition. However, it finds a sequence of approximate solutions $x_k \in x_0 + \mathcal{K}_{sk}(A, r_0)$, where $k > 0$, $s > 0$, and $\mathcal{K}_{sk}(A, r_0) = \text{span}\{r_0, Ar_0, A^2r_0, \ldots, A^{sk-2}r_0, A^{sk-1}r_0\}$. At the kth iteration, $x_k = x_{k-1} - P_k\alpha_k$ is obtained by minimizing $\phi(x)$, where P_k is a matrix containing the s search directions and $\alpha_k = ((P_k)^t A P_k)^{-1} P_k^t r_{k-1}$ is a vector containing the s corresponding step lengths. Initially, P_k is defined as the first s basis vectors of the Krylov subspace, i.e., $P_1 = R_0 = [r_0 Ar_0 \cdots A^{s-1}r_0]$. For $k > 1$, $P_k = R_{k-1} + P_{k-1}\beta_k$, where $R_{k-1} = [r_{k-1}\ Ar_{k-1}\ \cdots\ A^{s-1}r_{k-1}]$, and $\beta_k = -(P_{k-1}^t A P_{k-1})^{-1}(R_{k-1}^t P_{k-1})$ is an $s \times s$ matrix.

The block CG methods [23] introduced by O'Leary in 1980 solve an $n \times n$ symmetric positive definite system with m right-hand sides $AX = B$, where the $n \times m$ block residual $R_k = B - AX_k$ is orthogonal to the block Krylov subspace $\mathcal{K}_k^\square(A, R_0) = \left\{ \sum_{j=0}^{k-1} A^j R_0 \gamma_j \right\}$ with γ_j being an $m \times m$ matrix. At the kth iteration, the $n \times m$ block approximate solution $X_k = X_{k-1} - P_k\alpha_k$ is obtained by minimizing $\phi(x)$, where P_k is an $n \times m$ matrix and α_k is an $m \times m$ matrix.

On the other hand, the enlarged CG methods introduced in [22, 11] are enlarged Krylov projection methods that find a sequence of approximate solutions $x_k \in x_0 + \mathcal{K}_{k,t}(A, r_0)$ ($k > 0$) of the system $Ax = b$, by imposing the Petrov–Galerkin condition, $r_k \perp \mathcal{K}_{k,t}(A, r_0)$, where

$$\begin{aligned}
\mathcal{K}_{k,t}(A, r_0) &= \text{span}\{T(r_0), AT(r_0), A^2T(r_0), \ldots, A^{k-1}T(r_0)\} \\
&= \text{span}\{T_1(r_0), T_2(r_0), \ldots, T_t(r_0), \\
&\qquad AT_1(r_0), AT_2(r_0), \ldots, AT_t(r_0), \\
&\qquad \vdots \\
&\qquad A^{k-1}T_1(r_0), A^{k-1}T_2(r_0), \ldots, A^{k-1}T_t(r_0)\}
\end{aligned}$$

is the enlarged Krylov subspace of dimension at most tk, x_0 is the initial iterate vector, r_0 is the initial residual vector, and $T(r_0) = \{T_1(r_0), T_2(r_0), \ldots, T_t(r_0)\}$ is an operator that splits r_0 into t vectors based on a domain decomposition of matrix A, with $T_i(r_0)$ being the operator that projects vector r_0 on domain i of matrix A. Several enlarged CG versions were introduced in [11], such as MSDO-CG, LRE-CG, SRE-CG, SRE-CG2, and the truncated SRE-CG2.

The enlarged CG versions perform block operations. But they are not block methods since a system with one right-hand side is solved where the residual vector $r_k = b - Ax_k$ is not orthogonal to the enlarged Krylov subspace, hence the need for A-orthonormalizing the basis vectors. Note that in [12] a block variant of SRE-CG is proposed, whereby the number of search directions per iteration is reduced using deflation. The methods introduced in [12] are block CG methods that solve one system with multiple right-hand sides where R_0 is defined using the enlarged Krylov subspace, i.e., $R_0 = \mathcal{T}_0$ is the matrix whose columns are the vectors of the set $T(r_0)$, and the solution of $Ax = b$ is the sum of the block solution X_k.

3. s-step enlarged CG versions.

The aim of s-step enlarged CG methods is to merge s iterations of enlarged CG methods and perform more flops per communication, in order to reduce communication as compared to enlarged CG.

In the case of the SRE-CG and SRE-CG2 versions, reformulating into s-step versions is straightforward since these methods build an A-orthonormal basis for $\mathcal{K}_{k,t}(A, r_0) = \{T(r_0), AT(r_0), \ldots, A^{k-1}T(r_0)\}$ and update the approximate solutions x_k. The basis construction is independent from the consecutive approximate solutions. But the challenge is in constructing a numerically stable A-orthonormal basis of st vectors, where t is the number of domains and s is the number of merged iterations.

As for MSDO-CG, at each iteration k, t search directions are built and A-orthonormalized and used to update the approximate solution. Moreover, the construction of the search directions depends on the previously computed approximate solution. So merging s iterations of the MSDO-CG algorithm requires more work, since it is not possible to separate the search directions construction from the solution's update. Hence, a new version will be proposed where a modified enlarged Krylov subspace is built.

3.1. s-step SRE-CG and SRE-CG2.

The short recurrence enlarged CG (SRE-CG) and SRE-CG2 methods are iterative enlarged Krylov subspace projection methods that build at the kth iteration, an A-orthonormal candidate basis Q_k ($Q_k^t A Q_k = I$) for the enlarged Krylov subspace

$$\mathcal{K}_{k,t} = \operatorname{span}\left\{T(r_0), AT(r_0), \ldots, A^{k-1}T(r_0)\right\},$$

and approximate the solution, $x_k = x_{k-1} + Q_k\alpha_k$, by imposing the orthogonality condition on $r_k = r_{k-1} - AQ_k\alpha_k$, ($r_k \perp \mathcal{K}_{k,t}$), and minimizing

$$\phi(x) = \frac{1}{2}x^t Ax - x^t b,$$

over $x_0 + \mathcal{K}_{k,t}$, where Q_k is an $n \times kt$ matrix and $T(r_0)$ is the set of t vectors obtained by projecting r_0 on the t distinct domains of A.

There are two phases in these methods: building the candidate A-orthonormal basis and updating the approximate solution. The difference between SRE-CG and SRE-CG2 is in the basis construction. First, $W_1 = \mathcal{T}_0$ is A-orthonormalized using A-CholQR [24] or PreCholQR [20, 22], where $\mathcal{T}_0$ is the matrix containing the t vectors of $T(r_0)$, i.e., $\mathcal{T}_0 = [T(r_0)] = [T_1(r_0)\, T_2(r_0) \cdots T_t(r_0)]$ with $T_i(.)$ being the operator that projects a vector on domain i of matrix A. Then, it is shown in [11] that at each iteration $k \geq 3$, $W_k = AW_{k-1}$ has to be A-orthonormalized only against W_{k-1} and W_{k-2} using the CGS2 A-orthonormalization method (Algorithm 18 in [22]) and then against itself using A-CholQR [24] or Pre-CholQR [20, 22]. Finally, the approximate solution x_k and the residual r_k are updated, $x_k = x_{k-1} + W_k\alpha_k$ and $r_k = r_{k-1} - AW_k\alpha_k$, where $\alpha_k = W_k^t r_{k-1}$. This is the SRE-CG method.

However, in finite arithmetic there might be a loss of A-orthogonality at the kth iteration between the vectors of $Q_k = [W_1, W_2, \ldots, W_k]$. Hence, in SRE-CG2 $W_k = AW_{k-1}$ is A-orthonormalized against all W_i's for $i = 1, 2, \ldots, k-1$.

The construction of the W_k matrix is independent from updating the approximate solution x_k. Thus it is possible to restructure the SRE-CG and SRE-CG2 algorithms by first computing W_1, $W_2, \ldots, W_s$ and then updating x_1, $x_2, \ldots, x_s$ as shown in Algorithms 1 and 2.

Algorithm 1. Restructured SRE-CG.

```
    Input: $A$, $n \times n$ symmetric positive definite matrix; $k_{max}$, maximum allowed iterations; $b$, $n \times 1$ right-hand side; $x_0$, initial guess; $\epsilon$, stopping tolerance; $s$, s-step
    Output: $x_k$, approximate solution of the system $Ax = b$
 1: $r_0 = b - Ax_0$, $\rho_0 = ||r_0||_2$ , $\rho = \rho_0$, $k = 1$;
 2: while ( $\rho > \epsilon\rho_0$ and $k < k_{max}$ ) do
 3:     if ($k == 1$) then
 4:         A-orthonormalize $W_k = \mathfrak{T}_0$
 5:     else
 6:         A-orthonormalize $W_k = AW_{k-1}$ against $W_{k-2}$ and $W_{k-1}$
 7:         A-orthonormalize $W_k$
 8:     end if
 9:     for ($i = 1 : s - 1$) do
10:         A-orthonormalize $W_{k+i} = AW_{k+i-1}$ against $W_{k+i-2}$ and $W_{k+i-1}$
11:         A-orthonormalize $W_{k+i}$
12:     end for
13:     for ($i = k : k + s - 1$) do
14:         $\tilde{\alpha} = W_i^t r_{i-1}$
15:         $x_i = x_{i-1} + W_i\tilde{\alpha}$
16:         $r_i = r_{i-1} - AW_i\tilde{\alpha}$
17:     end for
18:     $k = k + s$,    $\rho = ||r_{k-1}||_2$
19: end while
```

Algorithm 2. Restructured SRE-CG2.

```
    Input: $A$, $n \times n$ symmetric positive definite matrix; $k_{max}$, maximum allowed iterations; $b$, $n \times 1$ right-hand side; $x_0$, initial guess; $\epsilon$, stopping tolerance; $s$, s-step
    Output: $x_k$, approximate solution of the system $Ax = b$
 1: $r_0 = b - Ax_0$, $\rho_0 = ||r_0||_2$ , $\rho = \rho_0$, $k = 1$;
 2: while ( $\rho > \epsilon\rho_0$ and $k < k_{max}$ ) do
 3:     if ($k == 1$) then
 4:         A-orthonormalize $W_k = \mathfrak{T}_0$, and let $Q = W_k$
 5:     else
 6:         A-orthonormalize $W_k = AW_{k-1}$ against $Q$
 7:         A-orthonormalize $W_k$ and let $Q = [Q\ W_k]$
 8:     end if
 9:     for ($i = 1 : s - 1$) do
10:         A-orthonormalize $W_{k+i} = AW_{k+i-1}$ against $Q$
11:         A-orthonormalize $W_{k+i}$ and let $Q = [Q\ W_{k+i}]$
12:     end for
13:     for ($i = k : k + s - 1$) do
14:         $\tilde{\alpha} = W_i^t r_{i-1}$
15:         $x_i = x_{i-1} + W_i\tilde{\alpha}$
16:         $r_i = r_{i-1} - AW_i\tilde{\alpha}$
17:     end for
18:     $k = k + s$,    $\rho = ||r_{k-1}||_2$
19: end while
```

The advantage of such reformulations (Algorithms 1 and 2) is that matrix A is fetched once from memory per construction of st A-orthonormal vectors, as opposed to fetched s times in the SRE-CG and SRE-CG2 algorithms. However, the number of messages and words sent in parallel is unchanged since the two corresponding algorithms perform the same operations but in a different order.

To reduce communication the inner for loops have to be replaced with a set of denser operations. Lines 3–12 of Algorithm 2 can be viewed as a block Arnoldi A-orthonormalization procedure, whereas lines 4–13 of Algorithm 1 can be viewed as a truncated block Arnoldi A-orthonormalization procedure. As for the second loop, by updating x_k and r_k once, we obtain an s-step version.

At the kth iteration of an s-step enlarged CG method, st new basis vectors of $\mathcal{K}_{ks,t} = \operatorname{span}\{T(r_0), AT(r_0), \ldots, A^{sk-1}T(r_0)\}$ are computed and stored in V_k, an $n \times st$ matrix. Since

$$\mathcal{K}_{ks,t} = \mathcal{K}_{(k-1)s,t} + \operatorname{span}\left\{A^{s(k-1)}T(r_0), A^{s(k-1)+1}T(r_0), \ldots, A^{sk-1}T(r_0)\right\},$$

then $Q_{ks} = [Q_{(k-1)s}, V_k]$, where $Q_{(k-1)s}$ is an $n \times (k-1)st$ matrix that contains the $(k-1)st$ vectors of $\mathcal{K}_{(k-1)s,t}$, and Q_{ks} is an $n \times kst$ matrix.

Then, $x_k = x_{k-1} + Q_{ks}\alpha_k \in \mathcal{K}_{ks,t}$, where $\alpha_k = (Q_{ks}^t A Q_{ks})^{-1}(Q_{ks}^t r_{k-1})$ is defined by minimizing $\phi(x)$ over $x_0 + \mathcal{K}_{ks,t}$. As a consequence, $r_k = b - Ax_k = r_{k-1} - AQ_{ks}\alpha_k \in \mathcal{K}_{(k+1)s,t}$ satisfies the Petrov–Galerkin condition $r_k \perp \mathcal{K}_{ks,t}$, i.e., $r_k^t y = 0$ for all $y \in \mathcal{K}_{ks,t}$.

In the s-step SRE-CG2 version, Q_{ks} is A-orthonormalized ($Q_{ks}^t A Q_{ks} = I$), then

$$\alpha_k = \left(Q_{ks}^t A Q_{ks}\right)^{-1}\left(Q_{ks}^t r_{k-1}\right) = Q_{ks}^t r_{k-1}.$$

But $r_{k-1} \perp \mathcal{K}_{(k-1)s,t}$, i.e., $r_{k-1}^t y = 0$ for all $y \in \mathcal{K}_{(k-1)s,t}$. Thus,

$$\begin{aligned}
\alpha_k &= Q_{ks}^t r_{k-1} = \left[Q_{(k-1)s}, V_k\right]^t r_{k-1} = \left[0_{(k-1)st\times n}; V_k^t r_{k-1}\right],\\
x_k &= x_{k-1} + Q_{ks}\alpha_k = x_{k-1} + \left[Q_{(k-1)s}, V_k\right]\left[0_{(k-1)st\times n}; V_k^t r_{k-1}\right]\\
&= x_{k-1} + V_k V_k^t r_{k-1} = x_{k-1} + V_k\tilde{\alpha}_k,\\
\tilde{\alpha}_k &= V_k^t r_{k-1}\\
\Longrightarrow\ r_k &= r_{k-1} - AV_k\tilde{\alpha}_k.
\end{aligned}$$

In Algorithm 4, the st new vectors are computed similarly to Algorithm 2, where t vectors are computed at a time (W_j), A-orthonormalized against all the previously computed vectors using the CGS2 A-orthonormalization method [22], and finally A-orthonormalized using A-CholQR [24] or Pre-CholQR [20, 22]. At the kth s-step iteration, all the kst vectors have to be stored in Q_{ks}.

Note that in exact arithmetic, at the kth s-step iteration, the A-orthonormalization of W_j for $j \geq (k-1)s+1$, against $\tilde{Q} = [W_1, W_2, W_3, \cdots W_{j-1}]$ can be summarized as follows, where $Q_{(k-1)s} = [W_1, W_2, W_3, \cdots W_{(k-1)s}]$,

$$\begin{aligned}
W_j &= AW_{j-1} - \tilde{Q}\tilde{Q}^t A\left(AW_{j-1}\right)\\
&= AW_{j-1} - \tilde{Q}\left[W_1^t A\left(AW_{j-1}\right); W_2^t A\left(AW_{j-1}\right); \cdots; W_{j-1}^t A\left(AW_{j-1}\right)\right]\\
&= AW_{j-1} - \tilde{Q}\left[0; 0; \cdots; 0; W_{j-2}^t A\left(AW_{j-2}\right); W_{j-1}^t A\left(AW_{j-1}\right)\right]\\
&= AW_{j-1} - W_{j-1}W_{j-1}^t A\left(AW_{j-1}\right) - W_{j-2}W_{j-2}^t A\left(AW_{j-1}\right),
\end{aligned}$$

since $(AW_i)^t AW_{j-1} = 0$ for all $i < j-2$ by the A-orthonormalization process. This version (Algorithm 3) is called the s-step short recurrence enlarged conjugate gradient (s-step SRE-CG), where only the last zt computed vectors ($z = \max(s, 3)$) are stored, and every t vectors W_j are A-orthonormalized against the previous $2t$ vectors W_{j-2} and W_{j-1} for $j > 2$. As for x_k and r_k, they are defined as in the s-step SRE-CG2 method.

For $s = 1$, Algorithms 3 and 4 are reduced to the SRE-CG and SRE-CG2 methods, where the total number of messages sent in parallel is $6k\log(t)$, assuming that the number of processors is set to t and that the methods converge in k iterations. Note that more words are sent in the SRE-CG2 Algorithm 4 than in SRE-CG Algorithm 3, due to the A-orthonormalization procedure [11].

For $s > 1$, Algorithms 3 and 4 send $(s-1)\log(t)$ fewer messages and words per s-step iteration than Algorithm 1 and 2, assuming we have t processors with distributed memory. This communication reduction is due to the computation of one α which consists of an $n \times st$ matrix vector multiplication, rather than s computations of $n \times t$ matrix vector multiplications. Thus the total number of messages sent in parallel in Algorithms 3 and 4 is $5sk_s\log(t) + k_s\log(t)$, where k_s is the number of s-step iterations needed to converge. Similarly to the case of $s = 1$, the s-step SRE-CG2 Algorithm 4 sends more words than the s-step SRE-CG Algorithm 3.

Algorithms 3 and 4 will converge in k_s iterations, where $k_s \geq \lceil \frac{k}{s} \rceil$, and k is number of iterations needed for convergence for $s = 1$. In exact arithmetic, every s-step iteration of Algorithms 3 and 4 is equivalent to s iterations of the SRE-CG and SRE-CG2 algorithms, respectively. However, they might not be equivalent in finite

Algorithm 3. s-step SRE-CG.

```
Input: $A$, $n \times n$ symmetric positive definite matrix; $k_{max}$, maximum allowed iterations; $b$, $n \times 1$ right-hand side; $x_0$, initial guess; $\epsilon$, stopping tolerance; $s$, s-step
Output: $x_k$, approximate solution of the system $Ax = b$
1: $r_0 = b - Ax_0$, $\rho_0 = ||r_0||_2$ , $\rho = \rho_0$, $k = 1$;
2: while ( $\rho > \epsilon\rho_0$ and $k < k_{max}$ ) do
3:     Let $j = (k-1)s + 1$
4:     if ($k == 1$) then
5:         A-orthonormalize $W_j = \mathcal{T}_0$, and let $V = W_j$
6:     else
7:         A-orthonormalize $W_j = AW_{j-1}$ against $W_{j-2}$ and $W_{j-1}$
8:         A-orthonormalize $W_j$ and let $V = W_j$
9:     end if
10:    for ($i = 1 : s-1$) do
11:        A-orthonormalize $W_{j+i} = AW_{j+i-1}$ against $W_{j+i-2}$ and $W_{j+i-1}$
12:        A-orthonormalize $W_{j+i}$ and let $V = [V\ W_{j+i}]$
13:    end for
14:    $\tilde{\alpha} = V^t r_{k-1}$
15:    $x_k = x_{k-1} + V\tilde{\alpha}$
16:    $r_k = r_{k-1} - AV\tilde{\alpha}$
17:    $\rho = ||r_k||_2$,
18:    $k = k + 1$
19: end while
```

Algorithm 4. s-step SRE-CG2.

```
Input: $A$, $n \times n$ symmetric positive definite matrix; $k_{max}$, maximum allowed iterations; $b$, $n \times 1$ right-hand side; $x_0$, initial guess; $\epsilon$, stopping tolerance; $s$, s-step
Output: $x_k$, approximate solution of the system $Ax = b$
1: $r_0 = b - Ax_0$, $\rho_0 = ||r_0||_2$ , $\rho = \rho_0$, $k = 1$;
2: while ( $\rho > \epsilon\rho_0$ and $k < k_{max}$ ) do
3:     Let $j = (k-1)s + 1$
4:     if $(k == 1)$ then
5:         A-orthonormalize $W_j = \mathcal{T}_0$, and let $Q = W_j$
6:     else
7:         A-orthonormalize $W_j = AW_{j-1}$ against $Q$
8:         A-orthonormalize $W_j$, and let $Q = [Q,\ W_j]$
9:     end if
10:    Let $V = W_j$
11:    for $(i = 1 : s-1)$ do
12:        A-orthonormalize $W_{j+i} = AW_{j+i-1}$ against $Q$
13:        A-orthonormalize $W_{j+i}$, let $V = [V,\ W_{j+i}]$ and $Q = [Q,\ W_{j+i}]$
14:    end for
15:    $\tilde{\alpha} = V^t r_{k-1}$
16:    $x_k = x_{k-1} + V\tilde{\alpha}$
17:    $r_k = r_{k-1} - AV\tilde{\alpha}$
18:    $\rho = ||r_k||_2$,
19:    $k = k + 1$
20: end while
```

arithmetic due to the loss of A-orthogonality of the Q_{ks} matrix. At the first iteration of Algorithms 3 and 4,

$$\begin{aligned} x_1 &= x_0 + V_1\tilde{\alpha}_1 = x_0 + V_1(V_1^t r_0) \\ &= x_0 + [W_1 W_2 \cdots\ Ws][W_1 W_2 \cdots\ Ws]^t r_0 \\ &= x_0 + \sum_{i=1}^{s} W_i W_i^t r_0. \end{aligned} \tag{3.1}$$

For $s = 3$,

$$x_1 = x_0 + W_1 W_1^t r_0 + W_2 W_2^t r_0 + W_3 W_3^t r_0.$$

On the other hand, after three iterations of the SRE-CG and SRE-CG2 algorithms, the solution x_3 is

$$x_1 = x_0 + W_1(W_1^t r_0), \tag{3.2}$$

$$r_1 = r_0 - AW_1 W_1^t r_0, \tag{3.3}$$

$$\begin{aligned} x_2 &= x_1 + W_2(W_2^t r_1) = x_0 + W_1(W_1^t r_0) + W_2 W_2^t (r_0 - AW_1 W_1^t r_0) \\ &= x_0 + W_1 W_1^t r_0 + W_2 W_2^t r_0 - W_2(W_2^t AW_1) W_1^t r_0, \end{aligned} \tag{3.4}$$

$$r_2 = r_0 - AW_1 W_1^t r_0 - AW_2 W_2^t r_0 + AW_2(W_2^t AW_1) W_1^t r_0, \tag{3.5}$$

$$\begin{aligned}
x_3 &= x_2 + W_3W_3^tr_2 = x_0 + W_1W_1^tr_0 + W_2W_2^tr_0 - W_2(W_2^tAW_1)W_1^tr_0 \\
&\quad + W_3W_3^t(r_0 - AW_1W_1^tr_0 - AW_2W_2^tr_0 + AW_2W_2^tAW_1W_1^tr_0), \\
&= x_0 + W_1W_1^tr_0 + W_2W_2^tr_0 + W_3W_3^tr_0 - W_2(W_2^tAW_1)W_1^tr_0 \\
&\quad - W_3(W_3^tAW_1)W_1^tr_0 - W_3(W_3^tAW_2)W_2^tr_0 \\
&\quad + W_3(W_3^tAW_2)(W_2^tAW_1)W_1^tr_0.
\end{aligned} \tag{3.6}$$

For $s > 3$, more terms with $W_j^tAW_i$ will be added. Assuming that $W_j^tAW_i = 0$ for all $j < i$, then the obtained x_s in the SRE-CG and SRE-CG2 algorithms is equivalent to x_1 (3.1) in the s-step SRE-CG and s-step SRE-CG2 algorithms. Similarly, under the same assumptions, x_{is} in the SRE-CG and SRE-CG2 algorithms is equivalent to x_i in the s-step SRE-CG and s-step SRE-CG2 algorithms. In the case for some $j < i$, $W_j^tAW_i \neq 0$, then all the subsequent s-step solutions will not be equal to the corresponding SRE-CG and SRE-CG2 solutions.

Assuming that the s-step versions converge in $k_s = \lceil \frac{k}{s} \rceil$ iterations, then $5k\log(t) + \frac{k}{s}\log(t)$ messages are sent in parallel. Hence, by merging s iterations of the enlarged CG methods for some given value t, communication is reduced by a total of at most $(s-1)\log(t)k_s = \frac{s-1}{s}\log(t)k$ fewer messages and words.

Theoretically, it is possible to further reduce communication by replacing the block Arnoldi A-orthonormalization (Algorithm 4, lines 3–12) and the truncated block Arnoldi A-orthonormalization (Algorithm 3, lines 4–13) with a communication avoiding kernel that first computes the st vectors and then A-orthonormalizes them against previous vectors and against themselves, as summarized in Algorithm 5. These methods are called communication avoiding SRE-CG2 (CA SRE-CG2) and communication avoiding SRE-CG (CA SRE-CG2), respectively. For the first iteration ($k = 1$), $W_{j-1} = [T(r_0)] = \mathcal{T}_0$ in Algorithm 5.

In s-step SRE-CG, the st vectors are computed and A-orthonormalized against the previous $2t$ vectors, t vectors at a time. But in the case of CA SRE-CG, the st vectors are all computed before being A-orthonormalized. Thus, it is not sufficient to just A-orthonormalize the st computed vectors against the last $2t$ vectors. Instead, the st computed vectors should be A-orthonormalized against the last $(s+1)t$ vectors.

Assuming that $Q = Q_{(k-1)s} = [W_1, W_2, W_3, \cdots W_{(k-1)s}]$ is A-orthonormal, then for all $i + l < j$ where $j = (k-1)s$, we have that

$$\left(A^iW_l\right)^t AW_j = W_l^tA\left(A^iW_j\right) = 0.$$

Algorithm 5. CA-Arnoldi A-orthonormalization.

```
Input: $W_{j-1}$, $n \times t$ matrix; $k$, iteration
    $Q$, $n \times m$ matrix, $m = (s+1)t$ in CA SRE-CG and $m = kst$ in CA SRE-CG2
Output: $V$, the $n \times st$ matrix containing the A-orthonormalized $st$ computed vectors
1: if ($k == 1$) then $W_j = W_{j-1}$, and $V = W_j$
2: else $W_j = AW_{j-1}$, and $V = W_j$
3: end if
4: for ($i = 1 : s-1$) do
5:     Let $W_{j+i} = AW_{j+i-1}$
6:     Let $V = [V\ W_{j+i}]$
7: end for
8: if ($k > 1$) then A-orthonormalize $V$ against $Q$ end if
9: A-orthonormalize $V$
```

After computing $W_{j+i} = A^i W_{(k-1)s} = A^i W_j$ for $i = 1, \ldots, s$, the A-orthonormalization is summarized as follows:

$$\begin{aligned} W_{j+i} &= A^i W_j - QQ^t A \left(A^i W_j\right) \\ &= A^i W_j - Q[W_1^t A(A^i W_j);\ W_2^t A(A^i W_j);\ \cdots;\ W_j^t A(A^i W_j)] \\ &= A^i W_j - \sum_{l=1}^{j} W_l W_l^t A(A^i W_j) = A^i W_j - \sum_{l=j-i}^{j} W_l W_l^t A(A^i W_j). \end{aligned}$$

This implies that W_{j+1} should be A-orthonormalized against the last $2t$ vectors W_{j-1} and W_j, whereas W_{j+s} should be A-orthonormalized against the last $(s+1)t$ vectors $W_{j-s}, W_{j-s+1}, \ldots, W_j$. And in general, W_{j+i} should be A-orthonormalized against the last $(i+1)t$ vectors. To reduce communication, in CA-SRE-CG all of the st computed vectors, $W_{j+1}, W_{j+2}, \ldots, W_{j+s}$, are A-orthonormalized against the previous $(s+1)t$ vectors.

Given that we are computing the monomial basis, the st computed vectors might be linearly dependent, which leads to a numerically unstable basis. The numerical stability and convergence of such communication avoiding and s-step versions are discussed in section 4.

3.2. s-step MSDO-CG. The MSDO-CG method [11] computes t search directions at each iteration k, $P_k = \mathcal{T}_{k-1} + P_{k-1}\text{diag}(\beta_k)$, where $P_0 = \mathcal{T}_0$ and $\mathcal{T}_i$ is the matrix containing the t vectors of $T(r_i)$. Then, P_k is A-orthonormalized against all P_i's ($i < k$) and used to update $x_k = x_{k-1} + P_k\alpha_k$ and $r_k = r_{k-1} - AP_k\alpha_k$, where $\alpha_k = P_k^t r_{k-1}$. This procedure is interdependent since we cannot update P_k without r_{k-1}, and we cannot update r_{k-1} without P_{k-1}. Thus, to build an s-step version we need to split the computation of P_k and the update of x_k, which is not possible. For that purpose we introduce a modified version of MSDO-CG where we build a modified enlarged Krylov basis rather than computing search directions.

As discussed in [11], the vectors of P_k belong to the enlarged Krylov subspace

$$\mathcal{K}_{k,t} = \text{span}\{T(r_0), AT(r_0), \ldots, A^{k-1}T(r_0)\}.$$

Moreover, the vectors of P_k belong to the modified enlarged Krylov subspace

$$\overline{\mathcal{K}}_{k,t} = \text{span}\{T(r_0), T(r_1), T(r_2), \ldots, T(r_{k-1})\}. \tag{3.7}$$

In general, we define the modified enlarged Krylov subspace for a given s value as follows:

$$\begin{aligned} \overline{\mathcal{K}}_{k,t,s} = \text{span}\{&T(r_0), AT(r_0), \ldots, A^{s-1}T(r_0), \\ &T(r_1), AT(r_1), \ldots, A^{s-1}T(r_1), \\ &T(r_2), AT(r_2), \ldots, A^{s-1}T(r_2), \\ &\vdots \\ &T(r_{k-1}), AT(r_{k-1}), \ldots, A^{s-1}T(r_{k-1})\}. \end{aligned}$$

Note that for $s = 1$, the modified enlarged Krylov subspace becomes $\overline{\mathcal{K}}_{k,t}$ defined in (3.7). Moreover, for $t = s = 1$, the modified enlarged Krylov subspace becomes

$$\overline{\mathcal{K}}_{k,1,1} = \text{span}\{r_0, r_1, r_2, \ldots, r_{k-1}\},$$

where the Krylov subspace $\mathcal{K}_k$=span$\{r_0, Ar_0, \ldots, A^{k-1}r_0\}$=span$\{r_0, r_1, r_2, \ldots, r_{k-1}\}$.

Similarly to the enlarged Krylov subspace $\mathcal{K}_{ks,t}$, the modified enlarged Krylov subspace $\overline{\mathcal{K}}_{k,t,s}$ is of dimension at most kst. Moreover, the modified enlarged Krylov subspace $\overline{\mathcal{K}}_{k,t,1}$ is a super set of the Krylov subspace $\mathcal{K}_k$, similarly to the enlarged Krylov subspace $\mathcal{K}_{k,t}$ (Theorem 3.2 in [11]).

THEOREM 3.1. *The Krylov subspace $\mathcal{K}_k$ is a subset of the modified enlarged Krylov subspace $\overline{\mathcal{K}}_{k,t,1}$ ($\mathcal{K}_k \subset \overline{\mathcal{K}}_{k,t,1}$).*

Proof. Let $y \in \mathcal{K}_k$, where $\mathcal{K}_k = \text{span}\{r_0, Ar_0, \dots, A^{k-1}r_0\} = \text{span}\{r_0, r_1, r_2, \dots, r_{k-1}\}$. Then,

$$y = \sum_{j=0}^{k-1} a_j r_j = \sum_{j=0}^{k-1} a_j \mathcal{T}_j * \mathbb{1}_t = \sum_{j=0}^{k-1}\sum_{i=1}^{t} a_j T_i(r_j) \in \overline{\mathcal{K}}_{k,t,1}$$

since $r_j = \mathcal{T}_j * \mathbb{1}_t = [T_1(r_j)\, T_2(r_j)\, \cdots\, T_t(r_j)] * \mathbb{1}_t$, where $\mathbb{1}_t$ is a $t \times 1$ vector of ones, and $\mathcal{T}_j = [T(r_j)] = [T_1(r_j)\, T_2(r_j)\, \cdots\, T_t(r_j)]$ is the matrix containing the t vectors of $T(r_j)$. ∎

Then one possible s-step reformulation of MSDO-CG would be to compute basis vectors of $\overline{\mathcal{K}}_{k,t,s}$ and use them to update the solution and the residual, similarly to the s-step SRE-CG versions. At iteration k of the s-step MSDO-CG method (Algorithm 6), the st vectors

$$T(r_{k-1}), AT(r_{k-1}), \dots, A^{s-1}T(r_{k-1})$$

are computed and A-orthonormalized similarly to the s-step SRE-CG2 method and stored in the $n \times st$ matrix V_k. Then, these st A-orthonormalized vectors are used to define $\tilde{\alpha}_k = V_k^t r_{k-1}$ and update $x_k = x_{k-1} + V_k\tilde{\alpha}_k$ and $r_k = r_{k-1} - AV_k\tilde{\alpha}_k$.

Algorithm 6. s-step MSDO-CG.

```
Input: $A$, $n \times n$ symmetric positive definite matrix; $k_{max}$, maximum allowed iterations; $b$, $n \times 1$ right-hand side; $x_0$, initial guess; $\epsilon$, stopping tolerance; $s$, s-step
Output: $x_k$, approximate solution of the system $Ax = b$
1: $r_0 = b - Ax_0$, $\rho_0 = ||r_0||_2$ , $\rho = \rho_0$, $k = 1$;
2: while ( $\rho > \epsilon\rho_0$ and $k < k_{max}$ ) do
3:     if ($k == 1$) then
4:         A-orthonormalize $W_1 = \mathcal{T}_0$, let $V_k = W_1$ and $Q = W_1$
5:     else
6:         A-orthonormalize $W_1 = \mathcal{T}_{k-1}$ against $Q$
7:         A-orthonormalize $W_1$, let $V_k = W_1$ and $Q = [Q\ W_1]$
8:     end if
9:     for ($i = 1 : s - 1$) do
10:        A-orthonormalize $W_{i+1} = AW_i$ against $Q$
11:        A-orthonormalize $W_{i+1}$, let $V_k = [V_k\ W_{i+1}]$ and $Q = [Q\ W_{i+1}]$
12:    end for
13:    $\tilde{\alpha} = V_k^t r_{k-1}$
14:    $x_k = x_{k-1} + V_k\tilde{\alpha}_k$
15:    $r_k = r_{k-1} - AV_k\tilde{\alpha}_k$
16:    $\rho = ||r_k||_2$, $k = k + 1$
17: end while
```

Note that for $s = 1$, the s-step MSDO-CG method is reduced to a modified version of MSDO-CG. Although the s-step MSDO-CG method for $s = 1$ is different algorithmically from the MSDO-CG method, but they converge in the same number of iterations as shown in section 4 due to their theoretical equivalence. Moreover, each iteration of the s-step MSDO-CG method with $s > 1$ is not equivalent to s iterations of the modified version of MSDO-CG, since the constructed bases of $\overline{\mathcal{K}}_{k,t,s}$ and $\overline{\mathcal{K}}_{ks,t,1}$ are different. For example, in the second iteration of s-step MSDO-CG ($k = 2$), $T(r_1), AT(r_1), \ldots, A^{s-1}T(r_1)$ are computed, whereas in the second s iterations of the modified version of MSDO-CG ($ks = 2s$), the vectors $T(r_s), T(r_{s+1}), \ldots, T(r_{2s-1})$ are computed.

The communication avoiding MSDO-CG differs from the s-step version (Algorithm 6) in the basis construction where at the kth iteration the st vectors $T(r_{k-1})$, $AT(r_{k-1}), \ldots,$ $A^{s-1}T(r_{k-1})$ are first computed and then A-orthonormalized against previous vectors and against themselves. Thus the communication avoiding MSDO-CG algorithm is Algorithm 6 with the replacement of lines 3–12 by the CA-Arnoldi A-orthonormalization Algorithm 5. However, Algorithm 5 is slightly modified, where in line 2 $W_j = W_{j-1}$ rather than $W_j = AW_{j-1}$, with $W_{j-1} = \mathcal{T}_{k-1} = [T(r_{k-1})]$ for $k \geq 1$.

The advantage of building the modified enlarged Krylov subspace basis is that at iteration k, each of the t processors can compute the s basis vectors

$$T_i(r_{k-1}), AT_i(r_{k-1}), A^2T_i(r_{k-1}), \ldots, A^{s-1}T_i(r_{k-1})$$

independently, where $T_i(r_{k-1})$ is the projection of the vector r_{k-1} on the ith domain of matrix A, i.e., a vector of all zeros except at n/t entries that correspond to the ith domain. Thus, there is no need for communication avoiding kernels, since processor i needs a part of matrix A and a part of the vector r_{k-1} to compute the s vectors. As a consequence, assuming that enough memory is available, then any preconditioner can be applied to the CA MSDO-CG since the matrix powers kernel is not used to compute the basis vectors, as discussed in section 6.2.

4. Numerical stability and convergence.

In sections 4.1.2 and 4.1.3 of [22], the convergence analysis of the enlarged Krylov subspace is discussed. It is shown that if the Petrov–Galerkin condition is respected ($r_k \perp \mathcal{K}_{k,t}$) where $\mathcal{K}_k \subset \mathcal{K}_{k,t}$, then the A-norm of the enlarged CG error is bounded from above by the A-norm of the CG error. This means that the enlarged CG version will converge at least as fast as CG, provided that the computed residual is orthogonal to $\mathcal{K}_{k,t}$. In the case of the s-step and CA methods, it can be shown that the A-norm of the kth error is bounded from above by the A-norm of CG's ksth error. The same proof applies with very few modifications, thus we will not include it in this paper.

For s-step and the CA, SRE-CG, and SRE-CG2 methods, the only difference is that the Petrov–Galerkin condition is $r_k \perp \mathcal{K}_{ks,t}$, where $\mathcal{K}_{ks} \subset \mathcal{K}_{ks,t}$. Thus, if the computed residual is no longer orthogonal to $\mathcal{K}_{ks,t}$, then $s * k_s$ might be larger than k, where k_s is the number of iterations for s-step and CA versions' convergence and k is that of CG. This loss of orthogonality results from the numerical loss of A-orthogonality of the computed candidate basis vectors. Similarly, for s-step and CA MSDO-CG, the Petrov–Galerkin condition is $r_k \perp \overline{\mathcal{K}}_{k,t,s}$ and it can be proven by induction that $\mathcal{K}_{ks} \subset \overline{\mathcal{K}}_{k,t,s}$. Hence, the numerical stability of the A-orthonormalization procedure is crucial for the convergence of the s-step and CA versions.

We compare the convergence behavior of the different introduced s-step enlarged CG versions and their communication avoiding versions for solving the system $Ax = b$

Table 4.1
The test matrices.

Matrix	Size	Nonzeros	2D/3D	Problem
Poisson2D	10000	49600	2D	Poisson equations
Nh2D	10000	49600	2D	Boundary value
Sky2D	10000	49600	2D	Boundary value
Sky3D	8000	53600	3D	Skyscraper
Ani3D	8000	53600	3D	Anisotropic layers

using different numbers of partitions ($t = 2, 4, 8, 16, 32$, and 64 partitions) and different numbers of s-values (1, 2, 3, 4, 5, 8, 10). Similarly to the enlarged CG methods [11], matrix A is first reordered using Metis's kway partitioning [18], which defines the t subdomains. Then x is chosen randomly using the MATLAB 2015b rand function (rand('twister', 5489); x = 4*rand(numeq,1);) and the right-hand side is defined as $b = Ax$. The initial iterate is set to $x_0 = 0$, and the stopping criteria tolerance is set to $tol = 10^{-8}$ for all the matrices, except Poisson2D ($tol = 10^{-6}$).

The characteristics of the test matrices are summarized in Table 4.1. The Poisson2D matrix is a block tridiagonal matrix obtained from Poisson's equation using the MATLAB "gallery('poisson',100)." The remaining matrices, referred to as Nh2D, Sky2D, Sky3D, and Ani3D, arise from different boundary value problems of convection diffusion equations and are generated using FreeFem++ [15]. For a detailed description of the test matrices, refer to [11].

The first phase in all the discussed algorithms is building the A-orthonormal basis by A-orthonormalizing a set of vectors against previous vectors using classical Gram–Schmidt A-orthonormalization (CGS), CGS2, or MGS and then against themselves using A-CholQR [24] or Pre-CholQR [20]. As discussed in [22], the combinations CGS2+A-CholQR and CGS2+Pre-CholQR are both numerically stable and require less communication. In this paper, we test the introduced methods using CGS2 (Algorithm 18 in [22]), A-CholQR (Algorithm 21 in [22]), and Pre-CholQR (Algorithm 23 in [22]). Based on the performed testing, Pre-CholQR is numerically more stable than A-CholQR. However, for most of the tested cases, the versions with CGS2+A-CholQR or CGS2+Pre-CholQR A-orthonormalization converge in the same number of iterations.

In Table 4.2 we compare the convergence behavior of the different SRE-CG versions with respect to number of partitions t and the s values. The restructured SRE-CG is a reordered version of SRE-CG, where the same operations of s iterations are performed, but in a different order. In addition, the check for convergence is done once every s iterations. Thus the restructured SRE-CG Algorithm 1 converges in $s * k_s$ iterations. In Table 4.2, k_s is shown rather than $s * k_s$, for comparison purposes with the s-step versions. Moreover, for $s = 1$, the restructured SRE-CG is reduced to SRE-CG, and it converges in k iterations.

If in Algorithm 1 the second inner for loop is replaced by a while loop with a check for convergence ($||r_{i-1}||_2 > \epsilon ||r_0||_2$), then the algorithm converges in exactly $s * \lceil \frac{k}{s} \rceil$ iterations, where SRE-CG converges in k iterations and $k \leq s * \lceil \frac{k}{s} \rceil \leq k + s - 1$.

Thus, it is expected that the restructured SRE-CG (Algorithm 1) converges in k_1 iterations, where $k_1 \geq s * \lceil \frac{k}{s} \rceil$. In case SRE-CG converges in k iterations, and k is divisible by s, then Algorithm 1 converges in exactly $k_1 = s * \lceil \frac{k}{s} \rceil = k$ iterations.

TABLE 4.2
Comparison of the convergence of different SRE-CG versions (restructured SRE-CG, s-step SRE-CG, and CA SRE-CG with Algorithm 7), with respect to number of partitions t and s values.

			Restructured SRE-CG							s-step SRE-CG						CA SRE-CG		
	CG	t \ s	1	2	3	4	5	8	10	2	3	4	5	8	10	2	3	4
Poisson2D	195	2	193	97	65	49	39	25	20	97	65	49	39	25	20	97	65	49
		4	153	77	51	39	31	20	16	77	51	39	31	20	16	77	51	39
		8	123	62	41	31	25	16	13	62	41	31	25	16	13	62	41	31
		16	95	48	32	24	19	12	10	48	32	24	19	12	10	48	32	24
		32	70	35	24	18	14	9	7	35	24	18	14	9	7	35	24	18
		64	52	26	18	13	11	7	6	26	18	13	11	7	6	26	18	13
Nh2D	259	2	245	123	82	62	49	31	25	123	82	62	49	31	25	123	82	62
		4	188	94	63	47	38	24	19	94	63	47	38	24	19	94	63	47
		8	149	75	50	38	30	19	15	75	50	38	30	19	15	75	50	38
		16	112	56	38	28	23	14	12	56	38	28	23	14	12	56	38	28
		32	82	41	28	21	17	11	9	41	28	21	17	11	9	41	28	21
		64	60	30	20	15	12	8	6	30	20	15	12	8	6	30	20	15
Sky2D	5951	2	5526	2763	1842	1395	1116	708	558	2763	1842	1395	1116	708	558	2793	1854	x
		4	4526	2263	1521	1141	913	571	457	2263	1521	1141	913	571	457	2328	1575	x
		8	2843	1423	949	712	575	356	288	1423	949	712	575	356	334	1405	973	x
		16	1770	885	590	450	354	225	177	885	590	450	354	225	183	910	605	x
		32	999	500	333	250	200	125	100	500	333	250	200	125	x	492	340	x
		64	507	255	169	128	102	64	51	255	169	128	102	x	x	257	179	x
Sky3D	902	2	829	435	290	218	174	109	87	435	290	218	174	109	87	426	285	x
		4	745	382	255	191	149	98	78	382	255	191	149	142	473	373	251	x
		8	590	295	197	148	118	74	59	295	197	148	118	110	199	294	198	x
		16	436	218	146	109	89	56	45	218	146	109	89	58	74	223	150	x
		32	279	142	93	71	57	36	29	142	93	71	57	x	x	140	97	x
		64	157	79	53	40	32	20	16	79	53	40	32	314	250	78	54	x
Ani3D	4146	2	4005	2030	1335	1015	801	510	406	2030	1335	1015	801	510	406	1985	1346	x
		4	3570	1785	1190	909	714	464	357	1785	1190	909	714	464	357	1776	1201	x
		8	3089	1612	1075	806	645	403	325	1612	1075	x	x	x	x	1548	1070	x
		16	2357	1219	815	610	488	305	244	1219	815	x	x	x	x	1169	800	x
		32	1640	820	552	410	328	205	164	820	552	1686	2729	1804	1499	816	549	x
		64	928	464	315	232	189	116	95	464	315	792	777	492	501	453	316	x

On the other hand, if k is not divisible by s, then Algorithm 1 either converges in $k_1 = s * \lceil \frac{k}{s} \rceil \leq k + s - 1$ or converges in $k_1 \geq k + s$ iterations. The first case occurs when the norm of the residual in the $s * \lceil \frac{k}{s} \rceil$ iteration remains less than $tol * ||r_0||_2$. Otherwise, if the L2 norm of the residual fluctuates, then Algorithm 1 requires slightly more iterations to converge.

For the matrices POISSON2D and NH2D, the restructured SRE-CG (Algorithm 1) converges in exactly $k_1 = s * \lceil \frac{k}{s} \rceil$ iterations. For the other matrices, the three discussed cases are observed, i.e., the restructured SRE-CG converges in k_1 iterations where $k_1 \geq s * \lceil \frac{k}{s} \rceil$. For example, for matrix SKY2D with $t = 32$ and $2 \leq s \leq 10$, Algorithm 1 converges in $k_1 = s * \lceil \frac{k}{s} \rceil$ iterations. But for $s = 4$, Algorithm 1 converges in $k_1 = s * \lceil \frac{k}{s} \rceil + j$ iterations where $j = 0$ for $t = 32$, $j = 4$ for $t = 8$ or 64, $j = 28$ for $t = 16$, $j = 36$ for $t = 4$, and $j = 52$ for $t = 2$.

The s-step SRE-CG method (Algorithm 3) differs from the restructured version in the update of the approximate solutions x_k. As discussed in section 3.1, if there is no loss of A-orthogonality of the basis, then the s-step SRE-CG method should converge in k_s iterations, where the restructured SRE-CG method converges in $k_1 = s * k_s$ iterations. This is the case for the matrices Poisson2D and Nh2D for all the tested t and s values ($2 \leq t \leq 64$ and $2 \leq s \leq 10$).

On the other hand, for the remaining three matrices for some values of s and t, the s-step SRE-CG method converges in $k_s + j$ iterations due to loss of A-orthogonality of the basis. For example, for Sky2D matrix with $t = 2, 4$ and $2 \leq s \leq 10$ the s-step SRE-CG method converges in exactly k_s iterations, and similarly for $t = 8, 16, 32$ with $2 \leq s \leq 8$, and for $t = 64$ with $2 \leq s \leq 5$. But, for $t = 8, 16$ with $8 \leq s \leq 10$, s-step SRE-CG converges in $k_s + j$ iterations. However, for $t = 32$ with $s = 10$ and $t = 64$ with $8 \leq s \leq 10$, the s-step SRE-CG requires more iterations to converge than the SRE-CG does for the corresponding t, which is why an $\times$ is placed in Table 4.2. A similar convergence behavior is observed for matrix Sky3D.

As expected, the communication avoiding SRE-CG method (Algorithm 3 with CA-Arnoldi A-orthonormalization Algorithm 5) is numerically unstable due to the enlarged monomial basis construction. Unlike the s-step version, at the ith iteration the st vectors $AW, A^2W, \ldots, A^sW$ are first computed and stored in V, then A-orthonormalized with respect to the $(s+1)t$ previous vectors and against themselves, where W is an $n \times t$ matrix containing the A-orthonormalized $A^{s(i-1)-1}T(r_0)$ vectors. To stabilize the CA-Arnoldi A-orthonormalization (Algorithm 5), the first t vectors AW are A-orthonormalized with respect to the previous vectors and against themselves, and then the $(s-1)t$ vectors $A(AW), A^2(AW), \ldots, A^{s-1}(AW)$ are computed, as shown in Algorithm 7.

In Table 4.2, we test the CA SRE-CG method, where the st vectors are A-orthonormalized against the previous $(s+1)t$ vectors using Algorithm 7 for $k > 1$. The CA SRE-CG with Algorithm 7 converges at a similar rate as the s-step version for ill-conditioned matrices, such as Sky2D, Sky3D, and Ani3D, with $s = 2$ and 3 only. However, for the matrices Nh2D and Poisson2D CA SRE-CG converges in the

Algorithm 7. Modified CA-Arnoldi A-orthonormalization.

```
Input: $W_{j-1}$, $n \times t$ matrix; $k$, iteration
       $Q$, $n \times m$ matrix, $m = (s+1)t$ in CA SRE-CG and $m = kst$ in CA SRE-CG2
Output: $V$, the $n \times st$ matrix containing the A-orthonormalized $st$ computed vectors
1: if $(k == 1)$ then Let $W_j = W_{j-1}$
2: else Let $W_j = AW_{j-1}$, A-orthonormalize $W_j$ against $Q$
3: end if
4: A-orthonormalize $W_j$, let $V = W_j$
5: for $(i = 1 : s-1)$ do
6:     Let $W_{j+i} = AW_{j+i-1}$
7:     Let $V = [V \ W_{j+i}]$
8: end for
9: if $(k > 1)$ then A-orthonormalize $V$ against $Q$ end if
10: A-orthonormalize $V$
```

same number of iterations as s-step SRE-CG, even for $s \geq 4$ (not shown in the table). This implies that CA SRE-CG should converge in approximately $\lceil \frac{k}{s} \rceil$ iterations for $s \geq 4$, once the ill-conditioned systems are preconditioned.

In Table 4.3, we compare the convergence behavior of the different SRE-CG2 versions with respect to number of partitions t and the s values. In general, a similar behavior to the corresponding SRE-CG versions in Table 4.2 is observed. Yet, the SRE-CG2 versions converge faster than their corresponding SRE-CG versions and are numerically more stable.

For $s = 1$, the restructured SRE-CG2 method is equivalent to the SRE-CG2 method and converges in k iterations. For $s > 1$, the restructured SRE-CG2 method

Table 4.3
Comparison of convergence of different SRE-CG2 versions (restructured SRE-CG2, s-step SRE-CG2, and CA SRE-CG2 with Algorithm 7) with respect to number of partitions t and s values.

			Restructured SRE-CG2							s-step SRE-CG2						CA SRE-CG2		
	CG	t \ s	1	2	3	4	5	8	10	2	3	4	5	8	10	2	3	4
Poisson2D	195	2	193	97	65	49	39	25	20	97	65	49	39	25	20	97	65	49
		4	153	77	51	39	31	20	16	77	51	39	31	20	16	77	51	39
		8	123	62	41	31	25	16	13	62	41	31	25	16	13	62	41	31
		16	95	48	32	24	19	12	10	48	32	24	19	12	10	48	32	24
		32	70	36	24	18	14	9	8	35	24	18	14	9	8	35	24	18
		64	52	26	18	13	11	7	6	26	18	13	11	7	6	26	18	13
Nh2D	259	2	243	122	81	61	49	31	25	122	81	61	49	31	25	123	81	61
		4	194	97	65	49	39	25	20	97	65	49	39	25	20	94	65	47
		8	150	75	50	38	30	19	15	75	50	38	30	19	15	75	50	38
		16	113	57	38	29	23	15	12	57	38	29	23	15	12	56	38	29
		32	84	42	28	21	17	11	9	42	28	21	17	11	9	41	28	21
		64	60	30	20	15	12	8	6	30	20	15	12	8	6	30	20	15
Sky2D	5951	2	1415	708	472	354	283	177	142	708	472	354	283	177	142	708	472	365
		4	756	378	252	189	152	95	76	378	252	189	152	95	76	378	252	576
		8	399	200	133	100	80	50	40	200	133	100	80	50	40	199	133	295
		16	219	110	73	55	44	28	22	110	73	55	44	28	22	109	73	147
		32	125	63	42	32	25	16	13	63	42	32	25	16	13	63	42	77
		64	74	37	25	19	15	10	8	37	25	19	15	10	8	37	25	39
Sky3D	902	2	570	285	190	143	114	72	57	285	190	144	114	72	57	285	190	155
		4	375	190	125	95	75	48	38	190	125	95	75	48	38	190	127	101
		8	213	107	71	54	43	27	22	107	71	54	43	27	22	107	71	224
		16	117	59	39	30	24	15	12	59	39	30	24	15	12	59	39	124
		32	69	35	23	18	14	9	7	35	23	18	14	9	7	35	23	69
		64	43	22	15	11	9	6	5	22	15	11	9	6	5	22	15	x
Ani3D	4146	2	875	438	292	219	175	110	88	438	292	219	175	110	88	438	292	219
		4	673	340	229	170	131	81	66	340	229	170	131	81	66	340	229	170
		8	449	225	150	113	90	57	45	225	150	113	90	57	45	225	150	113
		16	253	127	85	64	51	32	26	127	85	64	51	32	26	127	85	78
		32	148	74	49	37	30	19	15	74	50	37	30	19	15	74	50	58
		64	92	46	31	23	19	12	10	46	31	23	19	12	10	46	31	31

converges in k_1 iterations, where $k_1 = s * k_s \geq s * \lceil \frac{k}{s} \rceil$, similarly to the restructured SRE-CG method. The s-step SRE-CG2 converges in k_s iterations for $s \geq 2$ for all the tested matrices. As for the communication avoiding version (Algorithm 4) with the modified CA-Arnoldi A-orthonormalization (Algorithm 7), it does not converge as fast as the s-step version for ill-conditioned matrices, such as Sky2D, Sky3D, and Ani3D, with large s-values ($s \geq 4$). Yet, CA SRE-CG2 converges in the same number of iterations as s-step SRE-CG2, for the matrices Nh2D and Poisson2D, even with $s \geq 4$ (not shown in the table).

In Table 4.4, we compare the convergence behavior of MSDO-CG, s-step MSDO-CG, CA MSDO-CG with Algorithm 5, and CA MSDO-CG with Algorithm 7 (where

Table 4.4
Comparison of the convergence of different MSDO-CG versions (s-step MSDO-CG, CA MSDO-CG with Algorithm 5, and CA MSDO-CG with Algorithm 7) with respect to number of partitions t and s values.

			MSDOCG	s-step MSDO-CG									CA MSDO-CG with Alg5				CA MSDO-CG with Algorithm 7					
	CG	t \ s	1	1	2	3	4	5	6	7	8	10	2	3	4	5	2	3	4	5	6	7
Poisson2D	195	2	198	198	99	66	49	40	33	28	23	18	99	66	50	40	99	66	49	34	33	28
		4	166	166	83	56	42	34	28	24	21	17	83	56	42	34	83	55	42	33	28	25
		8	137	137	68	46	34	27	23	19	17	13	69	46	36	28	68	46	35	28	24	21
		16	121	121	59	39	29	23	18	16	14	11	61	41	31	25	59	38	28	23	19	16
		32	95	95	45	29	22	17	14	12	10	8	48	32	25	20	45	30	22	18	15	13
		64	69	69	33	21	16	12	10	9	8	6	37	25	20	16	33	22	16	14	12	10
Nh2D	259	2	255	255	127	84	63	51	42	36	32	26	127	85	64	51	127	84	63	51	42	37
		4	210	210	104	69	52	42	34	30	26	20	105	71	53	42	104	68	52	42	35	31
		8	170	170	84	56	42	33	27	23	20	16	85	57	43	34	84	56	42	33	29	25
		16	138	138	68	44	33	26	21	18	16	12	70	47	36	28	68	44	33	27	23	20
		32	106	106	51	33	25	19	16	13	12	10	54	36	28	23	51	33	25	21	18	16
		64	76	76	37	24	17	14	11	10	9	7	41	28	22	17	37	24	19	16	14	12
Sky2D	5951	2	1539	1539	719	480	360	288	240	206	180	144	778	527	401	327	720	493	374	307	259	224
		4	916	916	397	259	194	154	129	110	96	77	466	312	x	x	395	271	207	170	143	x
		8	517	517	214	141	105	84	70	60	52	42	260	167	129	x	214	149	114	95	80	x
		16	277	277	122	81	60	47	40	34	30	24	141	95	x	x	122	85	66	54	x	x
		32	192	192	74	48	36	28	23	20	17	14	92	61	x	x	73	51	40	33	x	x
		64	123	123	47	29	22	17	14	12	11	8	60	x	x	x	47	32	25	23	x	x
Sky3D	902	2	637	633	334	211	169	139	111	89	81	66	339	235	180	153	333	242	193	160	134	119
		4	374	373	205	137	103	81	68	58	51	41	204	138	107	86	206	140	109	89	75	66
		8	224	224	112	74	56	44	37	32	28	22	117	82	63	50	112	78	61	49	42	37
		16	137	137	63	42	31	25	21	18	16	13	73	50	38	32	63	45	35	29	25	22
		32	89	89	38	25	19	15	13	11	9	8	45	30	23	x	38	27	21	18	19	x
		64	50	50	24	16	12	10	8	7	6	5	27	19	16	x	24	17	14	12	x	x
Ani3D	4146	2	896	896	456	301	227	181	152	130	114	91	458	312	237	195	452	309	242	201	168	148
		4	796	796	362	238	177	140	115	99	88	69	413	281	216	176	362	253	194	159	135	117
		8	473	473	231	154	115	92	77	66	58	46	238	163	126	x	231	158	120	98	83	x
		16	292	292	130	86	64	51	43	37	32	26	140	95	x	x	130	91	70	57	55	x
		32	213	213	77	51	38	30	25	22	19	15	97	61	x	x	76	55	42	35	x	x
		64	115	115	48	31	24	19	16	14	12	10	56	x	x	x	48	34	27	24	x	x

$W_{j-1} = [T(r_{k-1})]$ and $W_j = W_{j-1}$ for $k \geq 1$) versions with respect to number of partitions t and the s values. We do not test a restructured MSDO-CG since the s-step version is not exactly equivalent to the merging of s iterations of MSDO-CG. The s-step MSDO-CG with $s = 1$ is equivalent to a modified version of MSDO-CG, which differs algorithmically from MSDO-CG but is equivalent theoretically. Moreover, MSDO-CG and s-step MSDO-CG with $s = 1$ converge in the same number of iterations for all t values and matrices. For $s \geq 2$, s-step MSDO-CG converges in m iterations where in most cases $m \leq \lceil \frac{k}{s} \rceil$ and MSDO-CG converges in k iterations. Moreover, for all the matrices, the s-step MSDO-CG converges for $s = 10$ and all values of t.

Unlike the CA SRE-CG and CA SRE-CG2 with the CA-Arnoldi Algorithm 5, the CA MSDO-CG with Algorithm 5 converges for $s = 2$ and 3, as shown in Table 4.4. The difference is that in the SRE-CG and SRE-CG2 we are computing a modified block version of the powers method, where t vectors ($T(r_0)$) are multiplied by powers of A and are A-orthonormalized. Thus there is a higher chance that these vectors converge to the largest eigenvector in a very fast rate, leading to a numerically linearly dependent basis, whereas in CA MSDO-CG at every iteration we are computing a block version of the powers method but starting with a new set of t vectors, i.e., $T(r_{k-1})$ at the kth iteration. For the matrices Nh2D and Poisson2D, CA MSDO-CG scales even for $s > 5$. But for the other matrices, as s grows, the CA MSDO-CG requires much more than $\lceil \frac{k}{s} \rceil$ and $\lceil \frac{k}{s-1} \rceil$ iterations to converge, due to the stagnation of the relative error.

For the matrices Nh2D and Poisson2D, CA MSDO-CG with Algorithm 7 converges in exactly the same number of iterations as CA MSDO-CG with Algorithm 5 and s-step MSDO-CG up to $s = 10$. On the other hand, the CA MSDO-CG with Algorithm 7 converges faster than CA MSDO-CG with the Algorithm 5 version for the corresponding s and t values, for the matrices Sky3D (except for $t = 2, 4$), Sky2D, and Ani3D (except for $t = 2, 4$). More importantly, CA MSDO-CG with Algorithm 7 is numerically more stable than CA MSDO-CG with Algorithm 5 and CA SRE-CG2 with Algorithm 7, as it scales up to at least $s = 5$, or 6, whereas CA SRE-CG2 with Algorithm 7 and CA MSDO-CG with Algorithm 5 scale up to $s = 3$, or 4, as shown in Tables 4.3 and 4.4.

As a summary, for the well-conditioned matrices, such as Nh2D and Poisson2D, the s-step and communication avoiding with Algorithm 7 versions of SRE-CG and SRE-CG2 converge in the same number of iterations and scale up to at least $s = 10$. But the communication avoiding with Algorithm 5 versions of SRE-CG and SRE CG2 do not converge due to the instability in the basis construction, specifically the A-orthonormalization process. On the other hand, the s-step, communication avoiding with Algorithm 5, and communication avoiding with Algorithm 7 versions of MSDO-CG for the matrices Nh2D and Poisson2D converge in the same number of iterations and scale up to at least $s = 10$. Moreover, the corresponding versions of SRE-CG, SRE-CG2, and MSDO-CG converge in approximately the same number of iterations as shown in Figure 4.1.

For the other matrices, the s-step versions of SRE-CG, SRE-CG2, and MSDO-CG converge and scale up to at least $s = 10$, as expected. The communication avoiding with Algorithm 5 versions of SRE-CG and SRE-CG2 do not converge. But the communication avoiding MSDO-CG with Algorithm 5 converges. Moreover, the communication avoiding MSDO-CG with Algorithm 7 scales better than the communication avoiding SRE-CG2 with Algorithm 7, even though it might require more iterations as shown in Figure 4.2 for the matrix Sky3D.

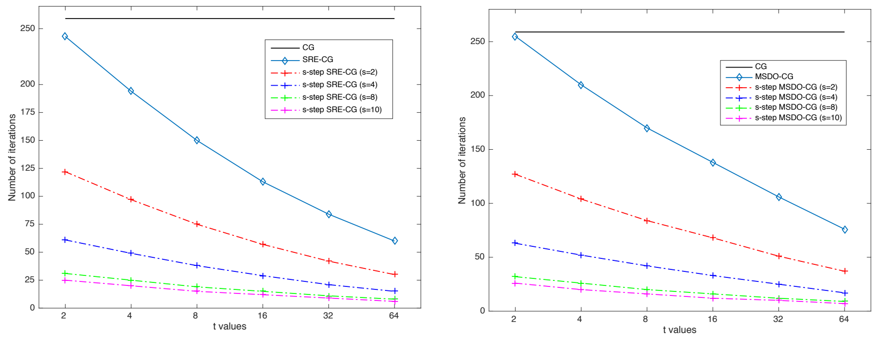

FIG. 4.1. *Convergence of s-step SRE-CG* (a) *and of s-step MSDO-CG* (b) *for* NH2D *with tol* $= 10^{-8}$. (a) *Same convergence for CA SRE-CG, s-step SRE-CG*2*, and CA SRE-CG*2. (b) *Same convergence for CA MSDO-CG.*

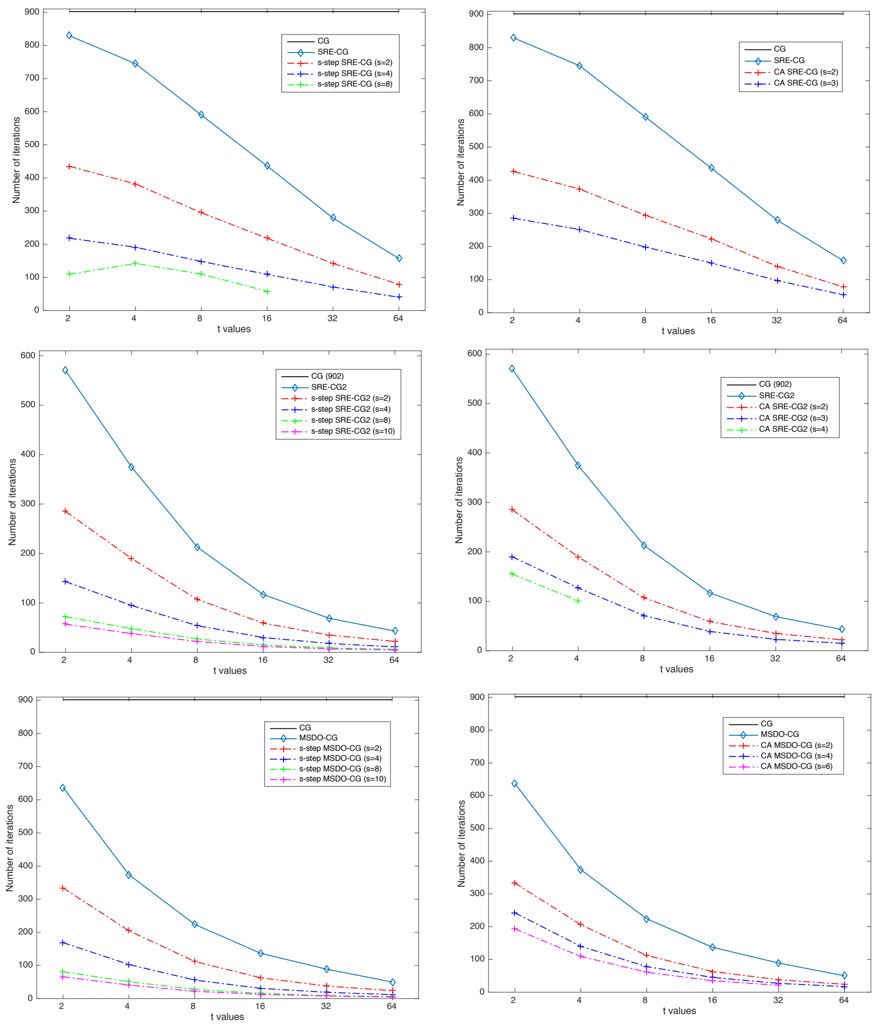

FIG. 4.2. *Convergence of s-step SRE-CG*2*, CA SRE-CG*2*, s-step SRE-CG*2*, CA SRE-CG*2*, s-step MSDO-CG, and CA MSDO-CG for* SKY3D *with tol* $= 10^{-8}$ *and different s values. CG convergence plot is not shown in the SRE-CG*2 *figures.*

5. The preconditioned versions.

Krylov subspace methods are rarely used without preconditioning. Moreover, CG is a method for solving symmetric positive definite matrices. For this purpose, split preconditioned versions of the abovementioned s-step methods for solving the system $L^{-1}AL^{-t}(L^tx) = L^{-1}b$ are introduced, where the preconditioner is $M = LL^t$. Then, the observed numerical stability of the preconditioned methods is briefly discussed.

5.1. Preconditioned algorithms.

One possible way for preconditioning the s-step versions is by simply replacing A by $L^{-1}AL^{-t}$ and b by $L^{-1}b$ in the algorithms, where first $L^{-1}AL^{-t}y = L^{-1}b$ is solved and then the solution x is obtained by solving $y = L^tx$. In [22], MSDO-CG is preconditioned in this manner (Algorithm 40), where the vectors are $L^{-1}AL^{-t}$-orthonormalized (Algorithms 19 and 22) rather than A-orthonormalized. In this paper we will precondition the s-step and communication avoiding methods by avoiding the use of $L^{-1}AL^{-t}$-orthonormalization.

Assume the following system $\widehat{A}\widehat{x} = \widehat{b}$, where $\widehat{A} = L^{-1}AL^{-t}$, $\widehat{x} = L^tx$, and $\widehat{b} = L^{-1}b$. The below relations summarize the SRE-CG, SRE-CG2, and modified MSDO-CG methods for this system:

$$\begin{aligned}\widehat{\alpha}_k &= \widehat{V}_k^t\widehat{r}_{k-1},\\ \widehat{x}_k &= \widehat{x}_{k-1} + \widehat{V}_k\widehat{\alpha}_k,\\ \widehat{r}_k &= \widehat{r}_{k-1} - \widehat{A}\widehat{V}_k\widehat{\alpha}_k.\end{aligned}$$

The difference is in how the $\widehat{V}_k$ vectors are constructed. In the modified MSDO-CG, $\widehat{V}_k$ is set to $[T(\widehat{r}_{k-1})]$ and then $\widehat{A}$-orthonormalized against all previous vectors. In the SRE-CG and SRE-CG2 methods,

$$\widehat{V}_k = \begin{cases}[T(\widehat{r}_0)] & \text{if } k = 1,\\ \widehat{A}\widehat{V}_{k-1} & \text{if } k \geq 2,\end{cases}$$

and then $\widehat{V}_k$ is $\widehat{A}$-orthonormalized against the previous $2t$ vectors (SRE-CG) or against all previous vectors (SRE-CG2). In the three methods $\widehat{V}_i^t\widehat{A}\widehat{V}_i = 0$. Moreover, $\widehat{V}_k^t\widehat{A}\widehat{V}_i = 0$ where i $= k-2$, $k-1$ for SRE-CG, and $i < k$ for SRE-CG2 and modified MSDO-CG.

Note that $\widehat{r}_k = \widehat{b} - \widehat{A}\widehat{x}_k = L^{-1}b - L^{-1}AL^{-t}L^tx_k = L^{-1}(b - Ax_k) = L^{-1}r_k$. Thus, we derive the corresponding equations for x_k and r_k:

$$\begin{aligned}&\widehat{\alpha}_k = \widehat{V}_k^t\widehat{r}_{k-1} = \widehat{V}_k^tL^{-1}r_k = (L^{-t}\widehat{V}_k)^tr_k,\\ &\widehat{x}_k = L^tx_k = \widehat{x}_{k-1} + \widehat{V}_k\widehat{\alpha}_k = L^tx_{k-1} + \widehat{V}_k\widehat{\alpha}_k, \quad \Longrightarrow\ x_k = x_{k-1} + (L^{-t}\widehat{V}_k)\widehat{\alpha}_k,\\ &\widehat{r}_k = L^{-1}r_k = \widehat{r}_{k-1} - \widehat{A}\widehat{V}_k\widehat{\alpha}_k = L^{-1}r_{k-1} - L^{-1}AL^{-t}\widehat{V}_k\widehat{\alpha}_k\\ \Longrightarrow\ &r_k = r_{k-1} - A(L^{-t}\widehat{V}_k)\widehat{\alpha}_k.\end{aligned}$$

Let $V_k = L^{-t}\widehat{V}_k$, then $\widehat{\alpha}_k = V_k^tr_k$, $x_k = x_{k-1} + V_k\widehat{\alpha}_k$, and $r_k = r_{k-1} - AV_k\widehat{\alpha}_k$. Moreover, $T(\widehat{r}_k) = T(L^{-1}r_k)$ and $\widehat{A}\widehat{V}_{k-1} = L^{-1}AL^{-t}\widehat{V}_{k-1} = L^{-1}AV_{k-1}$. As for the $\widehat{A}$-orthonormalization, we require that $\widehat{V}_k^t\widehat{A}\widehat{V}_i = 0$ for some values of $i \neq k$. But

$$\widehat{V}_k^t\widehat{A}\widehat{V}_i = \widehat{V}_k^tL^{-1}AL^{-t}\widehat{V}_i = (L^{-t}\widehat{V}_k)^tA(L^{-t}\widehat{V}_i) = V_k^tAV_i.$$

Thus, it is sufficient to A-orthonormalize $V_k = L^{-t}\widehat{V}_k$ instead of $\widehat{A}$-orthonormalizing $\widehat{V}_k$, where in modified MSDO-CG,

$$V_k = L^{-t}[T(\widehat{r}_{k-1})] = L^{-t}[T(L^{-1}r_k)],$$

and in SRE-CG and SRE-CG2

$$V_k = L^{-t}\widehat{V}_k = \begin{cases}L^{-t}[T(\widehat{r}_0)] & \text{if } k = 1,\\ L^{-t}L^{-1}AV_{k-1} = M^{-1}AV_{k-1} & \text{if } k \geq 2.\end{cases}$$

This summarizes the three methods for $s = 1$. In general, for $s > 1$ the s-step methods are described in Algorithms 8, 9, and 10. As for the communication avoiding versions, in Algorithms 5 and 7, AW_{j+i-1} is replaced by $M^{-1}AW_{j+i-1}$, and $W_{j-1} = L^{-t}[T(L^{-1}r_k)]$, for $k \geq 1$ in CA MSDO-CG and for $k = 1$ in CA SRE-CG and CA SRE-CG2.

Algorithm 8. Split preconditioned s-step SRE-CG.

```
    Input: $A$, $n \times n$ symmetric positive definite matrix; $k_{max}$, maximum allowed iterations; $b$, $n \times 1$ right-hand side; $x_0$, initial guess; $\epsilon$, stopping tolerance; $M = LL^t$; $s$, s-step
    Output: $x_k$, approximate solution of the system $L^{-t}AL^t(L^{-t}x) = L^{-t}b$
1:  $r_0 = b - Ax_0$, $\rho_0 = ||r_0||_2$ , $\rho = \rho_0$, $\widehat{r}_0 = L^{-1}r_0$, $k = 1$;
2:  while ( $\rho > \epsilon\rho_0$ and $k < k_{max}$ ) do
3:      Let $j = (k-1)s + 1$
4:      if $(k == 1)$ then
5:          A-orthonormalize $W_j = L^{-t}[T(\widehat{r}_0)]$, and let $V = W_j$
6:      else
7:          A-orthonormalize $W_j = M^{-1}AW_{j-1}$ against $W_{j-2}$ and $W_{j-1}$
8:          A-orthonormalize $W_j$ and let $V = W_j$
9:      end if
10:     for $(i = 1 : s-1)$ do
11:         A-orthonormalize $W_{j+i} = M^{-1}AW_{j+i-1}$ against $W_{j+i-2}$ and $W_{j+i-1}$
12:         A-orthonormalize $W_{j+i}$ and let $V = [V \; W_{j+i}]$
13:     end for
14:     $\widehat{\alpha} = V^t r_{k-1}$,     $x_k = x_{k-1} + V\widehat{\alpha}$
15:     $r_k = r_{k-1} - AV\widehat{\alpha}$,     $\rho = ||r_k||_2$,     $k = k + 1$
16: end while
```

Algorithm 9. Split preconditioned s-step SRE-CG2.

```
    Input: $A$, $n \times n$ symmetric positive definite matrix; $k_{max}$, maximum allowed iterations; $b$, $n \times 1$ right-hand side; $x_0$, initial guess; $\epsilon$, stopping tolerance; $M = LL^t$; $s$, s-step
    Output: $x_k$, approximate solution of the system $L^{-t}AL^t(L^{-t}x) = L^{-t}b$
1:  $r_0 = b - Ax_0$, $\rho_0 = ||r_0||_2$ , $\rho = \rho_0$, $\widehat{r}_0 = L^{-1}r_0$, $k = 1$;
2:  while ( $\rho > \epsilon\rho_0$ and $k < k_{max}$ ) do
3:      Let $j = (k-1)s + 1$
4:      if $(k == 1)$ then
5:          A-orthonormalize $W_j = L^{-t}[T(\widehat{r}_0)]$, let $Q = W_j$, and $V = W_j$
6:      else
7:          A-orthonormalize $W_j = M^{-1}AW_{j-1}$ against $Q$
8:          A-orthonormalize $W_j$, let $Q = [Q, \; W_j]$, and $V = W_j$
9:      end if
10:     for $(i = 1 : s-1)$ do
11:         A-orthonormalize $W_{j+i} = M^{-1}AW_{j+i-1}$ against $Q$
12:         A-orthonormalize $W_{j+i}$, let $V = [V, \; W_{j+i}]$ and $Q = [Q, \; W_{j+i}]$
13:     end for
14:     $\widehat{\alpha} = V^t r_{k-1}$,     $x_k = x_{k-1} + V\widehat{\alpha}$
15:     $r_k = r_{k-1} - AV\widehat{\alpha}$,     $\rho = ||r_k||_2$,     $k = k + 1$
16: end while
```

Algorithm 10. Split preconditioned s-step MSDO-CG.

```
Input: $A$, $n \times n$ symmetric positive definite matrix; $k_{max}$, maximum allowed iterations; $b$, $n \times 1$ right-hand side; $x_0$, initial guess; $\epsilon$, stopping tolerance; $M = LL^t$; $s$, s-step
Output: $x_k$, approximate solution of the system $L^{-t}AL^t(L^{-t}x) = L^{-t}b$
1: $r_0 = b - Ax_0$, $\rho_0 = ||r_0||_2$, $\rho = \rho_0$, $k = 1$;
2: while ( $\rho > \epsilon\rho_0$ and $k < k_{max}$ ) do
3:     $\widehat{r}_{k-1} = L^{-1}r_{k-1}$ and $W_1 = L^{-t}[T(\widehat{r}_{k-1})]$
4:     if ($k == 1$) then
5:         A-orthonormalize $W_1$, let $V = W_1$ and $Q = W_1$
6:     else
7:         A-orthonormalize $W_1$ against $Q$
8:         A-orthonormalize $W_1$, let $V = W_1$ and $Q = [Q\ W_1]$
9:     end if
10:    for ($i = 1 : s - 1$) do
11:        A-orthonormalize $W_{i+1} = M^{-1}AW_i$ against $Q$
12:        A-orthonormalize $W_{i+1}$, let $V = [V\ W_{i+1}]$ and $Q = [Q\ W_{i+1}]$
13:    end for
14:    $\widehat{\alpha} = V^t r_{k-1}$,    $x_k = x_{k-1} + V\widehat{\alpha}$
15:    $r_k = r_{k-1} - AV\widehat{\alpha}$,    $\rho = ||r_k||_2$,    $k = k + 1$
16: end while
```

If the preconditioner is a block diagonal preconditioner, with t blocks that correspond to the t partitions of matrix A, then $[T(L^{-1}r_k)]=L^{-1}[T(r_k)]$ and $L^{-t}[T(L^{-1}r_k)] = M^{-1}[T(r_k)]$. In this case, there is no need for split preconditioning, similarly to CG.

5.2. Convergence. We test the preconditioned versions using the block Jacobi preconditioner. First, the graphs of the matrices are partitioned into 64 domains using METIS k-way dissection [18]. Each of the 64 diagonal blocks is factorized using Cholesky decomposition (Table 5.2) or incomplete Cholesky zero fill-in decomposition (Table 5.1).

Then for a given t, each of the t domains is the union of $64/t$ consecutive domains, where the preconditioner $M = LL^t$, the L_i's are lower triangular blocks for $i = 1, 2, \ldots, 64$, and

$$L = \begin{bmatrix} L_1 & 0 & 0 & \ldots & 0 \\ 0 & \ddots & 0 & \ldots & 0 \\ 0 & 0 & L_i & \ldots & 0 \\ 0 & 0 & 0 & \ddots & 0 \\ 0 & 0 & \ldots & 0 & L_{64} \end{bmatrix}. \tag{5.1}$$

In Table 5.1, we show the convergence of incomplete Cholesky block Jacobi preconditioned s-step and CA versions of SRE-CG, SRE-CG2, and MSDO-CG for $t = 2, 4, 8, 16, 32, 64$ and $s = 1, 2, 4, 8$. For the matrices POISSON2D, NH2D, SKY2D, and ANI3D, and for all the s and t values, the preconditioned s-step versions and their corresponding CA versions with Algorithm 7 converge in the same number of iterations and scale even for $s \geq 8$. CA SRE-CG with Algorithm 5 stagnates, whereas CA SRE-CG2 with Algorithm 5 converges in exactly the same number of iterations

TABLE 5.1
Comparison of the convergence of different block Jacobi with incomplete Cholesky preconditioned E-CG versions (s-step SRE-CG, CA SRE-CG with Algorithm 7*, s-step SRE-CG*2*, CA SRE-CG*2 *with Algorithm* 5 *or* 7*, s-step MSDO-CG, CA MSDO-CG with Algorithm* 5 *and Algorithm* 7*) with respect to number of partitions t and s values.*

			SRE-CG							SRE-CG2							MSDO-CG									
			s-step				CA Alg7			s-step				CA Alg5/7			s-step				CA Alg5			CA Alg7		
	PCG	t \ s	1	2	4	8	2	4	8	1	2	4	8	2	4	8	1	2	4	8	2	4	8	2	4	8
Poisson2D	86	2	82	41	21	11	41	21	11	82	41	21	11	41	21	11	79	40	21	11	40	20	11	40	21	12
		4	65	33	17	9	33	17	9	65	33	17	9	33	17	9	71	36	18	9	35	18	9	36	18	10
		8	55	28	14	7	28	14	7	55	28	14	7	28	14	7	59	30	14	7	32	16	8	30	15	9
		16	41	21	11	6	21	11	6	41	21	11	6	21	11	6	51	26	12	6	27	14	7	26	12	7
		32	30	15	8	4	15	8	4	30	15	8	4	15	8	4	40	21	9	5	23	12	5	21	10	6
		64	23	12	6	3	12	6	3	23	12	6	3	12	6	3	30	16	7	3	19	9	4	16	8	5
Nh2D	115	2	105	53	27	14	53	27	14	105	53	27	14	53	27	14	106	53	27	13	54	27	14	53	27	15
		4	82	41	21	11	41	21	11	82	41	21	11	41	21	11	87	45	22	11	46	22	12	45	22	13
		8	65	33	17	9	33	17	9	65	33	17	9	33	17	9	71	36	17	9	38	19	10	36	18	11
		16	49	25	13	7	25	13	7	49	25	13	7	25	13	7	60	30	13	7	32	16	8	30	15	9
		32	36	18	9	5	18	9	5	36	18	9	5	18	9	5	45	24	11	5	27	13	6	24	12	7
		64	27	14	7	4	14	7	4	27	14	7	4	14	7	4	34	18	8	4	22	10	5	18	9	6
Sky2D	305	2	237	119	60	30	119	60	31	233	112	56	28	112	56	28	233	143	75	34	160	81	41	143	75	35
		4	135	68	34	17	68	34	18	131	66	33	17	66	33	17	193	124	49	20	135	68	36	124	47	22
		8	83	42	21	11	42	21	11	83	42	21	11	42	21	11	127	84	31	12	106	54	27	84	32	14
		16	54	27	14	7	27	14	7	54	27	14	7	27	14	7	94	56	20	8	80	41	20	56	20	10
		32	39	20	10	5	20	10	5	39	20	10	5	20	10	5	62	37	13	6	58	28	12	37	15	7
		64	29	15	8	4	15	8	4	29	15	8	4	15	8	4	43	25	9	4	41	21	8	25	11	6
Sky3D	245	2	216	108	56	28	108	56	29	201	103	52	26	103	52	25	203	116	50	26	117	69	30	116	60	34
		4	170	85	43	22	84	43	x	149	77	39	20	77	39	20	170	134	53	18	123	68	23	130	58	29
		8	108	54	27	14	55	28	x	101	51	26	13	51	26	13	140	99	32	14	124	55	16	99	36	18
		16	61	31	15	8	30	16	9	58	29	15	8	29	15	8	105	63	19	8	101	37	11	63	21	10
		32	34	17	9	5	18	9	5	34	17	9	5	17	9	5	77	38	11	5	71	23	7	38	13	7
		64	23	12	6	3	12	6	3	23	12	6	3	12	6	3	53	24	7	4	48	16	5	24	9	5
Ani3D	73	2	70	35	18	9	35	18	9	70	35	18	9	35	18	9	70	35	18	9	35	18	9	35	18	9
		4	63	32	16	8	32	16	8	63	32	16	8	32	16	8	66	33	16	9	33	17	9	33	17	10
		8	57	29	15	8	29	15	8	57	29	15	8	29	15	8	59	30	15	8	30	15	8	30	17	11
		16	50	25	13	7	25	13	7	50	25	13	7	25	13	7	54	27	14	7	28	14	7	27	16	10
		32	43	22	11	6	22	11	6	43	22	11	6	22	11	6	51	25	12	6	25	13	7	25	15	9
		64	35	18	9	5	18	9	5	35	18	9	5	18	9	5	44	21	10	5	23	11	6	21	13	7

as s-step SRE-CG2 and CA SRE-CG2 with Algorithm 7. Moreover, the corresponding preconditioned SRE-CG and SRE-CG2 versions converge in a similar number of iterations. As for SKY3D, the CA SRE-CG stagnates for $s = 8$ and $t = 4, 8$ only. The CA MSDO-CG with Algorithm 7 converges as fast as s-step MSDO-CG, whereas CA MSDO-CG with Algorithm 5 requires more iterations, in some cases (SKY2D, SKY3D).

A similar convergence behavior is observed for the complete Cholesky block Jacobi preconditioned s-step and CA versions of SRE-CG, SRE-CG2, and MSDO-CG in Table 5.2, where the only difference is that the methods converge faster than the corresponding incomplete Cholesky block Jacobi preconditioned versions.

Table 5.2
Comparison of the convergence of different block Jacobi with complete Cholesky preconditioned E-CG versions (s-step SRE-CG, CA SRE-CG with Algorithm 7*, s-step SRE-CG*2*, CA SRE-CG*2 *with* 5 *or* 7*, s-step MSDO-CG, CA MSDO-CG with Algorithm* 5 *and Algorithm* 7*) with respect to number of partitions t and s values.*

			SRE-CG							SRE-CG2							MSDO-CG										
			s-step				CA Alg7			s-step				CA Alg5/7			s-step				CA Alg5			CA Alg7			
	PCG	t \ s	1	2	4	8	2	4	8	1	2	4	8	2	4	8	1	2	4	8	2	4	8	2	4	8
Poisson2D	67	2	60	30	15	8	30	15	8	60	30	15	8	30	15	8	61	31	15	8	32	16	8	31	16	9
		4	51	26	13	7	26	13	7	51	26	13	7	26	13	7	53	27	14	7	27	14	7	27	14	8
		8	42	21	11	6	21	11	6	42	21	11	6	21	11	6	45	23	11	6	24	12	6	23	12	7
		16	33	17	9	5	17	9	5	33	17	9	5	17	9	5	37	20	9	5	21	10	5	20	10	6
		32	25	13	7	4	13	7	4	25	13	7	4	13	7	4	30	16	7	4	17	9	4	16	8	5
		64	20	10	5	3	10	5	3	20	10	5	3	10	5	3	23	12	6	3	14	7	3	12	6	4
Nh2D	92	2	74	37	19	10	37	19	10	74	37	19	10	37	19	10	81	40	19	10	41	21	11	40	21	12
		4	61	31	16	8	31	16	8	61	31	16	8	31	16	8	66	33	17	8	33	17	9	33	17	10
		8	51	26	13	7	26	13	7	51	26	13	7	26	13	7	55	28	13	7	29	14	7	28	14	9
		16	39	20	10	5	20	10	5	39	20	10	5	20	10	5	44	23	11	5	24	12	6	23	12	8
		32	30	15	8	4	15	8	4	30	15	8	4	15	8	4	34	19	8	4	20	10	5	19	9	6
		64	23	12	6	3	12	6	3	23	12	6	3	12	6	3	26	14	6	3	16	8	4	14	8	5
Sky2D	264	2	193	97	48	24	97	48	27	183	92	46	23	92	46	23	189	121	56	25	118	60	33	121	58	27
		4	105	53	27	14	53	27	14	105	53	27	14	53	27	14	146	91	37	16	106	54	27	91	38	17
		8	66	33	17	9	33	17	9	66	33	17	9	33	17	9	98	64	23	10	80	39	20	64	23	12
		16	44	22	11	6	22	11	6	44	22	11	6	22	11	6	70	41	15	7	58	28	13	41	17	8
		32	31	16	8	4	16	8	4	31	16	8	4	16	8	4	48	26	10	5	37	18	7	26	11	6
		64	19	10	5	3	10	5	3	19	10	5	3	10	5	3	30	17	6	3	21	10	4	17	7	4
Sky3D	225	2	181	91	48	24	94	48	26	173	87	46	23	87	46	23	186	106	48	25	114	61	28	106	49	32
		4	139	70	37	19	72	38	x	130	65	34	17	65	34	18	154	113	43	19	112	60	22	113	47	24
		8	80	40	20	10	40	20	14	77	39	20	10	39	20	10	117	76	26	11	99	48	15	76	28	14
		16	45	23	12	6	23	12	6	45	23	12	6	23	12	6	91	52	15	6	87	32	10	52	17	8
		32	29	15	8	4	15	8	4	29	15	8	4	15	8	4	62	29	9	4	57	20	6	29	10	6
		64	20	10	5	3	10	5	3	20	10	5	3	10	5	3	44	18	7	3	38	13	4	18	7	4
Ani3D	69	2	66	33	17	9	33	17	9	66	33	17	9	33	17	9	66	33	17	9	33	17	9	33	17	9
		4	61	31	16	8	31	16	8	61	31	16	8	31	16	8	61	31	15	8	31	16	8	31	16	10
		8	56	28	14	7	28	14	7	56	28	14	7	28	14	7	58	29	15	8	29	15	8	29	16	11
		16	49	25	13	7	25	13	7	49	25	13	7	25	13	7	54	27	13	7	28	14	7	27	16	10
		32	42	21	11	6	21	11	6	42	21	11	6	21	11	6	50	24	12	6	25	13	6	24	14	10
		64	35	18	9	5	18	9	5	35	18	9	5	18	9	5	44	20	10	5	21	11	5	20	12	7

6. Parallel performance model. The aim of this work is to introduce s-step versions of enlarged Krylov methods [11] that reduce communication in parallel by performing denser operations at a time, sending fewer but larger messages, and reducing the number of synchronizations between different processors in global reductions. Thus, these methods should converge faster in terms of runtime as compared to the corresponding enlarged Krylov methods, and eventually CG.

Moreover, even sequentially, the introduced s-step methods reduce communication with respect to their corresponding enlarged methods. This can be seen in Figure 6.1, where we show the runtime of the s-step and CA, SRE-CG2, and MSDO-CG algorithms for different t and s values for the matrix Nh2D using MATLAB 2015R on a 2015 MacBook Pro with dual 2.7 GHz Intel Core i5. The s-step version converges slightly faster than the corresponding enlarged versions, because the only difference is that they perform s times fewer saxpy's and dot products, whereas the CA versions converge up to 2.6 times faster than the corresponding enlarged versions, since they perform one (Algorithm 5) or two (Algorithm 7) A-orthonormalizations of blocks of st vectors per s-step iteration, as compared to s A-orthonormalizations of blocks of t vectors in s iterations. This obviously reduces the number of times the matrix A and the blocks of vectors are fetched from memory to cache, which is reflected in the runtime.

Figure 6.2 shows the runtime of the s-step and CA, SRE-CG2, and MSDO-CG algorithms for the matrix Sky3D. We observe the same behavior as for the matrix Nh2D with the exception that for $s = 2$ CA MSDO-CG with Algorithm 7 converges in more than half the iterations of MSDO-CG and thus performs more operations than MSDO-CG with approximately the same number of sent messages. This is reflected in the runtime, as CA MSDO-CG with Algorithm 7 takes more time to converge than MSDO-CG for $s = 2$.

In sections 6.1 and 6.2, we briefly describe the parallelization of the unpreconditioned and preconditioned s-step and CA SRE-CG, SRE-CG2, and MSDO-CG methods, assuming that the algorithms are executed on a distributed memory machine with t processors. Then, we compare the performance of the s-step and CA methods with respect to the SRE-CG, SRE-CG2, and MSDO-CG methods. Finally, in section 6.3, we compare the expected performance of the CA enlarged CG versions with respect to the classical CG, in terms of memory, flops, and communication.

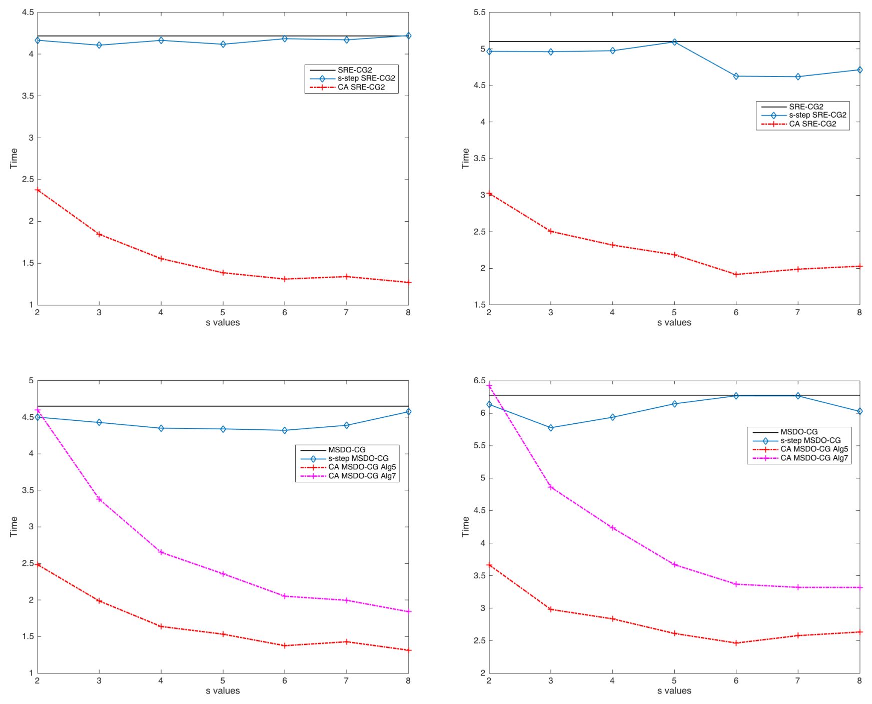

Fig. 6.1. *Sequential runtime in seconds of s-step and CA SRE-CG*2*, and MSDO-CG for* Nh2D *with* $t = 2, 4$ *respectively, and different* s *values.*

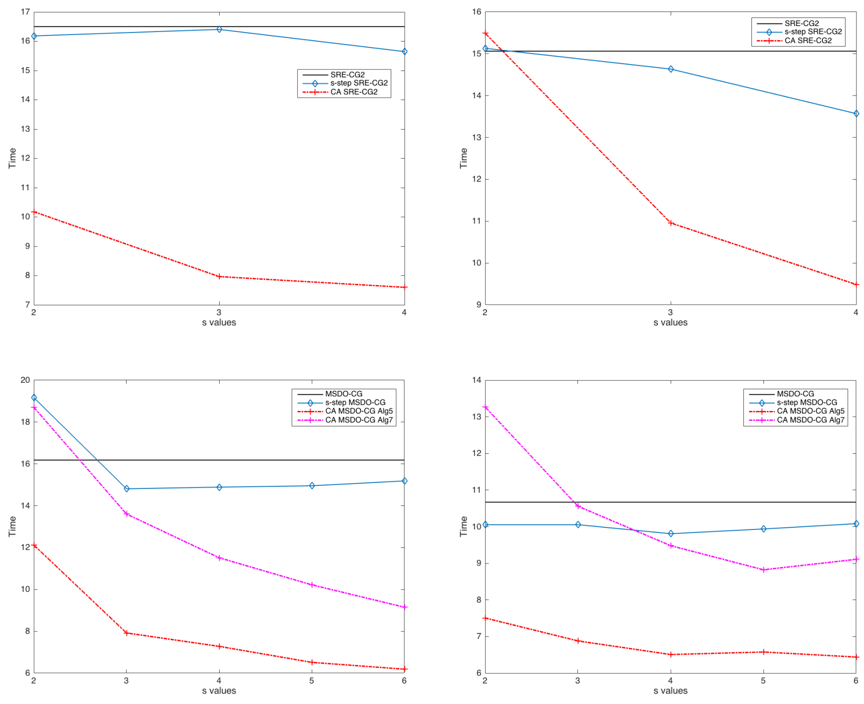

FIG. 6.2. *Sequential runtime in seconds of s-step and CA SRE-CG*2 *with* $t = 2, 4$*, and s-step and CA MSDO-CG with* $t = 4, 8$ *for* SKY3D *and different* s *values.*

In what follows, we assume that the estimated runtime of an algorithm with a total of z computed flops and s sent messages, each of size k, is $\gamma_c z + \alpha_c s + \beta_c sk$, where γ_c is time to perform a floating point operation on local data (seconds per floating-point operation), α_c is the latency cost incurred by every message (seconds), and β_c is the inverse bandwidth (seconds per word sent). We ignore the cost of data movement from main memory to cache and only consider the cost of data movement in parallel. Moreover, unless specified otherwise, we assume that the number of processors is equal to the number of partitions t.

6.1. Unpreconditioned methods. The unpreconditioned s-step SRE-CG and s-step SRE-CG2 parallelization is similar to that of SRE-CG and SRE-CG2 described in [11], with the difference that the s-step versions send $(s-1)\log(t)$ fewer messages and words per s-step iteration. Moreover, the s-step MSDO-CG's algorithm is similar to that of s-step SRE-CG in structure. Thus the number of messages sent in parallel is the same as that of s-step SRE-CG2. We assume that SRE-CG, SRE-CG2, and MSDO-CG converge in k iterations and the corresponding s-step versions converge in $k_s = \lceil \frac{k}{s} \rceil$ iteration. Thus, $5sk_s \log(t) + k_s \log(t) \approx 5k \log(t) + \frac{k}{s} \log(t)$ messages are sent in parallel in the s-step versions, compared to $6k \log(t)$ messages. This leads to a $\frac{(s-1)}{6s}$ reduction in communication, without increasing the number of computed flops. For example, for $s = 3$, 11.11% reduction is achieved in the s-step versions, and 15% reduction for $s = 10$.

The difference between the parallelization of unpreconditioned CA MSDO-CG and unpreconditioned CA SRE-CG2 is in the basis construction. In CA MSDO-CG each of the t processors can compute the s basis vectors

$$T_i(r_{k-1}), AT_i(r_{k-1}), A^2T_i(r_{k-1}), \ldots, A^{s-1}T_i(r_{k-1})$$

independently from other processors, where $T_i(r_{k-1})$ is a vector of all zeros except at n/t entries that correspond to the ith domain of matrix A. Thus, there is no need for communication avoiding kernels. To compute the s vectors without any communication, processor i needs $T_i(r_{k-1})$, the rowwise part of the vector r_{k-1} corresponding to the ith domain D_i, and a part of matrix A depending on s and the sparsity pattern of A. Specifically, processor i needs the columnwise part of A corresponding to $R(G(A), D_i, s)$, the set of vertices in the graph of A reachable by paths of length at most s from any vertex in D_i.

On the other hand, in CA SRE-CG2 at iteration k, the st basis vectors

$$AW_{(k-1)s}, A^2W_{(k-1)s}, \dots, A^sW_{(k-1)s}$$

have to be computed, where $W_{(k-1)s}$ is a block of t dense vectors. Similarly to CA MSDO-CG, each of the t processors can compute the s basis vectors

$$AW_{(k-1)s}(:,i), A^2W_{(k-1)s}(:,i), \dots, A^sW_{(k-1)s}(:,i)$$

independently from other processors. But processor i needs the full matrix A and the vector $W_{(k-1)s}(:,i)$. Another alternative is to use a block version of the matrix powers kernel, where processor i computes a rowwise part of the s blocks without any communication, by performing some redundant computations. Moreover, as discussed in section 4, for numerical stability purposes, $AW_{(k-1)s}$ has to be A-orthonormalized before proceeding in the basis construction. This increases the number of messages sent.

Then, all of the computed st vectors in CA MSDO-CG and CA SRE-CG2 are A-orthonormalized against the previous $st(k-1)$ vectors using CGS2 (Algorithm 18 in [22]) and against themselves using A-CholQR (Algorithm 21 in [22]) or Pre-CholQR (Algorithm 23 in [22]). The parallelization of the A-orthonormalization algorithms is described in details for a block of t vectors in [22]. For a block of st vectors, the same number of messages is sent but with more words. Thus, $5k_s \log(t) + k_s \log(t) \approx 6\frac{k}{s}\log(t)$ messages are sent in parallel in CA MSDO-CG, leading to a $\frac{(s-1)}{s}$ reduction in communication as compared to MSDO-CG (for $s = 3 \implies 66.6\%$ reduction), whereas in CA SRE-CG $2 * 5k_s \log(t) + k_s \log(t) \approx 11\frac{k}{s}\log(t)$ messages are sent, leading to a $\frac{(6s-11)}{6s}$ reduction in communication (for $s = 3 \implies 17.4\%$ reduction).

As for the unpreconditioned CA SRE-CG, its parallelization is exactly the same as that of CA SRE-CG2, where the same number of messages is sent in parallel. However, fewer words are sent per message, since in CA SRE-CG the st computed vectors are A-orthonormalized against the previous $(s+1)t$ vectors.

Thus the s-step and CA versions of the enlarged CG methods reduce communication as compared to their corresponding enlarged versions for the same number of processors and the same number of partitions t. All the s-step versions are comparable in terms of numerical stability and communication reduction. However, CA MSDO-CG is a better choice since it reduces communication the most. Knowing that the number of processors could be equal, a multiple or a divisor of the number of partitions t, the question that poses itself is, "Is it better, in terms of communication, to double t in MSDO-CG or merge two iterations of MSDO-CG?"

The number of flops performed per iteration in the s-step and CA MSDO-CG for $s = 2^i$ and a given t is comparable to that of the MSDO-CG algorithms where we have 2^it partitions. This is due to the fact that in both versions, we are constructing and A-orthonormalizing 2^it basis vectors per iteration for $\overline{\mathcal{K}}_{k,t,2^i}$ (s-step and CA MSDO-CG versions) and $\overline{\mathcal{K}}_{k,2^it,1}$ (MSDO-CG). On the other hand, based on the observed results in sections 4 and 5.2, by doubling t in any of the enlarged CG methods, the number of iterations needed for convergence is not halved, but empirically we observe that it is about a quarter less, whereas in s-step MSDO-CG and CA MSDO-CG, by doubling s the number of iterations is halved (up to some value s). In what follows, we assume that each of the three methods (MSDO-CG, s-step, and CA MSDO-CG) computes 2^it basis vectors per iteration, and we compare the number of sent messages for different numbers of processors.

In the first case, we assume that the number of processors for all the methods is equal to t. Let k be the number of iterations needed for convergence of MSDO-CG, where t basis vectors are computed per iteration. Then, the number of messages sent in parallel in MSDO-CG where we have 2^it partitions is $6(0.75)^ik\log(t)$, whereas for $s = 2^i$, $[5 + (0.5)^i]k\log(t)$ messages are sent in parallel in s-step MSDO-CG, and $6(0.5)^ik\log(t)$ messages are sent in CA MSDO-CG. But, for $i \geq 1$, we have that

$$6(0.5)^ik\log(t) < 6(0.75)^ik\log(t) < [5 + (0.5)^i]k\log(t).$$

Thus, in this case it is clear that doubling s and using the CA version is better than doubling the number of partitions in MSDO-CG, which is better than using the s-step version.

In the second case, we assume that the number of processors is equal to the number of partitions in each method. Then, the number of messages sent in parallel in MSDO-CG where we have 2^it processors and partitions is $6(0.75)^ik\log(2^it)$. However, in s-step MSDO-CG and CA MSDO-CG we assume that we have t processors and $s = 2^i$. In this case, s-step MSDO-CG sends fewer messages than MSDO-CG if and only if

$$\begin{aligned} &6(0.75)^i(i + \log(t)) > [5 + (0.5)^i]\log(t) \\ \iff\ &6i(0.75)^i > [5 - 6(0.75)^i + (0.5)^i]\log(t). \end{aligned} \tag{6.1}$$

The inequality (6.1) is valid for $i = 1$ and $t = 2, 4, 8, 16$. This means that for $s = 2$ s-step MSDO-CG requires less communication with $t = 2, 4, 8, 16$ processors/partitions than MSDO-CG with $2t$ processors/partitions. Hence, assuming that communication is much more expensive than flops, it is better to merge two iterations of MSDO-CG and compute a basis for $\overline{\mathcal{K}}_{k,t,2}$ than to double the t value and compute a basis for $\overline{\mathcal{K}}_{k,2t,1}$. Moreover, (6.1) is valid for $i = 2$ ($s = 4$) and $t = 2, 4, 8$, and for $i = 3, 4$ ($s = 8, 16$) and $t = 2, 4$. On the other hand, CA MSDO-CG sends fewer messages than MSDO-CG for all values of $i \geq 1$ and $t \geq 2$, since

$$\begin{aligned} &6(0.75)^i(i + \log(t)) > 6(0.5)^i\log(t) \\ \iff\ &i(0.75)^i > ((0.5)^i - (0.75)^i)\log(t). \end{aligned} \tag{6.2}$$

6.2. Preconditioned methods. The only difference between the preconditioned and unpreconditioned algorithms is in the matrix block of vectors multiplication where the preconditioner is applied. The parallelization of these "multiplications" depends on the type of the preconditioner and the sparsity pattern of A. Thus, if the preconditioned matrix block of vectors multiplication can be performed without communication, then the same number of messages will be sent per iteration. In what follows, we consider L to have the same format as (5.1), i.e., a block diagonal lower triangular matrix with t blocks, L_i, for $i = 1, \dots, t$.

In the s-step versions, the preconditioner is applied twice, $L^{-t}[T(\widehat{r}_{k-1})]$ and $W_{i+1} = M^{-1}AW_i$, where $\widehat{r}_{k-1} = L^{-1}r_{k-1}$, $M = LL^t$, and W_i is a dense $n \times t$

matrix. Then, $T_i(\widehat{r}_{k-1}) = T_i(L^{-1}r_{k-1}) = L^{-1}T_i(r_{k-1})$ is an all zero vector except the entries corresponding to domain D_i, which are obtained by solving $z_i = L_i^{-1}r_{k-1}(D_i)$ using forward substitution. Thus, processor i needs the ith diagonal block L_i and $T_i(r_{k-1})$ to compute $T_i(\widehat{r}_{k-1})$. Similarly, computing $L^{-t}T_i(\widehat{r}_{k-1})$ is reduced to computing $L_i^{-t}z_i$ using backward substitution. Thus, processor i computes the vector $L^{-t}[T_i(\widehat{r}_{k-1})]$ without any communication. As for $W_{j+1} = M^{-1}AW_j$, processor i computes the rowwise part of W_{j+1} corresponding to domain D_i, using the rowwise part of A, L_i, L_i^t, and the rowwise part of W_j corresponding to $\delta_i = R(G(A), D_i, 1)$. Thus, processor i computes $Z_i = A(D_i, \delta_i)W_j(\delta_i, :)$ without any communication and solves for $W_{j+1}(D_i, :) = (L_iL_i^t)^{-1}Z_i$ using a backward and forward substitution.

In preconditioned CA MSDO-CG, at the kth iteration the st vectors

$$L^{-t}T(\hat{r}_{k-1}), M^{-1}AL^{-t}T(\hat{r}_{k-1}), (M^{-1}A)^2L^{-t}T(\hat{r}_{k-1}), \ldots, (M^{-1}A)^{s-1}L^{-t}T(\hat{r}_{k-1})$$

are computed. Assuming that L is a block diagonal lower triangular matrix, then each of the t processors can compute the s basis vectors

$$L^{-t}T_i(\hat{r}_{k-1}), M^{-1}AL^{-t}T_i(\hat{r}_{k-1}), (M^{-1}A)^2L^{-t}T_i(\hat{r}_{k-1}), \ldots, (M^{-1}A)^{s-1}L^{-t}T_i(\hat{r}_{k-1})$$

independently from other processors, but using a relatively big columnwise part of A and M, depending on s and the sparsity patterns of A and L. Another alternative is that each of the t processors computes the rowwise part of the st vectors corresponding to D_i without communication using a preconditioned block version of the matrix powers kernel. However, the same relatively big columnwise part of A and M is needed.

To reduce the memory storage needed per processor, one option is to overlap communication with computation in the preconditioner's application. Let $W_1 = L^{-t}T(\hat{r}_{k-1})$ and $W_{j+1} = M^{-1}AW_j$ for $j \geq 1$. Each processor i can compute $W_1(D_i, :)$ independently, since $W_1(D_i, :)$ is all zeros except the ith column, which is equivalent to solving for $L_i^{-t}T_i(\hat{r}_{k-1})$. To compute $W_{j+1}(D_i, :) = L_i^{-t}L_i^{-1}Z_i$, where $Z_i = A(D_i, \delta_i)W_j(\delta_i, :)$, processor i needs part of $W_j(\delta_i, :)$ from neighboring processors. This local communication occurs once W_j is computed, and it is overlapped with the computation of $A(D_i, D_i)W_j(D_i, :)$. Then the remaining part of the multiplication is performed once the messages are received from neighboring processors. In this case, processor i only needs $A(D_i, \delta_i)$ and L_i. And there are $s-1$ communication phases, once before the last $s-1$ preconditioned matrix multiplications. Even though these local communications are hidden with computations, they might require some additional time. However, the gain in communication reduction from replacing s A-orthonormalization procedures by just one overweighs this "possible" additional communication, as the A-orthonormalization requires global communication.

In preconditioned CA SRE-CG and CA SRE-CG2, at the first iteration

$$L^{-t}T(\hat{r}_0), M^{-1}AL^{-t}T(\hat{r}_0), (M^{-1}A)^2L^{-t}T(\hat{r}_0), \ldots, (M^{-1}A)^{s-1}L^{-t}T(\hat{r}_0)$$

are computed, but at the kth iteration the st vectors

$$M^{-1}AW, (M^{-1}A)^2W, \ldots, (M^{-1}A)^sW$$

are computed, where $W = W_{(k-1)s}$ is a dense $n \times t$ matrix. The communication pattern and parallelization of the preconditioned matrix multiplication are the same as that of CA MSDO-CG, with the exception that for $k > 1$ an additional local communication is required for $M^{-1}AW$. Moreover, the communication reduction in the preconditioned CA SRE-CG2 is comparable to that of CA MSDO-CG, since once preconditioned, SRE-CG2 with Algorithm 5 converges and scales even for $s = 8$, as discussed in section 5.2.

6.3. Expected performance. By merging s iterations of the enlarged CG versions, communication is reduced in the corresponding s-step and CA versions as discussed in sections 6.1 and 6.2. Moreover, the enlarged CG versions converge faster in terms of iterations than CG by enlarging the Krylov subspace. However, are these reductions enough to obtain a method that converges faster than CG in terms of runtime, using comparable resources?

CG is known for its short recurrence formulae and the limited memory storage. In preconditioned CG (Algorithm 11), if processor i computes part of the vectors $p_k(D_i), w(D_i), x_k(D_i), r_k(D_i), \widehat{r}_k(D_i)$, then it needs $A(D_i, :)$, L_i, and $b(D_i)$, assuming that $M = LL^t$ is a block diagonal matrix. Moreover, two global communications are needed per iteration to perform the dot products $p^t w, \rho_k, \widehat{\rho}_k$, and local communication with neighboring processors is needed to compute $w(D_i) = A(D_i, \delta_i)p(\delta_i)$, where $\delta_i = R(G(A), D_i, 1)$. Given that there is a total of m processors, $2\log(m)$ messages are sent per CG iteration without considering local communication.

Algorithm 11. Preconditioned CG.

```
   Input: $A$, $M = LL^t$, $b$, $x_0$, $\epsilon$, $k_{max}$
   Output: $x_k$, the approximate solution of the system $L^{-t}AL^t(L^{-t}x) = L^{-t}b$
1: $r_0 = b - Ax_0$, $\rho_0 = ||r_0||_2^2$, $\widehat{r}_0 = M^{-1}r_0$, $\widehat{\rho}_0 = r_0^t\widehat{r}_0$, $k = 1$;
2: while ( $\sqrt{\rho_{k-1}} > \epsilon\sqrt{\rho_0}$ and $k \leq k_{max}$ ) do
3:     if ($k == 1$) then $p = \widehat{r}_0$
4:     else $\beta = \frac{\widehat{\rho}_{k-1}}{\widehat{\rho}_{k-2}}$, $p = \widehat{r}_k + \beta p$
5:     end if
6:     $w = Ap$,    $\alpha = \dfrac{\widehat{\rho}_{k-1}}{p^t w}$
7:     $x_k = x_{k-1} + \alpha p$,    $r_k = r_{k-1} - \alpha w$,    $\widehat{r}_k = M^{-1}r_k$
8:     $\rho_k = ||r_k||_2^2$,    $\widehat{\rho}_k = r_k^t\widehat{r}_k$,    $k = k + 1$
9: end while
```

Similarly to CG, the SRE-CG, SRE-CG2, and MSDO-CG versions have short recurrence formulae. However, in terms of memory, the SRE-CG versions are the best choice since a limited number of vectors, depending only on t and s, need to be stored. In SRE-CG, s-step SRE-CG, and CA SRE-CG, $(3t)$ vectors, $(s+2)t$ vectors, and $(2s+1)t$ vectors are stored, respectively, whereas in the SRE-CG2 and MSDO-CG versions, stk vectors need to be stored, where k is the number of iterations needed for convergence which is not known a priori.

Given that $s = 2^i$, the number of partitions is $t = 2^j$, and a total of m processors run the algorithms where m is a multiple, divisor, or equal to $t = 2^j$ for $j, i \geq 1$; then, $2k\log(m)$ messages are sent in CG that converges k iterations. As for SRE-CG, a total of $6(0.75)^j k\log(m)$ messages are sent, assuming that as t is doubled the number of iterations is reduced by 25% on average, whereas in s-step and CA SRE-CG, a total of $[5+(0.5)^i](0.75)^j k\log(m)$ and $11(0.5)^i(0.75)^j k\log(m)$ messages are sent, respectively.

As compared to CG, SRE-CG, s-step SRE-CG, and CA SRE-CG communicate less in total when

$$\text{SRECG:} \quad 2k\log(m) > 6(0.75)^j k\log(m), \tag{6.3}$$

$$\text{s-step SRECG:} \quad 2k\log(m) > [5+(0.5)^i](0.75)^j k\log(m), \tag{6.4}$$

$$\text{CA SRE-CG:} \quad 2k\log(m) > 11(0.5)^i(0.75)^j k\log(m). \tag{6.5}$$

For $j \geq 4$, inequality (6.3) is satisfied, i.e., SRE-CG reduces communication with respect to CG for number of partitions $t \geq 16$. Similarly, s-step SRE-CG further reduces communication with respect to CG for $s = 2, 4, 8$ and $t \geq 16$, whereas CA SRE-CG reduces communication for $s = 2$ and $j \geq 4$ ($t \geq 16$), $s = 4$ and $j \geq 2$ ($t \geq 4$), and $s = 8$ and $j \geq 1$ ($t \geq 2$).

Hence, for $s = 2^i$ and $t = 2^j$ in the SRE-CG, s-step SRE-CG, and CA SRE-CG, the relative reduction in communication with respect to CG is respectively ($j = 5$ and $i = 2$)

$$\begin{aligned}
\frac{2k\log(m) - 6(0.75)^j k\log(m)}{2k\log(m)} &= 1 - 3(0.75)^j &&= (0.288),\\
\frac{2k\log(m) - [5+(0.5)^i](0.75)^j k\log(m)}{2k\log(m)} &= 1 - (2.5 + 0.5^{i+1})0.75^j &&= (0.377),\\
\frac{2k\log(m) - 11(0.5)^i(0.75)^j k\log(m)}{2k\log(m)} &= 1 - 5.5(0.5)^i(0.75)^j &&= (0.674).
\end{aligned}$$

Thus, it is expected that SRE-CG, s-step SRE-CG, and CA SRE-CG will converge faster than CG in parallel considering the s and t values discussed above and that communication is much more expensive than flops. Similarly, the s-step and CA versions of the SRE-CG2 and MSDO-CG are expected to converge faster than CG but require much more memory storage per processor.

7. Conclusion.

In this paper, we introduced the s-step and communication avoiding versions of SRE-CG, and SRE-CG2, which are based on the enlarged Krylov subspace. We have also introduced a modified MSDO-CG version that is equivalent to MSDO-CG theoretically and numerically but based on a modified enlarged Krylov subspace which allows the s-step and CA formulations. The split preconditioned s-step and CA versions are also presented in section 5.

The s-step and communication avoiding versions merge s iterations of the enlarged CG methods into one iteration where denser operations are performed for less communication. Numerical stability of the s-step and CA version is tested in section 4, where as s is doubled, the number of iterations needed for convergence in the s-step methods is roughly divided by two, even for $s \geq 10$. As for the CA methods, once the system is preconditioned, a similar scaling behavior is observed in section 5.2. Accordingly, it is shown in sections 6.1 and 6.2 that the s-step and CA versions reduce communication with respect to the corresponding enlarged methods for $s \geq 2$.

The number of messages per iteration of the enlarged CG methods and their s-step and CA versions is more than that of CG. However, due to the reduction in the number of iterations in the enlarged versions, the total messages sent is less as discussed in section 6.3. This implies that all the enlarged CG variants should require less time to converge than CG. However, the SRE-CG variants are the most feasible candidates due to their limited memory storage requirements.

Future work will focus on implementing, testing, and comparing the runtime of the introduced enlarged CG versions on CPUs and GPUs with respect to existing similar methods.

REFERENCES

[1] E. Carson, N. Knight, and J. Demmel, *An efficient deflation technique for the communication-avoiding conjugate gradient method*, Electron. Trans. Numer. Anal., 43 (2014), pp. 125–141.

[2] E. Carson, N. Knight, and J. Demmel, *Avoiding Communication in Two-Sided Krylov Subspace Methods*, Technical report UCB/EECS-2011-93, EECS Department, University of California, Berkeley, 2011.

[3] A. Chapman and Y. Saad, *Deflated and augmented Krylov subspace techniques*, Numer. Linear Algebra Appl., 4 (1997), pp. 43–66.

[4] A. T. Chronopoulos and W. Gear, *s-step iterative methods for symmetric linear systems*, J. Comput. Appl. Math., 25 (1989), pp. 153–168.

[5] J. Demmel, M. Heath, and H. van der Vorst, *Parallel Numerical Linear Algebra*, Acta Numer., 2 (1993), pp. 111–197.

[6] J. Erhel, *A parallel GMRES version for general sparse matrices*, Electron. Trans. Numer. Anal., 3 (1995), pp. 160–176.

[7] J. Erhel and F. Guyomarc'h, *An augmented conjugate gradient method for solving consecutive symmetric positive definite linear systems*, SIAM J. Matrix Anal. Appl., 21 (2000), pp. 1279–1299.

[8] R. Fletcher, *Gradient Methods for Indefinite Systems*, in Numerical Analysis, Lecture Notes in Math. 506, G. A. Watson, ed., Springer, Berlin, 1976, pp. 73–89.

[9] P. Ghysels, T. J. Ashby, K. Meerbergen, and W. Vanroose, *Hiding global communication latency in the GMRES algorithm on massively parallel machines*, SIAM J. Sci. Comput., 35 (2013), pp. 48–71.

[10] L. Grigori and S. Moufawad, *Communication avoiding ILU0 preconditioner*, SIAM J. Sci. Comput., 37 (2015), pp. 217–246.

[11] L. Grigori, S. Moufawad, and F. Nataf, *Enlarged Krylov subspace conjugate gradient methods for reducing communication*, SIAM J. Matrix Anal. Appl., 37 (2016), pp. 744–773.

[12] L. Grigori and O. Tissot, *Reducing the Communication and Computational Costs of Enlarged Krylov Subspaces Conjugate Gradient*, Research report RR-9023, Inria, Paris, 2017.

[13] W. Gropp, *Update on Libraries for Blue Waters*, http://jointlab.ncsa.illinoisedu/events/workshop3/pdf/presentations/Gropp-Update-on-Libraries.pdf.

[14] T. Gu, X. Liu, Z. Mo, and X. Chi, *Multiple search direction conjugate gradient method* I: *Methods and their propositions*, Int. J. Comput. Math., 81 (2004), pp. 1133–1143.

[15] F. Hecht, *New development in FreeFEM++*, J. Numer. Math., 20 (2012), pp. 251–265.

[16] M. R. Hestenes and E. Stiefel, *Methods of conjugate gradients for solving linear systems*, J. Res. National Bureau of Standards, 49 (1952), pp. 409–436.

[17] M. Hoemmen, *Communication-Avoiding Krylov Subspace Methods*, Ph.D. thesis, EECS Department, University of California, Berkeley, 2010.

[18] G. Karypis and V. Kumar, *METIS* 4.0: *Unstructured Graph Partitioning and Sparse Matrix Ordering System*, Technical report, Department of Computer Science, University of Minnesota, 1998.

[19] C. Lanczos, *Solution of systems of linear equations by minimized iterations*, J. Res. National Bureau of Standards, 49 (1952), pp. 33–53.

[20] B. Lowery and J. Langou, *Stability Analysis of QR Factorization in an Oblique Inner Product*, https://arxiv.org/abs/1401.5171, 2014.

[21] M. Mohiyuddin, M. Hoemmen, J. Demmel, and K. Yelick, *Minimizing Communication in Sparse Matrix Solvers*, in Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis, ACM, New York, 2009, pp. 1–12.

[22] S. Moufawad, *Enlarged Krylov Subspace Methods and Preconditioners for Avoiding Communication*, Ph.D. thesis, Université Pierre et Marie Curie-Paris VI, 2014.

[23] D. P. O'Leary, *The block conjugate gradient algorithm and related methods*, Linear Algebra Appl., 29 (1980), pp. 293–322.

[24] M. Rozloznik, M. Tuma, A. Smoktunowicz, and J. Kopal, *Numerical stability of orthogonalization with a non-standard inner product*, BIT, 52 (2012), pp. 1035–1058.

[25] Y. Saad, *Analysis of augmented Krylov subspace methods*, SIAM J. Matrix Anal. Appl., 18 (1997), pp. 435–449.

[26] 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.

[27] H. Van der Vorst, *Bi-CGSTAB: A fast and smoothly converging variant of Bi-CG for the solution of nonsymmetric linear systems*, SIAM J. Sci. Stat. Comput., 13 (1992), pp. 631–644.

[28] J. Van Rosendale, *Minimizing inner product data dependence in conjugate gradient iteration*, in Proceeding of the IEEE International Conference on Parallel Processing, 1983, pp. 44–46.

[29] H. F. Walker, *Implementation of the GMRES method using Householder transformations*, SIAM J. Sci. Stat. Comput., 9 (1988), pp. 152–163.