Numerical Properties and Scalability of s-Step Preconditioned Conjugate Gradient Methods

Viktoria Mayer
University of Vienna, UniVie Doctoral School Computer Science DoCS, Faculty of Computer Science
Vienna, Austria
viktoria.mayer@univie.ac.at

Wilfried N. Gansterer
University of Vienna, Faculty of Computer Science
Vienna, Austria
wilfried.gansterer@univie.ac.at

Abstract

s-step Preconditioned Conjugate Gradient (PCG) variants for iteratively solving large sparse linear systems reduce the number of global synchronization points of standard PCG by a factor of $O(s)$. Despite improving scalability on large-scale parallel computers, they have worse numerical properties than standard PCG. Choosing a suitable basis type for the *s*-step basis matrices is known to potentially improve numerical stability strongly. The *s*-step method proposed first in the literature was designed to only use the monomial basis. We generalize this method to support arbitrary basis types, denoting our new method as sPCG.

Moreover, we theoretically and experimentally compare all *s*-step PCG methods. To the best of our knowledge, this is the first comprehensive comparison in the literature. Our theoretical analysis, strong scaling experiments with a synthetic test problem, and runtime experiments with real-world problems confirm that our novel sPCG algorithm achieves higher speedup over standard PCG than existing *s*-step algorithms.

CCS Concepts

• **Mathematics of computing** → **Solvers**; • **Computing methodologies** → **Parallel algorithms**.

Keywords

Communication-avoiding s-step conjugate gradient algorithms, extreme-scale parallel computing, numerical stability

1 Introduction

The Preconditioned Conjugate Gradient (PCG) algorithm is an important Krylov subspace method for solving large sparse linear systems $\boldsymbol{Ax} = \boldsymbol{b}$ with a sparse symmetric and positive-definite (SPD) system matrix $\boldsymbol{A}$. On large-scale parallel computers, global collective operations are required in PCG to compute scalar products of distributed dense vectors. These global collectives become major bottlenecks due to their limited parallel scalability.

To reduce these performance limitations, scalable PCG variants have been developed: *Communication-hiding* PCG methods overlap global communication with local communication and computation [6, 9, 12, 19], while *communication-avoiding s-step* methods reduce the number of global synchronizations by rearranging PCG so that *s* iterations can be computed without communication [7, 14, 21].

In this paper, we focus on approaches for *reducing* the communication cost. Therefore, we do not consider communication-hiding or pipelined PCG methods and leave the comparison of *s*-step methods and state-of-the-art pipelined methods for future work.

s-step methods enhance the Krylov subspace by *s* vectors at once by computing a basis of dimension $O(s)$ in each iteration for the next *s* steps. The earliest *s*-step PCG method proposed by [7] computes the PCG iterations in blocks of *s* and thus reduces global communication latency by a factor of $O(s)$. Moreover, local computations can efficiently utilize BLAS2 and BLAS3, replacing BLAS1 and BLAS2 required in standard PCG. We denote this method as sPCG_{mon}.

Two *s*-step PCG algorithms have been proposed after sPCG_{mon}: CA-PCG [21] and CA-PCG3 [14]. Both achieve the same global communication reduction as sPCG_{mon}. However, CA-PCG requires more preconditioner applications and matrix vector (MV) products than standard PCG and sPCG_{mon}, making it only suitable for very simple and cheap preconditioners and very sparse input matrices. CA-PCG3 on the other hand requires the usage of BLAS1, which leads to performance drawbacks in the local computations.

Although communication-avoiding PCG methods are mathematically equivalent to standard PCG, they have different numerical error propagation, which may slow down or prevent convergence in practice [1]. When designing algorithms for large-scale parallel computers, a reduction of communication bottlenecks should also maintain numerical stability.

CA-PCG3 is based on three-term recurrence relations, which are known to be numerically less stable than the two-term recurrences of standard PCG [13]. This *s*-step method has been applied in large-scale simulations, where only a limited value of *s* was possible due to numerical round-off errors [15, 16].

Several strategies to improve the numerical stability of CA-PCG [21] have been presented, e.g., in [2, 3, 5]. Some of these ideas might be applicable to other *s*-step methods as well. However, choosing a basis type different from the monomial basis (e.g., the Newton or the Chebyshev basis) for generating the *s*-step basis matrices is considered to be the most important strategy to achieve a similar numerical stability and convergence speed as standard PCG [14].

sPCG_{mon} suffers from poor numerical stability as, unlike CA-PCG and CA-PCG3, it is formulated such that its *s*-step basis matrices can only be computed based on the monomial basis [8, 21]. Because of its advantages over CA-PCG and CA-PCG3 in terms of performance mentioned before, we propose extensions for sPCG_{mon} that enable the usage of arbitrary basis types and thus improve numerical stability.

Contributions of this work. We extend sPCG_{mon} [8] for enabling the usage of arbitrary basis types to improve its numerical stability, denoting our novel extended version as sPCG. Moreover, we provide a theoretical comparison of sPCG and the other two existing *s*-step PCG methods, demonstrating that sPCG is the least

Algorithm 1 Preconditioned Conjugate Gradient method (PCG)

$\boldsymbol{r}^{(0)} = \boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}^{(0)}$, $\boldsymbol{u}^{(0)} = \boldsymbol{M}^{-1}\boldsymbol{r}^{(0)}$, $\boldsymbol{p}^{(0)} = \boldsymbol{u}^{(0)}$
for $i = 0, 1, \ldots$ until convergence **do**
  $\boldsymbol{s}^{(i)} = \boldsymbol{A}\boldsymbol{p}^{(i)}$
  $\alpha^{(i)} = \boldsymbol{r}^{(i)^T}\boldsymbol{u}^{(i)} / \boldsymbol{p}^{(i)^T}\boldsymbol{s}^{(i)}$
  $\boldsymbol{x}^{(i+1)} = \boldsymbol{x}^{(i)} + \alpha^{(i)}\boldsymbol{p}^{(i)}$ ▷ Approximate solution
  $\boldsymbol{r}^{(i+1)} = \boldsymbol{r}^{(i)} - \alpha^{(i)}\boldsymbol{s}^{(i)}$ ▷ Unpreconditioned residual
  $\boldsymbol{u}^{(i+1)} = \boldsymbol{M}^{-1}\boldsymbol{r}^{(i+1)}$ ▷ Preconditioned residual
  $\beta^{(i+1)} = \boldsymbol{r}^{(i+1)^T}\boldsymbol{u}^{(i+1)} / \boldsymbol{r}^{(i)^T}\boldsymbol{u}^{(i)}$
  $\boldsymbol{p}^{(i+1)} = \boldsymbol{u}^{(i+1)} + \beta^{(i+1)}\boldsymbol{p}^{(i)}$ ▷ Search direction
end for

expensive. In numerical experiments on the Austrian Scientific Computing (ASC) infrastructure, we also illustrate experimentally that sPCG achieves larger speedup over standard PCG than the other s-step PCG variants. To the best of our knowledge, this is the first systematic comparison of different s-step PCG variants in terms of numerical stability and sustained performance.

Synopsis. In Section 2, we summarize the existing s-step PCG methods sPCG_{mon} [7], CA-PCG [21] and CA-PCG3 [14]. We extend sPCG_{mon} such that it can be used with arbitrary basis types in Section 3, proposing our new extended method sPCG. In Section 4, we provide a theoretical cost analysis for all three s-step algorithms and demonstrate that sPCG is superior to the other considered solvers in terms of computational cost. Section 5 provides an experimental evaluation of the performance of all the s-step solvers, both in terms of numerical stability and runtime performance. We conclude our work in Section 6.

2 Existing methods

The PCG method iteratively solves a system of linear equations $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$ for the solution $\boldsymbol{x} \in \mathbb{R}^n$ with the sparse SPD system matrix $\boldsymbol{A} \in \mathbb{R}^{n\times n}$ and the right-hand side $\boldsymbol{b} \in \mathbb{R}^n$. A preconditioner $\boldsymbol{M}^{-1} \in \mathbb{R}^{n\times n}$ is used to accelerate convergence. Standard PCG is shown in Algorithm 1. In the remainder of this section, we review the s-step algorithms sPCG_{mon}, CA-PCG and CA-PCG3.

2.1 s-step PCG for monomial basis (sPCG_{mon})

The s-step PCG method sPCG_{mon} proposed by [7] computes the iterations of standard PCG in blocks of s, global reduction operations are only occurring every s steps. In each iteration k, s search directions are computed. To update the approximate solution, we compute

$$\boldsymbol{x}^{(k+1)} = \boldsymbol{x}^{(k)} + \boldsymbol{P}^{(k)}\boldsymbol{a}^{(k)}$$

with the search direction matrix $\boldsymbol{P}^{(k)} = [\boldsymbol{p}^{(i)}, \ldots, \boldsymbol{p}^{(i+s-1)}]$ and the vector $\boldsymbol{a}^{(k)}$ of length s that replaces the coefficients $\alpha^{(i)}, \ldots, \alpha^{(i+s-1)}$, where the index i refers to the respective iteration in standard PCG, i.e. $\lfloor i/s \rfloor = k$. To perform s steps without communication, the basis matrices $\boldsymbol{R}^{(k)} = [\boldsymbol{r}^{(k)}, \boldsymbol{A}\boldsymbol{M}^{-1}\boldsymbol{r}^{(k)}, \ldots, (\boldsymbol{A}\boldsymbol{M}^{-1})^{s-1}\boldsymbol{r}^{(k)}]$ and $\boldsymbol{U}^{(k)} = \boldsymbol{M}^{-1}\boldsymbol{R}^{(k)}$ are computed. Analogously to PCG, we update the search directions with

$$\boldsymbol{P}^{(k)} = \boldsymbol{U}^{(k)} + \boldsymbol{P}^{(k-1)}\boldsymbol{B}^{(k)}, \tag{1}$$

where the matrix $\boldsymbol{B}^{(k)} \in \mathbb{R}^{s\times s}$ replaces the coefficients $\beta^{(i)}$ in PCG.

Algorithm 2 s-step PCG for the monomial basis (sPCG_{mon}) [7]

$\boldsymbol{r}^{(0)} = \boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}^{(0)}$, $\boldsymbol{P}^{(0)} = 0_{n,s}$, $\boldsymbol{A}\boldsymbol{P}^{(0)} = 0_{n,s}$
for $k = 0, 1, \ldots$ until convergence **do**
  $\boldsymbol{u}^{(k)} = \boldsymbol{M}^{-1}\boldsymbol{r}^{(k)}$
  $\boldsymbol{S}^{(k)} = \left[\boldsymbol{r}^{(k)}, (\boldsymbol{A}\boldsymbol{M}^{-1})\boldsymbol{r}^{(k)}, \ldots, (\boldsymbol{A}\boldsymbol{M}^{-1})^s\boldsymbol{r}^{(k)}\right]$
  $\boldsymbol{U}^{(k)} = \left[\boldsymbol{u}^{(k)}, (\boldsymbol{M}^{-1}\boldsymbol{A})\boldsymbol{u}^{(k)}, \ldots, (\boldsymbol{M}^{-1}\boldsymbol{A})^{s-1}\boldsymbol{u}^{(k)}\right]$
  $\boldsymbol{R}^{(k)}$ = first s columns of $\boldsymbol{S}^{(k)}$
  $\boldsymbol{A}\boldsymbol{U}^{(k)}$ = last s columns of $\boldsymbol{S}^{(k)}$
  $\boldsymbol{a}^{(k)}, \boldsymbol{B}^{(k)} \leftarrow$ Scalar Work $(\boldsymbol{S}^{(k)}, \boldsymbol{U}^{(k)}, \boldsymbol{a}^{(k-1)})$
  $\boldsymbol{P}^{(k)} = \boldsymbol{U}^{(k)} + \boldsymbol{P}^{(k-1)}\boldsymbol{B}^{(k)}$
  $\boldsymbol{A}\boldsymbol{P}^{(k)} = \boldsymbol{A}\boldsymbol{U}^{(k)} + \boldsymbol{A}\boldsymbol{P}^{(k-1)}\boldsymbol{B}^{(k)}$
  $\boldsymbol{x}^{(k+1)} = \boldsymbol{x}^{(k)} + \boldsymbol{P}^{(k)}\boldsymbol{a}^{(k)}$
  $\boldsymbol{r}^{(k+1)} = \boldsymbol{r}^{(k)} - \boldsymbol{A}\boldsymbol{P}^{(k)}\boldsymbol{a}^{(k)}$
end for

The routine "Scalar Work" computes $\boldsymbol{a}^{(k)}$ and $\boldsymbol{B}^{(k)}$ and requires only one global reduction operation. Since it requires the matrix $\boldsymbol{A}\boldsymbol{U}^{(k)}$, the s-step basis matrix $\boldsymbol{R}^{(k)}$ is enhanced by one additional vector. This enhanced basis matrix is denoted as $\boldsymbol{S}^{(k)}$.

As in [20], we compute the residual $\boldsymbol{r}^{(k)}$ recursively. This avoids the additional MV product per s steps of the original algorithm in [7].

$$\boldsymbol{r}^{(k+1)} = \boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}^{(k+1)} = \boldsymbol{r}^{(k)} - \boldsymbol{A}\boldsymbol{P}^{(k)}\boldsymbol{a}^{(k)} \tag{2}$$

The matrix $\boldsymbol{A}\boldsymbol{P}^{(k)}$ is computed recursively. Multiplying (1) with $\boldsymbol{A}$, we obtain

$$\boldsymbol{A}\boldsymbol{P}^{(k)} = \boldsymbol{A}\boldsymbol{U}^{(k)} + \boldsymbol{A}\boldsymbol{P}^{(k-1)}\boldsymbol{B}^{(k)}.$$

sPCG_{mon} is outlined in Algorithm 2.

2.2 Communication-avoiding PCG (CA-PCG)

In the communication-avoiding PCG (CA-PCG) method presented in [21], the vectors of standard PCG are linearly transformed such that the next s steps can be computed in a changed basis without communication.

CA-PCG divides the loop of PCG into an outer and an inner loop. The outer loop $k = 0, 1, \ldots$ iterates until convergence or if a user-defined maximum number of iterations is reached. Communication takes place only at the beginning of each outer iteration. The iterations of the inner loop $j = 0, \ldots, s-1$ are performed without communication.

We denote the unpreconditioned search direction by $\boldsymbol{q}^{(i)}$, i.e. $\boldsymbol{p}^{(i)} = \boldsymbol{P}\boldsymbol{q}^{(i)}$. By induction, for $0 \le j \le s$ and $s \ge 1$,

$$\boldsymbol{r}^{(sk+j)}, \boldsymbol{q}^{(sk+j)} \in \mathcal{K}_{s+1}\left(\boldsymbol{A}\boldsymbol{M}^{-1}, \boldsymbol{q}^{(sk)}\right) + \mathcal{K}_s\left(\boldsymbol{A}\boldsymbol{M}^{-1}, \boldsymbol{r}^{(sk)}\right),$$
$$\boldsymbol{u}^{(sk+j)}, \boldsymbol{p}^{(sk+j)}, \boldsymbol{x}^{(sk+j)} - \boldsymbol{x}^{(sk)} \in \mathcal{K}_{s+1}\left(\boldsymbol{M}^{-1}\boldsymbol{A}, \boldsymbol{p}^{(sk)}\right) + \mathcal{K}_s\left(\boldsymbol{M}^{-1}\boldsymbol{A}, \boldsymbol{u}^{(sk)}\right).$$

At the beginning of outer iteration k, the s-step basis matrices $\boldsymbol{Y}^{(k)} = \left[\boldsymbol{Q}^{(k)}, \boldsymbol{R}^{(k)}\right]$ and $\boldsymbol{Z}^{(k)} = \boldsymbol{P}\boldsymbol{Y}^{(k)} = \left[\boldsymbol{P}^{(k)}, \boldsymbol{U}^{(k)}\right]$ are computed such that $\text{span}\left(\boldsymbol{Q}^{(k)}\right) = \mathcal{K}_{s+1}\left(\boldsymbol{A}\boldsymbol{M}^{-1}, \boldsymbol{q}^{(sk)}\right)$, $\text{span}\left(\boldsymbol{R}^{(k)}\right) = \mathcal{K}_s\left(\boldsymbol{A}\boldsymbol{M}^{-1}, \boldsymbol{r}^{(sk)}\right)$, $\boldsymbol{P}^{(k)} = \boldsymbol{M}^{-1}\boldsymbol{Q}^{(k)}$ and $\boldsymbol{U}^{(k)} = \boldsymbol{M}^{-1}\boldsymbol{R}^{(k)}$. Subsequently, the Gram matrix $\boldsymbol{G}^{(k)} = \boldsymbol{Y}^{(k)^T}\boldsymbol{Z}^{(k)}$ of size $(2s+1)\times(2s+1)$

Algorithm 3 Communication-avoiding PCG (CA-PCG) [21]

$\boldsymbol{r}^{(0)} = \boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}^{(0)}, \boldsymbol{u}^{(0)} = \boldsymbol{M}^{-1}\boldsymbol{r}^{(0)}, \boldsymbol{q}^{(0)} = \boldsymbol{r}^{(0)}, \boldsymbol{p}^{(0)} = \boldsymbol{u}^{(0)}$
for $k = 0, 1, \ldots$ until convergence **do**
  $\text{span}\left(Y^{(k)}\right) = \mathcal{K}_{s+1}\left(\boldsymbol{A}\boldsymbol{M}^{-1}, \boldsymbol{q}^{(k)}\right) + \mathcal{K}_s\left(\boldsymbol{A}\boldsymbol{M}^{-1}, \boldsymbol{r}^{(k)}\right)$
  $\text{span}\left(Z^{(k)}\right) = \mathcal{K}_{s+1}\left(\boldsymbol{M}^{-1}\boldsymbol{A}, \boldsymbol{p}^{(k)}\right) + \mathcal{K}_s\left(\boldsymbol{M}^{-1}\boldsymbol{A}, \boldsymbol{u}^{(k)}\right)$
  $\boldsymbol{G}^{(k)} = {Z^{(k)}}^T Y^{(k)}$
  $\boldsymbol{p}^{(sk)\prime} = \left[1, 0_{1,2s}\right]^T, \boldsymbol{r}^{(sk)\prime} = \left[0_{1,s+1}, 1, 0_{1,s-1}\right]^T, \boldsymbol{x}^{(sk)\prime} = \left[0_{1,2s+1}\right]^T$ (see Alg. 2 in [2])
  for $j = 0, 1, \ldots, s-1$ **do**
    $\alpha^{(sk+j)} = \frac{{\boldsymbol{r}^{(sk+j)\prime}}^T \boldsymbol{G}^{(k)} \boldsymbol{r}^{(sk+j)\prime}}{{\boldsymbol{p}^{(sk+j)\prime}}^T \boldsymbol{G}^{(k)} \boldsymbol{B}\boldsymbol{p}^{(sk+j)\prime}}$
    $\boldsymbol{x}^{(sk+j+1)\prime} = \boldsymbol{x}^{(sk+j)\prime} + \alpha^{(sk+j)} \boldsymbol{p}^{(sk+j)\prime}$
    $\boldsymbol{r}^{(sk+j+1)\prime} = \boldsymbol{r}^{(sk+j)\prime} - \alpha^{(sk+j)} \boldsymbol{B}\boldsymbol{p}^{(sk+j)\prime}$
    $\beta^{(sk+j)} = \frac{{\boldsymbol{r}^{(sk+j+1)\prime}}^T \boldsymbol{G}^{(k)} \boldsymbol{r}^{(sk+j+1)\prime}}{{\boldsymbol{r}^{(sk+j)\prime}}^T \boldsymbol{G}^{(k)} \boldsymbol{r}^{(sk+j)\prime}}$
    $\boldsymbol{p}^{(sk+j+1)\prime} = \boldsymbol{r}^{(sk+j+1)\prime} + \beta^{(sk+j)} \boldsymbol{p}^{(sk+j)\prime}$
  end for
  $\left[\boldsymbol{q}^{(sk+s)}, \boldsymbol{r}^{(sk+s)}\right] = Y^{(k)} \left[\boldsymbol{p}^{(sk+s)\prime}, \boldsymbol{r}^{(sk+s)\prime}\right]$
  $\left[\boldsymbol{p}^{(sk+s)}, \boldsymbol{u}^{(sk+s)}\right] = Z^{(k)} \left[\boldsymbol{p}^{(sk+s)\prime}, \boldsymbol{r}^{(sk+s)\prime}\right]$
  $\boldsymbol{x}^{(sk+s)} = \boldsymbol{x}^{(sk)} + Z^{(k)} \boldsymbol{x}^{(sk+s)\prime}$
end for

is computed using a single global reduction operation, which is used to form the scalars of the s inner iterations without communication.

To express the vectors $\boldsymbol{q}^{(sk+j)}$, $\boldsymbol{p}^{(sk+j)}$, $\boldsymbol{r}^{(sk+j)}$, $\boldsymbol{u}^{(sk+j)}$ and $\boldsymbol{x}^{(sk+j)} - \boldsymbol{x}^{(sk)}$ in the changed basis, we define small vectors $\boldsymbol{p}^{(sk+j)\prime}$, $\boldsymbol{r}^{(sk+j)\prime}, \boldsymbol{x}^{(sk+j)\prime} \in \mathbb{R}^{2s+1}$ for $0 \leq j \leq s$ such that

$$\boldsymbol{q}^{(sk+j)} = Y^{(k)} \boldsymbol{p}^{(sk+j)\prime}, \ \boldsymbol{p}^{(sk+j)} = Z^{(k)} \boldsymbol{p}^{(sk+j)\prime}, \tag{3}$$

$$\boldsymbol{r}^{(sk+j)} = Y^{(k)} \boldsymbol{r}^{(sk+j)\prime}, \ \boldsymbol{u}^{(sk+j)} = Z^{(k)} \boldsymbol{r}^{(sk+j)\prime}, \tag{4}$$

$$\boldsymbol{x}^{(sk+j)} = \boldsymbol{x}^{(sk)} + Z^{(k)} \boldsymbol{x}^{(sk+j)\prime}. \tag{5}$$

In each inner iteration j, these small vectors are updated analogously to the recursive update equations in PCG. At the end of each outer iteration, the vectors in the original basis are regained using (3) to (5). The MV products are computed in the changed basis using a "change-of-basis" matrix $\boldsymbol{B} \in \mathbb{R}^{(2s+1)\times(2s+1)}$ such that

$$\boldsymbol{A}\underline{Z}^{(k)} = Y^{(k)} \boldsymbol{B},$$

with $\underline{Z}^{(k)} = \left[\underline{P}^{(k)}, \underline{U}^{(k)}\right]$, where $\underline{P}^{(k)}$ and $\underline{U}^{(k)}$ are the same as $P^{(k)}$ and $U^{(k)}$ except their respective last column is a zero vector. In the inner iterations, we therefore compute the MV products in the changed basis with $\boldsymbol{B}\boldsymbol{p}^{(k,j)\prime}$ instead of the global MV products of standard PCG. CA-PCG is outlined in Algorithm 3.

2.3 Matrix Powers Kernel (MPK)

The choice of the basis is the main factor that influences stability of communication-avoiding Krylov subspace methods [14]. The Matrix Powers Kernel (MPK) [11] computes the columns of the basis matrices

$$V = \left[P_0(\boldsymbol{A}\boldsymbol{M}^{-1})\boldsymbol{w}, P_1(\boldsymbol{A}\boldsymbol{M}^{-1})\boldsymbol{w}, \ldots, P_s(\boldsymbol{A}\boldsymbol{M}^{-1})\boldsymbol{w}\right], \tag{6}$$

$$\boldsymbol{M}^{-1}V = \left[P_0(\boldsymbol{M}^{-1}\boldsymbol{A})\boldsymbol{v}, P_1(\boldsymbol{M}^{-1}\boldsymbol{A})\boldsymbol{v}, \ldots, P_s(\boldsymbol{M}^{-1}\boldsymbol{A})\boldsymbol{v}\right], \tag{7}$$

where $\boldsymbol{w}$ and $\boldsymbol{v} = \boldsymbol{M}^{-1}\boldsymbol{w}$ are vectors of length n and $P_l(z)$ is a polynomial of degree l [2, 4] that satisfies the three-term recurrence

$$\begin{aligned} &P_0(z) = 1, \quad P_1(z) = (z - \theta_0)P_0(z)/\gamma_0, \\ &P_l(z) = ((z - \theta_{l-1})P_{l-1}(z) + \mu_{l-2}P_{l-2}(z))/\gamma_{l-1}, \quad l \geq 2. \end{aligned} \tag{8}$$

When using the monomial basis, the columns of $\boldsymbol{V}$ consist of the first iterations of the Power Iteration, where a start vector is multiplied with the same matrix multiple times. The iterate of the Power Iteration converges to the eigenvector corresponding to the largest eigenvalue. This can lead to severe accuracy loss in s-step Krylov subspace methods if $s > 5$. Since the residuals get smaller in magnitude throughout the iterations, they get vulnerable to round-off errors. In finite precision, these vectors are not linearly independent anymore, which causes the algorithm to converge slower or not at all [21].

The Newton basis uses shifts obtained from estimating eigenvalues by computing s or $2s$ iterations of standard PCG before starting the actual solver [14, 18]. Another possibility is the Chebyshev basis, which uses scaled and shifted Chebyshev polynomials. A detailed description of different basis types can be found in [14].

While sPCG_{mon} [7] can only use the monomial basis, in CA-PCG the "change-of-basis" matrix $\boldsymbol{B}$ enables the algorithm to improve its numerical stability with other basis types. Define the matrix $\boldsymbol{B}_i$ of size $i \times (i-1)$ as

$$\boldsymbol{B}_i = \begin{bmatrix} \theta_0 & \mu_0 & & \\ \gamma_0 & \theta_1 & \ddots & \\ & \gamma_1 & \ddots & \mu_{i-3} \\ & & \ddots & \theta_{i-2} \\ & & & \gamma_{i-2} \end{bmatrix}. \tag{9}$$

Then the matrix $\boldsymbol{B}$ used in CA-PCG is defined as

$$\boldsymbol{B} = \begin{bmatrix} \boldsymbol{B}_{s+1} & 0_{s+1,1} & 0_{s+1,s-1} & 0_{s+1,1} \\ 0_{s,s} & 0_{s,1} & \boldsymbol{B}_s & 0_{s,1} \end{bmatrix},$$

where $0_{i,j}$ denotes a matrix of size $i \times j$ whose entries are all zeros.

2.4 Communication-avoiding PCG3 (CA-PCG3)

The communication-avoiding algorithm CA-PCG3 [14] is based on the three-term recurrence variant PCG3 [17]. PCG3 does not compute search directions, instead the three-term recurrence relation between residuals

$$\boldsymbol{r}^{(i+1)} = \rho^{(i)}\left(\boldsymbol{r}^{(i)} - \gamma^{(i)}\boldsymbol{A}\boldsymbol{u}^{(i)}\right) + \left(1 - \rho^{(i)}\right)\boldsymbol{r}^{(i-1)}$$

is used to update the residual vectors, with $\gamma^{(i)}$ and $\rho^{(i)}$ being scalars computed in iteration i. The solution vectors are updated using an analogous recurrence relation.

The communication-avoiding CA-PCG3 computes the basis matrices $\boldsymbol{W}^{(k)}$ and $\boldsymbol{V}^{(k)} = \boldsymbol{M}^{-1}\boldsymbol{W}^{(k)}$ at the beginning of outer iteration k such that $\text{span}\left(\boldsymbol{W}^{(k)}\right) = \mathcal{K}_{s+1}\left(\boldsymbol{A}\boldsymbol{M}^{-1}, \boldsymbol{r}^{(sk)}\right)$. The residual

Algorithm 4 Communication-avoiding PCG3 (CA-PCG3) [14]

1: $\boldsymbol{r}^{(-1)} = 0_{n,1}, \boldsymbol{u}^{(-1)} = 0_{n,1}, \boldsymbol{x}^{(-1)} = 0_{n,1}$
2: $\boldsymbol{r}^{(0)} = \boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}^{(0)}, \boldsymbol{u}^{(0)} = \boldsymbol{P}\boldsymbol{r}^{(0)}, \rho^{(0)} = 1$
3: **for** $k = 0, 1, \ldots$ until convergence **do**
4: span $\left(\boldsymbol{W}^{(k)}\right) = \mathcal{K}_{s+1}\left(\boldsymbol{A}\boldsymbol{M}^{-1}, \boldsymbol{r}^{(sk)}\right)$
5: span $\left(\boldsymbol{V}^{(k)}\right) = \mathcal{K}_{s+1}\left(\boldsymbol{M}^{-1}\boldsymbol{A}, \boldsymbol{u}^{(sk)}\right)$
6: $\boldsymbol{G}^{(k)} = \left[\boldsymbol{U}^{(k-1)}, \boldsymbol{V}^{(k)}\right]^T \left[\boldsymbol{R}^{(k-1)}, \boldsymbol{W}^{(k)}\right]$
7: **for** $j = 0, 1, \ldots, s-1$ **do**
8: Compute $\boldsymbol{d}^{(sk+j)}$ and $\boldsymbol{g}^{(sk+j)}$ according to (10) and (11)
9: $\mu^{(sk+j)} = \boldsymbol{g}^{(sk+j)^T}\boldsymbol{G}^{(k)}\boldsymbol{g}^{(sk+j)}$ ▷ $r^{(sk+j)^T}u^{(sk+j)}$
10: $\nu^{(sk+j)} = \boldsymbol{g}^{(sk+j)^T}\boldsymbol{G}^{(k)}\boldsymbol{d}^{(sk+j)}$ ▷ $w^{(sk+j)^T}u^{(sk+j)}$
11: $\gamma^{(sk+j)} = \mu^{(sk+j)} / \nu^{(sk+j)}$
12: $\boldsymbol{w}^{(sk+j)} = \left[\boldsymbol{R}^{(k-1)}, \boldsymbol{W}^{(k)}\right]\boldsymbol{d}^{(sk+j)}$ ▷ $\boldsymbol{A}\boldsymbol{u}^{(sk+j)}$
13: $\boldsymbol{v}^{(sk+j)} = \left[\boldsymbol{U}^{(k-1)}, \boldsymbol{V}^{(k)}\right]\boldsymbol{d}^{(sk+j)}$ ▷ $\boldsymbol{P}\boldsymbol{A}\boldsymbol{u}^{(sk+j)}$
14: **if** $sk + j > 0$ **then**
15: $\rho^{(sk+j)} = \left(1 - \frac{\gamma^{(sk+j)}}{\gamma^{(sk+j-1)}} \frac{\mu^{(sk+j)}}{\mu^{(sk+j-1)}} \frac{1}{\rho^{(sk+j-1)}}\right)^{-1}$
16: **end if**
17: $\boldsymbol{x}^{(sk+j+1)} = \rho^{(sk+j)}\left(\boldsymbol{x}^{(sk+j)} + \gamma^{(sk+j)}\boldsymbol{u}^{(sk+j)}\right) + \left(1 - \rho^{(sk+j)}\right)\boldsymbol{x}^{(sk+j-1)}$
18: $\boldsymbol{r}^{(sk+j+1)} = \rho^{(sk+j)}\left(\boldsymbol{r}^{(sk+j)} - \gamma^{(sk+j)}\boldsymbol{w}^{(sk+j)}\right) + \left(1 - \rho^{(sk+j)}\right)\boldsymbol{r}^{(sk+j-1)}$
19: $\boldsymbol{u}^{(sk+j+1)} = \rho^{(sk+j)}\left(\boldsymbol{u}^{(sk+j)} - \gamma^{(sk+j)}\boldsymbol{v}^{(sk+j)}\right) + \left(1 - \rho^{(sk+j)}\right)\boldsymbol{u}^{(sk+j-1)}$
20: **end for**
21: **end for**

matrices $\boldsymbol{R}^{(k)} = \left[\boldsymbol{r}^{(sk)}, \ldots, \boldsymbol{r}^{(sk+s-1)}\right]$ and $\boldsymbol{U}^{(k)} = \boldsymbol{M}^{-1}\boldsymbol{R}^{(k)}$ store the residual vectors computed in outer iteration k.

In each inner iteration j, the vectors $\boldsymbol{w}^{(sk+j)} = \boldsymbol{A}\boldsymbol{u}^{(sk+j)}$ and $\boldsymbol{v}^{(sk+j)} = \boldsymbol{M}^{-1}\boldsymbol{A}\boldsymbol{u}^{(sk+j)}$ are formed without explicitly computing the MV products and preconditioner applications using auxiliary vectors $\boldsymbol{d}^{(sk+j)} \in \mathbb{R}^{2s+1}$ that fulfill the relation

$$\boldsymbol{A}\boldsymbol{u}^{(sk+j)} = \left[\boldsymbol{R}^{(k-1)}, \boldsymbol{W}^{(k)}\right]\boldsymbol{d}^{(sk+j)}. \tag{10}$$

Moreover, auxiliary vectors $\boldsymbol{g}^{(sk+j)} \in \mathbb{R}^{2s+1}$ are formed such that

$$\boldsymbol{r}^{(sk+j)} = \left[\boldsymbol{R}^{(k-1)}, \boldsymbol{W}^{(k)}\right]\boldsymbol{g}^{(sk+j)}. \tag{11}$$

Forming these auxiliary vectors requires a "change-of-basis" matrix as defined in (9). See [14] for a detailed description on how these auxiliary vectors are computed.

The Gram matrix $\boldsymbol{G}^{(k)} = \left[\boldsymbol{R}^{(k-1)}, \boldsymbol{W}^{(k)}\right]^T \left[\boldsymbol{U}^{(k-1)}, \boldsymbol{V}^{(k)}\right] \in \mathbb{R}^{(2s+1)\times(2s+1)}$ is computed at the beginning of outer iteration k to compute the scalars of the s inner iterations without communication. CA-PCG3 is outlined in Algorithm 4.

3 sPCG with arbitrary basis types

The basis type chosen for the s-step basis can significantly influence the stability of an s-step algorithm [14]. In this section, we extend sPCG$_{\text{mon}}$ for arbitrary basis types to improve its numerical stability.

In our extended version sPCG, the basis matrices $\boldsymbol{S}^{(k)}$ and $\boldsymbol{U}^{(k)}$ are computed with the MPK as shown in (6) and (7) (the last column of (7) is ommitted as $\boldsymbol{U}^{(k)}$ only has s columns). Consequently, the matrix $\boldsymbol{A}\boldsymbol{U}^{(k)}$ does not simply consist of the last s columns of $\boldsymbol{S}^{(k)}$ when not using the monomial basis. Instead, we introduce a "change-of-basis" matrix similar to the one used in CA-PCG. Define a matrix $\boldsymbol{B} = \boldsymbol{B}_{s+1}$ (see (9)) of size $(s+1) \times s$ such that

$$\boldsymbol{A}\boldsymbol{U}^{(k)} = \boldsymbol{S}^{(k)}\boldsymbol{B}.$$

The rest of the algorithm remains unchanged except for the routine "Scalar Work" for computing the global reductions and $\boldsymbol{a}^{(k)}$ and $\boldsymbol{B}^{(k)}$ (see line 8 of Algorithm 2). In the following, we explain this routine in its original form for sPCG$_{\text{mon}}$ as well as our novel extension for arbitrary basis types.

3.1 "Scalar Work" for the monomial basis

Since search directions are $\boldsymbol{A}$-orthogonal, $\boldsymbol{P}^{(k)^T}\boldsymbol{A}\boldsymbol{P}^{(k-1)} = 0$. To compute $\boldsymbol{B}^{(k)}$, we multiply (1) by $\boldsymbol{P}^{(k-1)^T}\boldsymbol{A}$ and get

$$\left(\boldsymbol{P}^{(k-1)^T}\boldsymbol{A}\boldsymbol{P}^{(k-1)}\right)\boldsymbol{B}^{(k)} = -\boldsymbol{P}^{(k-1)^T}\boldsymbol{A}\boldsymbol{U}^{(k)}.$$

To obtain a relation for computing $\boldsymbol{a}^{(k)}$, the recursive relation of the residual in (2) is multipled by $\boldsymbol{U}^{(k)^T}$.

$$\left(\boldsymbol{P}^{(k)^T}\boldsymbol{A}\boldsymbol{U}^{(k)}\right)\boldsymbol{a}^{(k)} = \boldsymbol{R}^{(k)^T}\boldsymbol{u}^{(k)} \tag{12}$$

These small linear systems with problem size s are solved locally on each node for $\boldsymbol{B}^{(k)}$ respectively $\boldsymbol{a}^{(k)}$.

Using the recurrence relation between search direction matrices in (1) and their $\boldsymbol{A}$-orthogonality,

$$\boldsymbol{W}^{(k)} = \boldsymbol{P}^{(k)^T}\boldsymbol{A}\boldsymbol{P}^{(k)} = \boldsymbol{P}^{(k)^T}\boldsymbol{A}\boldsymbol{U}^{(k)}.$$

"Scalar Work" requires only one global collective for computing $\boldsymbol{B}^{(k)}$ and $\boldsymbol{a}^{(k)}$. Using (1), $\boldsymbol{W}^{(k)}$ can be computed as

$$\boldsymbol{W}^{(k)} = \boldsymbol{P}^{(k)^T}\boldsymbol{A}\boldsymbol{U}^{(k)} = \boldsymbol{U}^{(k)^T}\boldsymbol{A}\boldsymbol{U}^{(k)} + \boldsymbol{P}^{(k-1)^T}\boldsymbol{A}\boldsymbol{U}^{(k)}\boldsymbol{B}^{(k)},$$

whose communication can be executed without having $\boldsymbol{P}^{(k)}$ available and thus before computing $\boldsymbol{B}^{(k)}$.

The algorithm in [7] computes a vector of moments $\mu^{(k)} =$

$$\left[\boldsymbol{r}^{(k)^T}\boldsymbol{u}^{(k)}, \boldsymbol{r}^{(k)^T}\boldsymbol{M}^{-1}\boldsymbol{A}\boldsymbol{u}^{(k)}, \ldots, \boldsymbol{r}^{(k)^T}(\boldsymbol{M}^{-1}\boldsymbol{A})^{2s-1}\boldsymbol{u}^{(k)}\right]. \tag{13}$$

With these values, the matrix of moments $\boldsymbol{U}^{(k)^T}\boldsymbol{A}\boldsymbol{U}^{(k)}$ as well as the right-hand side $\boldsymbol{R}^{(k)^T}\boldsymbol{u}^{(k)}$ of (12) are formed. The entries of the matrix $\boldsymbol{P}^{(k-1)^T}\boldsymbol{A}\boldsymbol{U}^{(k)}$ are computed using a recurrence formula involving the moments $\mu^{(k)}$ and $\boldsymbol{a}^{(k-1)}$ of the previous iteration and thus do not require additional global communication.

3.2 "Scalar Work" for arbitrary basis types

When using arbitrary basis types, $P_i(\boldsymbol{A}\boldsymbol{M}^{-1})P_j(\boldsymbol{A}\boldsymbol{M}^{-1})$ for $0 \leq i, j \leq s$ is not necessarily the same as $P_{i+j}(\boldsymbol{A}\boldsymbol{M}^{-1})$. Thus, a vector of

Algorithm 5 s-step PCG for arbitrary basis types (sPCG)

1: $\boldsymbol{r}^{(0)} = \boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}^{(0)}$
2: $\boldsymbol{P}^{(0)} = 0, \boldsymbol{A}\boldsymbol{P}^{(0)} = 0$
3: **for** $k = 0, 1, \ldots$ until convergence **do**
4: $\quad \boldsymbol{u}^{(k)} = \boldsymbol{M}^{-1}\boldsymbol{r}^{(k)}$
5: $\quad \boldsymbol{S}^{(k)} = \text{span}\{\boldsymbol{r}^{(k)}, (\boldsymbol{A}\boldsymbol{M}^{-1})\boldsymbol{r}^{(k)}, \ldots, (\boldsymbol{A}\boldsymbol{M}^{-1})^{s}\boldsymbol{r}^{(k)}\}$
6: $\quad \boldsymbol{U}^{(k)} = \text{span}\{\boldsymbol{u}^{(k)}, (\boldsymbol{M}^{-1}\boldsymbol{A})\boldsymbol{u}^{(k)}, \ldots, (\boldsymbol{M}^{-1}\boldsymbol{A})^{s-1}\boldsymbol{u}^{(k)}\}$
7: $\quad \boldsymbol{R}^{(k)} = \text{first } s \text{ columns of } \boldsymbol{S}^{(k)}$
8: $\quad \boldsymbol{A}\boldsymbol{U}^{(k)} = \boldsymbol{S}^{(k)}\boldsymbol{B}$
9: $\quad \boldsymbol{a}^{(k)}, \boldsymbol{B}^{(k)} \leftarrow \text{Scalar Work } (\boldsymbol{S}^{(k)}, \boldsymbol{U}^{(k)}, \boldsymbol{P}^{(k-1)}, \boldsymbol{B})$
10: $\quad \boldsymbol{P}^{(k)} = \boldsymbol{U}^{(k)} + \boldsymbol{P}^{(k-1)}\boldsymbol{B}^{(k)}$
11: $\quad \boldsymbol{A}\boldsymbol{P}^{(k)} = \boldsymbol{S}^{(k)}\boldsymbol{B} + \boldsymbol{A}\boldsymbol{P}^{(k-1)}\boldsymbol{B}^{(k)}$
12: $\quad \boldsymbol{x}^{(k+1)} = \boldsymbol{x}^{(k)} + \boldsymbol{P}^{(k)}\boldsymbol{a}^{(k)}$
13: $\quad \boldsymbol{r}^{(k+1)} = \boldsymbol{r}^{(k)} - \boldsymbol{A}\boldsymbol{P}^{(k)}\boldsymbol{a}^{(k)}$
14: **end for**

Algorithm 6 Scalar Work of sPCG for arbitrary basis types

1: $\boldsymbol{m}^{(k)} = \boldsymbol{R}^{(k)^T}\boldsymbol{u}^{(k)}$ ▷ First column of $\boldsymbol{U}^{(k)^T}\boldsymbol{S}^{(k)}$ (see line 6)
2: **if** $k \neq 0$ **then**
3: $\quad \boldsymbol{C}^{(k)} = -\boldsymbol{B}^T\boldsymbol{S}^{(k)^T}\boldsymbol{P}^{(k-1)}$ ▷ $-\boldsymbol{U}^{(k)^T}\boldsymbol{A}\boldsymbol{P}^{(k-1)}$
4: $\quad$ Solve $\boldsymbol{W}^{(k-1)}\boldsymbol{B}^{(k)} = \boldsymbol{C}^{(k)}$ for $\boldsymbol{B}^{(k)}$
5: **end if**
6: $\boldsymbol{W}^{(k)} = \boldsymbol{U}^{(k)^T}\boldsymbol{S}^{(k)}\boldsymbol{B} - \boldsymbol{C}^{(k)}\boldsymbol{B}^{(k)}$ ▷ $\boldsymbol{U}^{(k)^T}\boldsymbol{A}\boldsymbol{U}^{(k)} - \boldsymbol{C}^{(k)}\boldsymbol{B}^{(k)}$
7: Solve $\boldsymbol{W}^{(k)}\boldsymbol{a}^{(k)} = \boldsymbol{m}^{(k)}$ for $\boldsymbol{a}^{(k)}$

moments analogously to $\mu^{(k)}$ in (13) is not sufficient for computing $\boldsymbol{P}^{(k-1)^T}\boldsymbol{A}\boldsymbol{U}^{(k)}$, $\boldsymbol{R}^{(k)^T}\boldsymbol{u}^{(k)}$ and $\boldsymbol{U}^{(k)^T}\boldsymbol{A}\boldsymbol{U}^{(k)}$.

For enabling arbitrary basis types, we replace computing the moments $\mu^{(k)}$ with computing $\boldsymbol{U}^{(k)^T}\boldsymbol{S}^{(k)}$. The first column of this matrix is $\boldsymbol{R}^{(k)^T}\boldsymbol{u}^{(k)}$, which we need for (12). As $\boldsymbol{U}^{(k)^T}\boldsymbol{S}^{(k)}\boldsymbol{B} = \boldsymbol{U}^{(k)^T}\boldsymbol{A}\boldsymbol{U}^{(k)}$, we can inexpensively compute this matrix of moments from $\boldsymbol{U}^{(k)^T}\boldsymbol{S}^{(k)}$ locally on each node.

Similarly, the matrix $\boldsymbol{P}^{(k-1)^T}\boldsymbol{A}\boldsymbol{U}^{(k)} = \boldsymbol{P}^{(k-1)^T}\boldsymbol{S}^{(k)}\boldsymbol{B}$ can inexpensively be obtained by computing $\boldsymbol{P}^{(k-1)^T}\boldsymbol{S}^{(k)}$ and subsequently applying $\boldsymbol{B}$. The communication for $\boldsymbol{U}^{(k)^T}\boldsymbol{S}^{(k)}$ and $\boldsymbol{P}^{(k-1)^T}\boldsymbol{S}^{(k)}$ can be combined in one global reduction operation.

We show sPCG for abitrary basis types in Algorithm 5 and its corresponding "Scalar Work" routine in Algorithm 6. The modifications required for using arbitrary basis types are indicated in red. Note that sPCG with the monomial basis is not the same as sPCG_{mon} in finite precision. sPCG computes the matrices $\boldsymbol{U}^{(k)^T}\boldsymbol{A}\boldsymbol{U}^{(k)}$ and $\boldsymbol{C}^{(k)}$ directly, while sPCG_{mon} forms $\boldsymbol{U}^{(k)^T}\boldsymbol{A}\boldsymbol{U}^{(k)}$ with the vector of moments $\mu^{(k)}$ and computes the values in $\boldsymbol{C}^{(k)}$ recursively. Although mathematically equivalent, our direct computations in sPCG tend to be slightly more numerically stable than sPCG_{mon}.

4 Theoretical analysis

In this section, we theoretically analyze the performance of the discussed s-step PCG solvers. All three s-step solvers only require one global reduction operation per s steps, i.e., per (outer) iteration. Therefore, they reduce the number of global collectives by a factor of $2s$ compared to standard PCG. This is done at the expense of additional computation. We argue that our new version sPCG is beneficial in this regard compared to the other two s-step methods. The computational cost for each algorithm is listed in Table 1.

4.1 Computation of vectors

While CA-PCG computes $2s - 1$ MV products and preconditioner applications per s steps, standard PCG, sPCG and CA-PCG3 only require s of these operations per s steps.

In sPCG, updating the search direction matrix $\boldsymbol{P}^{(k)}$ as well as the matrix $\boldsymbol{A}\boldsymbol{P}^{(k)}$ requires $O(s^2n)$ floating-point operations (FLOPs) as $\boldsymbol{P}^{(k-1)}$ and $\boldsymbol{A}\boldsymbol{P}^{(k-1)}$ are multiplied with the $s \times s$ matrix $\boldsymbol{B}^{(k)}$ (lines 10 and 11 of Algorithm 5). Contrarily, the cost of recovering the full vectors from their small counterparts in the changed basis at the end of an outer CA-PCG iteration is only $O(sn)$ (lines 14, 15 and 16 of Algorithm 3). CA-PCG3 forms the results of the MV product and preconditioner application in each inner iteration recursively without communication using the s-step basis matrices and the residuals of the previous s inner iterations, which results in an $O(s^2n)$ cost for s steps (lines 12 and 13 of Algorithm 4). Moreover, the residuals and solution vectors in CA-PCG3 are updated utilizing BLAS1 (lines 17, 18 and 19 of Algorithm 4).

4.2 Cost for using arbitrary basis types

Using arbitrary basis types introduces additional local vector operations during the MPK. Depending on the basis type, applying the parameters resulting from the three-term recurrence relation in (8) introduces at most $3n$ additional FLOPs for the first MV product of one MPK execution (as there is no superdiagonal value in the first column of (9)), and at most $5n$ FLOPs per subsequent MV product.

Otherwise, the usage of arbitrary basis types introduces only negligible computations with vectors and matrices of dimensions $O(s)$ respectively $O(s) \times O(s)$ in CA-PCG and CA-PCG3. sPCG must additionally compute $\boldsymbol{A}\boldsymbol{U}^{(k)} = \boldsymbol{S}^{(k)}\boldsymbol{B}$. Since $\boldsymbol{B}$ is tridiagonal, its application involves at most $(5s-2)n$ FLOPs (as the first column of (9) has no superdiagonal value). As sPCG_{mon} only computes $2s$ local reductions for the vector of moments, it has slightly less computational cost than sPCG for the monomial basis, which computes two matrices of size $(s+1) \times s$.

4.3 Summary

In terms of local vector operations, CA-PCG is the least expensive solver for $s \geq 10$. However, it computes more MV products and preconditioner applications than sPCG and CA-PCG3. Our novel method sPCG is less expensive than CA-PCG3 in terms of local vector operations for all s and can fully utilize BLAS2/BLAS3 (unlike CA-PCG3). Our extension for arbitrary basis types introduces only negligible overhead and makes sPCG an efficient algorithm suitable for situations where it is important to reduce global synchronization points.

5 Experimental evaluation

We experimentally evaluate the runtime performance of all considered s-step methods and investigate their numerical stability.

Table 1: Computational cost per s steps for each algorithm. Only operations involving vectors and matrix columns of length n are considered. Second column: Number of MV products and preconditioner applications. Remaining columns: cost for different types of local computations per system matrix row (i.e., number of floating point operations (FLOPs) divided by n).

Algorithm	#MV + #prec. appl.	Remaining #FLOPs/n (beyond MV and prec. appl.)				
		Local reductions	Vector/Matrix column computations		Total remaining #FLOPs/n	
			for monomial b.	Additional for arb. b.	Monomial basis	Arbitrary basis
PCG	s	$2s$	$6s$	-	$8s$	-
sPCG_{mon}	s	$2s$	$4s^2+4s$	-	$4s^2+6s$	-
sPCG	s	$2s(s+1)$	$4s^2+4s$	$10s-4$	$6s^2+6s$	$6s^2+16s-4$
CA-PCG	$2s-1$	$(2s+1)^2$	$20s+6$	$10s-9$	$4s^2+24s+7$	$4s^2+34s-2$
CA-PCG3	s	$(2s+1)^2$	$8s^2+17s$	$5s-2$	$12s^2+21s+1$	$12s^2+26s-1$

5.1 Implementation and experimental setup

The algorithms were implemented in C++ using Trilinos 16.0.0 [22] based on Trilinos' own PCG implementation. We used OpenMPI 4.1.6 and the GCC compiler 12.2.0 with compiler flag `-O3`. The computational results presented have been achieved using the Austrian Scientific Computing (ASC) infrastructure. Matrices of size $n \times n$ are block-row distributed and vectors of length n are distributed accordingly. We chose the right-hand side $\boldsymbol{b}$ so that each entry of the solution $\boldsymbol{x}$ has the value $1/\sqrt{n}$. The start vector $\boldsymbol{x}^{(0)}$ is a zero vector.

We used 128 processes per node. For each execution variant (different values of s, different numbers of nodes), 10 test runs were executed. We show the median runtimes over these 10 test runs.

We ran experiments with PCG, sPCG, CA-PCG, and CA-PCG3 using a Jacobi or Chebyshev preconditioner. Both preconditioners require little or no communication and are thus suitable for s-step methods. Estimates for the largest and smallest eigenvalues necessary for the Chebyshev basis type and the Chebyshev preconditioner were computed with a few iterations of standard PCG (not included in the runtimes).

5.2 Numerical stability

A basis type different from the monomial basis is crucial for convergence [14]. In Table 2, we show results for test matrices from the SuiteSparse Matrix collection [10] using a Chebyshev preconditioner with degree 3 and $s = 10$, executed on one node (128 processes). The algorithms were terminated once the 2-norm of the true relative residual $(\boldsymbol{b}-\boldsymbol{A}\boldsymbol{x}^{(k)})/\|\boldsymbol{b}-\boldsymbol{A}\boldsymbol{x}^{(0)}\|_2$ was below 10^{-9}. We used all SPD test matrices from the SuiteSparse Matrix collection with a problem size between 100000 and 2000000 that converged within 10000 iterations of standard PCG. We list the number of iterations required to achieve the desired accuracy for each solver considered. The s-step methods evaluate the convergence criterion only every s steps. We consider less than 20% iteration overhead or less than 10 extra iterations compared to standard PCG as not significant. If convergence was not achieved within 12000 iterations, we considered the instance not to have converged.

CA-PCG converged for 23 out of 40 matrices with the monomial basis. However, only six of these matrices did not have significant convergence delay compared to standard PCG. sPCG and CA-PCG3 with the monomial basis converged for only one and two matrices, respectively. This underlines the necessity for using arbitrary basis types in s-step methods.

When using the Chebyshev basis, CA-PCG converged for 35 of the 40 test matrices, with only two of them having significant convergence delay compared to standard PCG. Thus, ~ 80% of the matrices had a similar convergence behavior as for standard PCG. sPCG and CA-PCG3 converged for 19 and 21 matrices, respectively (~ 50%), all of them without significant convergence delay.

Our results indicate that CA-PCG is more stable than sPCG and CA-PCG3. However, CA-PCG is significantly more expensive than the other two solvers due to additional MV products and preconditioner applications. Although sPCG is doing slightly worse than CA-PCG3 in Table 2, it clearly outperforms CA-PCG and CA-PCG3 in terms of runtime performance.

5.3 Runtime performance

Columns 2-5 of Table 3 show the performance results of the seven largest matrices in Table 2 where at least two s-step methods converged using $s = 10$, the Chebyshev basis and a Chebyshev preconditioner of degree 3. The algorithms were executed on four nodes (512 processes), the convergence criterion was the reduction of the 2-norm of the recursively computed residual by a factor of 10^9. This is less expensive than computing the true residual as in Table 2. Columns 6-9 show results for the same matrices with the Chebyshev basis, $s = 10$, but with a Jacobi preconditioner. The algorithms were terminated once the $\boldsymbol{M}$-norm $\sqrt{\boldsymbol{r}^{(i)^T}\boldsymbol{M}^{-1}\boldsymbol{r}^{(i)}}$ of the recursively computed residual had been reduced by a factor of 10^9. This convergence criterion can be assessed inexpensively as all considered solvers compute the term under the square root.

We see that sPCG achieves the best speedup in all cases. On the contrary, CA-PCG3 could not achieve speedup over standard PCG for two matrices using the Jacobi preconditioner. CA-PCG did not achieve a speedup for any matrix for both preconditioners. Note that the main performance drawback of CA-PCG results from the additional MV products and preconditioner applications. The results in Table 3 show that CA-PCG cannot even with the very cheap Jacobi preconditioner reliably achieve the same performance as standard PCG.

Fig. 1 shows strong scaling results with a 3D-Poisson matrix of size $256 \times 256 \times 256$ resulting from discretizing Poisson's equation with a 7-point stencil. We used $s = 5, 10, 15$, a Jacobi preconditioner and the Chebyshev basis. We show the speedup of all solver variants

Table 2: All test matrices from the SuiteSparse Matrix collection of size between 100000 and 2000000, where standard PCG converged within 10000 iterations. Chebyshev preconditioner of degree 3, $s = 10$, one node (128 processes), monomial (left) and Chebyshev (right) basis. Convergence criterion: $\|(b - Ax^{(k)})\|_2 / \|b - Ax^{(0)}\|_2 < 10^{-9}$. Hyphen: the algorithm diverged, the residual stagnated before reaching the desired accuracy, or convergence was not achieved within 12000 iterations. "M" = 10^6. Instances with < 20% iteration overhead or < s extra iterations compared to standard PCG are in bold.

Matrix	Size	NNZ	PCG	sPCG	CA-PCG	CA-PCG3
2cubes_s.	0.1M	1.6M	22	-/**30**	**30**/**30**	**30**/**30**
therm._TC	0.1M	0.7M	11	30/**20**	30/**20**	-/**20**
shipsec8	0.1M	3.3M	1666	-/-	2150/**1960**	-/-
ship_003	0.1M	3.8M	1584	-/**1590**	4590/**1590**	-/**1590**
cfd2	0.1M	3.1M	1731	-/**1750**	**1770**/**1750**	-/**1750**
boneS01	0.1M	5.5M	787	-/**790**	1750/**790**	-/**790**
shipsec1	0.1M	3.6M	909	-/**910**	**910**/**910**	-/**910**
bmw7st_1	0.1M	7.3M	7243	-/-	-/**7260**	-/**7280**
Dubcova3	0.1M	3.6M	73	-/**80**	130/**80**	170/**80**
bmwcra_1	0.1M	11M	2183	-/-	-/7890	-/-
G2_circuit	0.2M	0.7M	506	-/**510**	-/**510**	-/**510**
shipsec5	0.2M	4.6M	751	-/**760**	**750**/**760**	-/**760**
therm._dM	0.2M	1.4M	11	-/**20**	250/**20**	-/**20**
pwtk	0.2M	12M	7377	-/-	-/-	-/-
hood	0.2M	9.9M	1515	-/**1520**	1840/**1520**	-/**1520**
offshore	0.3M	4.2M	178	-/**180**	**210**/**180**	-/**180**
af_0_k101	0.5M	18M	8891	-/-	11190/**8960**	-/**8960**
af_1_k101	0.5M	18M	8359	-/-	-/**8360**	-/**8360**
af_2_k101	0.5M	18M	9956	-/-	-/**10000**	-/-
af_3_k101	0.5M	18M	8076	-/-	-/**8110**	-/-
af_4_k101	0.5M	18M	9881	-/-	**11390**/**9890**	-/**9890**
af_5_k101	0.5M	18M	9467	-/-	-/**9470**	-/**9470**
af_shell3	0.5M	18M	993	-/-	1440/**1000**	-/-
af_shell4	0.5M	18M	993	-/-	1440/**1000**	-/-
af_shell7	0.5M	18M	991	-/-	1650/**1000**	-/-
af_shell8	0.5M	18M	991	-/-	1650/**1000**	-/-
parabolic.	0.5M	18M	540	-/**540**	660/**540**	-/-
Fault_639	0.6M	27M	5414	-/-	-/-	-/-
apache2	0.7M	4.8M	1554	-/**1560**	-/**1560**	-/-
Emilia_923	0.9M	40M	4564	-/-	-/**5200**	-/-
audikw_1	0.9M	78M	2520	-/**2520**	4040/**2520**	-/**2520**
ldoor	1.0M	42M	2764	-/**2770**	-/**2770**	-/**2770**
bone010	1.0M	48M	4308	-/-	-/-	-/-
ecology2	1.0M	5.0M	2345	-/**2350**	-/**2350**	-/-
thermal2	1.2M	8.6M	1674	-/-	7960/**1680**	-/-
Serena	1.4M	64M	570	-/-	-/-	-/-
Geo_1438	1.4M	60M	545	-/**550**	790/**550**	-/**550**
Hook_1498	1.5M	59M	1817	-/-	7410/2610	-/-
Flan_1565	1.6M	114M	4469	-/-	-/-	-/-
G3_circuit	1.6M	7.7M	628	-/**630**	-/**630**	-/**630**

with different numbers of nodes over standard PCG executed on one node, which required 9.34126 seconds until convergence. As

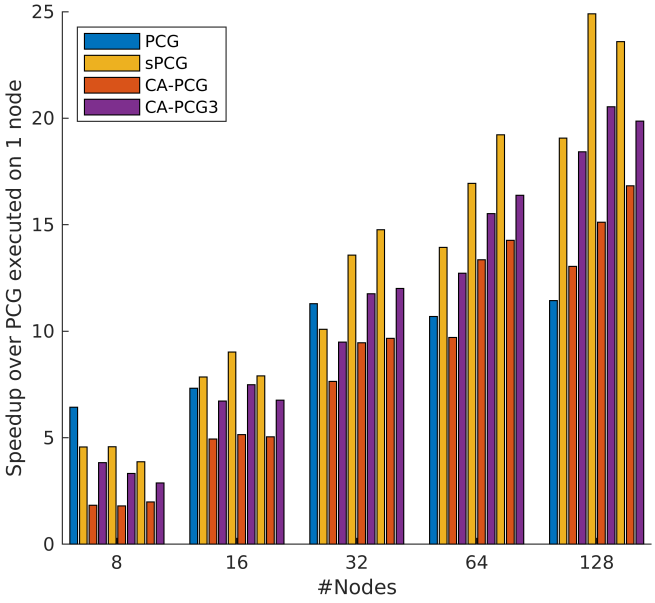

Figure 1: Speedup for a 7-point 3D Poisson matrix of size $256 \times 256 \times 256$ for different numbers of nodes and different values of s over standard PCG executed on one node (128 processes). Chebyshev basis and Jacobi preconditioner. For each bar group: bar 1: standard PCG, bars 2-4: s-step methods with $s = 5$, bars 5-7: $s = 10$, bars 8-10: $s = 15$.

convergence criterion, the algorithms were terminated once the M-norm $\sqrt{r^{(i)^T} M^{-1} r^{(i)}}$ of the recursively computed residual had been reduced by a factor of 10^9.

Standard PCG does not scale beyond 32 nodes. While all s-step methods continue to scale for higher numbers of nodes, sPCG performs best and CA-PCG worst. sPCG achieves a better speedup than standard PCG already with 16 nodes, CA-PCG and CA-PCG3 significantly improve over standard PCG only with 64 and 128 nodes. This is consistent with our theoretical analysis as CA-PCG must perform more MV products and preconditioner applications, while CA-PCG3 cannot block the local operations like sPCG, which can utilize BLAS2/3 instead of BLAS1/2. While performing best in our numerical stability experiments, CA-PCG is clearly the worst out of the three methods in terms of runtime performance, even though the test matrix is very sparse and we used a very cheap Jacobi preconditioner.

6 Conclusion and future work

We extended the s-step PCG method sPCG_{mon} presented in [7] to general basis types beyond the monomial basis. This new algorithm, denoted as sPCG, has better numerical stability than sPCG_{mon}. We showed theoretically that sPCG requires fewer local computations than the other two existing s-step methods [14, 21], which had already been formulated for arbitrary basis types, while retaining the advantage of reducing global communication bottlenecks. In strong scaling experiments with a synthetic test problem as well as in runtime experiments with real-world problems, our novel

Table 3: Performance results (runtimes for standard PCG and speedups of the s-step methods over standard PCG) for the seven largest matrices in Table 2, for which at least two s-step methods converged for the Chebyshev basis. $s = 10$, four nodes (512 processes). Best speedups are in bold. Convergence criterion: 2-norm (columns 2-5) respectively M-norm (columns 6-9) of the recursively computed residual has been reduced by a factor of 10^9. Hyphen: see Table 2.

Matrix	Chebyshev preconditioner (degree 3)				Jacobi preconditioner			
	PCG	sPCG	CA-PCG	CA-PCG3	PCG	sPCG	CA-PCG	CA-PCG3
parabolic_fem	0.790s	**1.41**	0.80	-	0.902s	**1.55**	0.84	1.19
apache2	1.487s	**1.53**	0.85	-	1.333s	**1.05**	0.82	0.87
audikw_1	10.632s	**1.31**	0.79	1.27	2.200s	-	0.77	-
ldoor	11.165s	**1.06**	0.86	1.04	2.570s	-	0.81	-
ecology2	1.465s	**1.35**	0.81	-	1.605s	**1.05**	0.56	0.69
Geo_1438	1.304s	**1.26**	0.71	1.08	0.466s	**1.06**	0.71	1.02
G3_circuit	0.921s	**1.11**	0.66	1.01	0.916s	**1.63**	0.95	1.27

algorithm showed clear performance advantages over the other two existing s-step PCG methods.

References

[1] Erin Carson. 2015. *Communication-Avoiding Krylov Subspace Methods in Theory and Practice.* Ph. D. Dissertation. EECS Department, University of California, Berkeley.

[2] Erin Carson. 2018. The Adaptive s-Step Conjugate Gradient Method. *SIAM J. Matrix Anal. Appl.* 39, 3 (2018), 1318–1338. doi:10.1137/16M1107942

[3] Erin Carson and James Demmel. 2014. A Residual Replacement Strategy for Improving the Maximum Attainable Accuracy of s-Step Krylov Subspace Methods. *SIAM J. Matrix Anal. Appl.* 35, 1 (2014), 22–43. doi:10.1137/120893057

[4] Erin Carson, Nicholas Knight, and James Demmel. 2014. An efficient deflation technique for the communication-avoiding conjugate gradient method. *Electronic transactions on numerical analysis ETNA* 43 (2014), 125–141.

[5] Erin C. Carson, Tomás Gergelits, and Ichitaro Yamazaki. 2022. Mixed precision s-step Lanczos and conjugate gradient algorithms. *Numerical Linear Algebra with Applications* 29, 3 (2022), e2425.

[6] Tyler Chen and Erin C. Carson. 2020. Predict-and-recompute conjugate gradient variants. *SIAM Journal on Scientific Computing* 42, 5 (2020), A3084–A3108. arXiv:1905.01549 doi:10.1137/19m1276856

[7] Anthony T. Chronopoulos and C. William Gear. 1989. On the efficient implementation of preconditioned s-step conjugate gradient methods on multiprocessors with memory hierarchy. *Parallel Comput.* 11, 1 (1989), 37–53. doi:10.1016/0167-8191(89)90062-8

[8] Anthony T. Chronopoulos and C. William Gear. 1989. s-step iterative methods for symmetric linear systems. *J. Comput. Appl. Math.* 25, 2 (1989), 153–168. doi:10.1016/0377-0427(89)90045-9

[9] Siegfried Cools, Jeffrey Cornelis, and Wim Vanroose. 2019. Numerically Stable Recurrence Relations for the Communication Hiding Pipelined Conjugate Gradient Method. *IEEE Transactions on Parallel and Distributed Systems* 30, 11 (2019), 2507–2522. doi:10.1109/TPDS.2019.2917663

[10] Timothy A. Davis and Yifan Hu. 2011. The University of Florida Sparse Matrix Collection. *ACM Trans. Math. Software* 38, 1, Article 1 (2011), 25 pages. doi:10.1145/2049662.2049663

[11] James Demmel, Mark F. Hoemmen, Marghoob Mohiyuddin, and Katherine A. Yelick. 2007. *Avoiding Communication in Computing Krylov Subspaces.* Technical Report UCB/EECS-2007-123. EECS Department, University of California, Berkeley.

[12] Pieter Ghysels and Wim Vanroose. 2014. Hiding global synchronization latency in the preconditioned Conjugate Gradient algorithm. *Parallel Comput.* 40, 7 (2014), 224–238. doi:10.1016/j.parco.2013.06.001 7th Workshop on Parallel Matrix Algorithms and Applications.

[13] Martin Gutknecht and Zdenvek Strakos. 2000. Accuracy of Two Three-term and Three Two-term Recurrences for Krylov Space Solvers. *SIAM J. Matrix Anal. Appl.* 22, 1 (2000), 213–229. doi:10.1137/S0895479897331862

[14] Mark F. Hoemmen. 2010. *Communication-avoiding Krylov subspace methods.* Ph. D. Dissertation. EECS Department, University of California, Berkeley.

[15] Yasuhiro Idomura, Takuya Ina, Susumu Yamashita, Naoyuki Onodera, Susumu Yamada, and Toshiyuki Imamura. 2018. Communication Avoiding Multigrid Preconditioned Conjugate Gradient Method for Extreme Scale Multiphase CFD Simulations. In *2018 IEEE/ACM 9th Workshop on Latest Advances in Scalable Algorithms for Large-Scale Systems (scalA)* (Dallas, Texas, USA). IEEE, Piscataway, NJ, USA, 17–24. doi:10.1109/ScalA.2018.00006

[16] Akie Mayumi, Yasuhiro Idomura, Takuya Ina, Susumu Yamada, and Toshiyuki Imamura. 2016. Left-Preconditioned Communication-Avoiding Conjugate Gradient Methods for Multiphase CFD Simulations on the K Computer. In *2016 7th Workshop on Latest Advances in Scalable Algorithms for Large-Scale Systems (ScalA)* (Salt Lake City, Utah, USA). IEEE, Los Alamitos, CA, USA, 17–24. doi:10.1109/ScalA.2016.007

[17] Heinz Rutishauser. 1959. Theory of gradient methods. In *Refined iterative methods for computation of the solution and the eigenvalues of self-adjoint boundary value problems.* Mitteilungen aus dem Institut für Angewandte Mathematik, Vol. 8. Birkhäuser, Basel, Switzerland, 24–49. doi:10.1007/978-3-0348-7224-9_2

[18] Yousef Saad. 2003. *Iterative Methods for Sparse Linear Systems* (2nd edition ed.). SIAM, Philadelphia,PA,USA. doi:10.1137/1.9780898718003.ch4

[19] Manasi Tiwari and Sathish Vadhiyar. 2020. Pipelined Preconditioned Conjugate Gradient Methods for Distributed Memory Systems. In *2020 IEEE 27th International Conference on High Performance Computing, Data, and Analytics (HiPC)* (Pune, India). IEEE, Piscataway, NJ, USA, 151–160. doi:10.1109/HiPC50609.2020.00029

[20] Manasi Tiwari and Sathish Vadhiyar. 2021. Pipelined Preconditioned s-step Conjugate Gradient Methods for Distributed Memory Systems. In *2021 IEEE International Conference on Cluster Computing (CLUSTER).* IEEE, Piscataway, NJ, USA, 215–225. doi:10.1109/Cluster48925.2021.00061

[21] Sivan Toledo. 1995. *Quantitative Performance Modeling of Scientific Computations and Creating Locality in Numerical Algorithms.* Ph. D. Dissertation. Massachusetts Institute of Technology, Department of Electrical Engineering and Computer Science.

[22] The Trilinos Project Team. 2025 (acccessed July 29, 2025). *The Trilinos Project Website.* https://trilinos.github.io